Three Methods for Optimizing Single-Shortest Path Routing

Mateusz Dzida
Michał Zagożdżon
and Mateusz Żotkiewicz
Warsaw University of Technology
Warsaw, Poland
Email: mdzida@tele.pw.edu.pl

Mats Petter Pettersson
and Michał Pióro
Lund University
Lund, Sweden
Email: matsp@cs.lth.se

Michael Duelli
and Michael Menth
University of Würzburg
Würzburg, Germany
Email: duelli@informatik.uni-wuerzburg.de

Abstract—**Intra-domain routing in IP networks is based on the shortest path principle by assigning administrative weights (costs) to links. The resulting least-cost paths determine routes between pairs of routers. If several such equal-cost paths exist between a pair of routers, it may not be clear which of them is actually used to route traffic. This makes it difficult to predict the network traffic flow distribution. Therefore, the selected link costs should assure uniqueness of the shortest paths. On top of that, the link costs can be optimized with respect to some traffic objective. The resulting optimization problem, referred to as SSPP, turns out to be $\mathcal{NP}$-hard. SSPP can be formulated as a mixed-integer programming problem and, as such, solved with branch-and-bound (B&B). In this paper, we consider three methods for SSPP. Two of them are exact methods based on B&B, namely branch-and-cut and constraint programming. Since the exact solutions of SSPP may require excessive computation time and may not always be effective when applied to practical networks, we also study a fast heuristic method. Finally, in a numerical study, we compare the effectiveness of the three approaches.**

I. Introduction

Packet forwarding in IP networks depends on the destination address of the packets. Upon arrival of a packet in a router, the longest match of the address with the prefixes in the router forwarding table is determined, and the corresponding entry determines the interface to which the packet is sent. Forwarding tables are typically determined using the data from a distributed routing protocol of an *autonomous system* (AS) of the Internet. Examples of such shortest path routing protocols are *Open Shortest Path First* (OSPF) or *Intermediate System-to-Intermediate System* (IS-IS). Intra-domain routing protocols usually rely on administrative link weights considered as virtual link costs. The protocols determine the entries of the forwarding tables so that traffic is forwarded on the least-cost paths. Thereby, the cost of a path is the sum of the costs of its links: this is known as the shortest-path principle. When there are several equal-cost paths between two routers, two options can be applied to cope with this issue.

- Packets are forwarded to the interface with the lowest number [1, Section 7.2.7] (or a similar rule is used). Since this rule is not always implemented in practice, the choice of the path over which the traffic is routed can become random. Thus, the exact flow distribution in the network cannot be predicted. This may cause problems in capacity planning and lead to problems with assuring quality of service (QoS), hence.
- Traffic is equally distributed to all outgoing interfaces of a router that lead to a shortest path. This is called *equal-cost multipath* (ECMP) routing. However, in ECMP per-flow load balancing is required to avoid re-ordering of packets. Balancing flows of different size may be difficult [2].

This work is part of the project "RESILIENT-IP" which was funded by the FP6 Network of Excellence Euro-FGI. The authors alone are responsible for the content of the paper.

As both options have drawbacks, it is reasonable to avoid equal-cost paths and enforce unique shortest paths through a properly designed system of administrative weights.

Traffic engineering with IP routing consists in modifying the administrative link weights that control the layout of the shortest paths. Weight modifications should lead to path patterns optimizing certain objectives. Optimization of different objective functions has been studied in the literature. For example, the maximum $\rho = \max_{e\in\mathcal{E}} \rho_e$ of link utilization ρ_e ($e \in \mathcal{E}$) can be minimized. In [3], [4], excessive link utilization is penalized by introducing a piece-wise linear increasing convex function $\Phi(\rho_e)$ and minimizing the sum of penalties $\sum_{e\in\mathcal{E}} \Phi(\rho_e)$. Resilient routing requires that the minimization is performed for all links of the network both for the failure-free scenario and for considered failure scenarios (see [5], [6], [7], [8], [9], [10]). Additional constraints for multi-layer traffic engineering can also be taken into account [11].

In this paper we study three methods for optimization of single-shortest path (SSP) routing patterns in order to minimize the maximum link utilization ρ. The underlying problem is $\mathcal{NP}$-hard (see [3] and [12]) and has been approached with stochastic heuristics such as tabu-search [3], [10] or evolutionary algorithms [13], [14], [15], and with exact methods [16], [17], [9], [18], [19], [20].

To obtain an optimal set of link weights, we formulate a *mixed-integer programming* (MIP) problem called SSPP (SSP Problem), and solve it by two enhancements of the *branch-and-bound* (B&B) approach, namely *branch-and-cut* (B&C) [12], [19], [21] and *constraint programming* (CP) [22], [23]. These enhancements aim at finding exact solutions and may require excessive computation time already for medium-size network instances. Therefore, we also present a simple *"hill hopping"* heuristic (HH) [11]. We explain our methods in detail and apply them to a set of different problem instances. We compare the quality of their results and their computation time.

The paper is organized as follows. In Section II, we give an MIP formulation of SSPP. Section III introduces the three different solution methods. In Section IV, we compare the effectiveness of the three methods on a set of network instances. Conclusions are given in Section V.

II. SSPP: Basic Problem Formulation

An AS network is modeled by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of nodes $\mathcal{V}$ and the set of links $\mathcal{E}$. The originating node of link $e \in \mathcal{E}$ is denoted by $a(e)$, and the terminating node by $b(e)$. The notations $\delta^+(v)$ and $\delta^-(v)$ stand for the sets of all links originating and terminating, respectively, at node $v \in \mathcal{V}$, i.e., $\delta^+(v) = \{e \in \mathcal{E} : a(e) = v\}$ and $\delta^-(v) = \{e \in \mathcal{E} : b(e) = v\}$. Link capacities c_e, $e \in \mathcal{E}$, are assumed to be given, as well as (deterministic) traffic demands between the nodes. Each traffic demand is represented by a pair of nodes and a traffic volume that must be carried by the network. The volume of traffic generated at node $v \in \mathcal{V}$ and destined to node $t \in \mathcal{V}$ is given by h_{vt}. Link capacities and demand volumes are expressed in the same units of bandwidth.

A (somewhat complicated) MIP formulation of SSPP is as follows:

$$\textbf{minimize} \quad \rho \qquad \text{(1a)}$$

subject to

$$\begin{array}{lll}
\sum_{e\in\delta^+(v)} x_{et} - \sum_{e\in\delta^-(v)} x_{et} = h_{vt} & v, t \in \mathcal{V} & \text{(1b)}\\
\sum_{e\in\delta^-(t)} x_{et} = \sum_{v\in\mathcal{V}\setminus\{t\}} h_{vt} & t \in \mathcal{V} & \text{(1c)}\\
\sum_{t\in\mathcal{V}} x_{et} \le c_e \rho_e & e \in \mathcal{E} & \text{(1d)}\\
\rho \ge \rho_e & e \in \mathcal{E} & \text{(1e)}\\
r_{b(e)t} + w_e - r_{a(e)t} \ge 1 - u_{et} & t \in \mathcal{V},\ e \in \mathcal{E} & \text{(1f)}\\
r_{b(e)t} + w_e - r_{a(e)t} \le M(1 - u_{et}) & t \in \mathcal{V},\ e \in \mathcal{E} & \text{(1g)}\\
r_{tt} = 0 & t \in \mathcal{V} & \text{(1h)}\\
\sum_{e\in\delta^+(v)} u_{et} \le 1 & v, t \in \mathcal{V} & \text{(1i)}\\
x_{et} \le u_{et} M & t \in \mathcal{V},\ e \in \mathcal{E}. & \text{(1j)}
\end{array}$$

SSPP uses a big constant M and the following variables:

- $\boldsymbol{x} = (x_{et} \ge 0 :\ e \in \mathcal{E},\ t \in \mathcal{V})$: vector of continuous flow variables; x_{et} denotes the total flow destined to node t realized on link e
- $\boldsymbol{u} = (u_{et} \in \{0,1\} :\ e \in \mathcal{E},\ t \in \mathcal{V})$: binary vector of routing variables; $u_{et} = 1$ if, and only if, link e is on a shortest path to destination t
- $\boldsymbol{w} = (1 \le w_e \le W :\ e \in \mathcal{E})$: vector of continuous variables representing link weights (W - maximum weight)
- $\boldsymbol{r} = (r_{vt} \ge 0 :\ v, t \in \mathcal{V})$: vector of variables representing lengths of the shortest paths; r_{vt} denotes the length of the shortest path from node v to destination t, calculated according to link weights $\boldsymbol{w}$.

The variables are supposed to fulfil the following relations:

- $u_{et} = 0$ implies that $x_{et} = 0$
- $r_{vt} = \sum_{e\in\mathcal{P}} w_e$, where $\mathcal{P}$ is a shortest path from v to t ($u_{et} = 1$ for all $e \in \mathcal{P}$); $r_{tt} = 0$ for each $t \in \mathcal{V}$.

Formulation (1) specifies a multi-commodity flow optimization problem in aggregated node-link notation (cf. [12]). Constraints (1b)–(1c) express the aggregated flow conservation conditions for the flow variables $\boldsymbol{x}$.

Constraints (1d)–(1e) force that the maximum over all normalized link utilization levels $\rho_e, e \in \mathcal{E}, \rho$ is expressed by variable ρ which is minimized through objective (1a).

The quantity $r_{b(e)t} + w_e - r_{a(e)t}$ in constraints (1f)–(1g) measures the difference between the length of the shortest path from $a(e)$ to t (given by $r_{a(e)t}$) and the length of the shortest path from $a(e)$ to t necessarily traversing link e (for the latter, the length of the sub-path from $b(e)$ to t is determined by $r_{b(e)t}$). Note that link e is on a shortest path to node t if, and only if, $r_{b(e)t} + w_e = r_{a(e)t}$ – this condition is enforced by constraints (1f)–(1h). Hence, the routing vector $\boldsymbol{u}$ determines the shortest paths according to the weight vector $\boldsymbol{w}$.

Constraint (1i) enforces that the shortest path between each pair of nodes is unique, and constraint (1j) makes sure that traffic is not routed on the links that do not belong to the shortest paths. Namely, constraint (1i) assures that for each node $v \in \mathcal{V}$ there is at most one outgoing link that belongs to the shortest path to destination $t \in \mathcal{V}$, while constraint (1j) enforces traffic destined to node t to use only the links $e \in \mathcal{E}$ allowed by the routing configuration specified by vector $\boldsymbol{u}$ (i.e., the links with $u_{et} = 1$). Thus, both constraints assure a single shortest path routing pattern.

The MIP formulation (1) was invented by A. Tomaszewski for the ECMP version of the considered problem in 2000 (see Section 7.2.1 in [12] and references there). Analogous formulations were published in the literature (see [17], [9], [24], [25]). In [20], a non-linear version of the considered problem is described. Although the above formulation is among the best in terms of the number of involved variables and the quality of the lower bounds provided by its linear relaxation, a direct use of standard MIP solvers to SSPP can fail already for rather small networks with, say, $V = 10$ nodes.

III. Resolution Methods for SSPP

A. Branch-and-Cut

Problem SSP, as any other MIP problem, may be approached with B&B. B&B is a general optimization method that systematically explores the problem solution space through the so called *branching*. The method excludes subregions of the space that can be proved not to contain any optimal solution (*bounding*). In SSPP, variables $\boldsymbol{u}$ are binary and all other variables are continuous. Hence, only variables $\boldsymbol{u}$ are subject to branching. Initially, all $u_{et},\ e \in \mathcal{E}, t \in \mathcal{V}$, are relaxed, i.e., assumed to be continuous in the range $[0, 1]$. The current set of the relaxed variables $\boldsymbol{u}$ is denoted by $\mathcal{U}$; hence, initially, $\mathcal{U} = \mathcal{E} \times \mathcal{V}$. In further steps of the B&B algorithm, some variables from $\boldsymbol{u}$ will be fixed to 0, and some to 1. The corresponding current sets of variables will be denoted by $\mathcal{U}_0$ and $\mathcal{U}_1$, respectively. (Initially, $\mathcal{U}_0 = \mathcal{U}_1 = \emptyset$.) It will always hold that $\mathcal{U} \cup \mathcal{U}_0 \cup \mathcal{U}_1 = \mathcal{E} \times \mathcal{V}$, and the three sets are mutually disjoint. Besides, ρ^{best} is initially set to $+\infty$.

B&B for SSPP is presented in Algorithm 1. The algorithm successively solves linear programming (LP) (sub)problems, being linear relaxations of SSPP determined by the sets $\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1$. Each such relaxed subproblem is denoted by $P(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1)$, and is given by (1) with the binary requirement for $\boldsymbol{u}$ substituted with: $0 \le u_{et} \le 1, (e,t) \in \mathcal{U},\ u_{et} = 0, (e,t) \in \mathcal{U}_0,\ u_{et} = 1, (e,t) \in \mathcal{U}_1$. Certainly, problems $P(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1)$ can be efficiently solved using LP solvers, for example CPLEX. In Algorithm 1, such a solution is returned by procedure $solution(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1, \hat{\boldsymbol{u}}, \hat{\boldsymbol{x}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{w}}, \hat{\rho})$. The value of $\hat{\rho}$ determines the lower bound on the value of the objective function (ρ) that can be attained in the B&B subtree determined by the sets $\mathcal{U}_0$ and $\mathcal{U}_1$. If in vector $\hat{\boldsymbol{u}}$ all entries are binary then the current solution is an optimal solution for the original problem assuming $\mathcal{U}_0$ and $\mathcal{U}_1$. Otherwise, a pair $(e,t) \in \mathcal{U}$ with $0 < \hat{u}_{et} < 1$ is selected for branching, and two new subproblems, with variable u_{et} equal to 0 and to 1, respectively, are created. As this procedure is repeated recursively, all the resulting sub-regions of variables $\boldsymbol{u}$ represented by sets $\mathcal{U}_0$ and $\mathcal{U}_1$ form a tree structure (referred to as *B&B tree*) with the nodes (called *B&B nodes*) represented by triples $(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1)$.

B&B nodes related to feasible solutions (with binary $\boldsymbol{u}$) of (1) are the leafs of the B&B tree. All other nodes, are related to fractional solutions and are subject to further branching. However, branching in a certain node $(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1)$ may not be necessary: this is the case when the current lower bound $\hat{\rho}$ is greater than or equal to ρ^{best}, i.e., to the value of the objective of the currently best feasible (binary) solution of SSPP. The value of ρ^{best} is referred to as *upper bound.*

The upper bound ρ^{best}, initially set to $+\infty$, is updated whenever a leaf of the B&B is encountered. As this in general happens very seldom, the algorithm can be additionally equipped with a heuristic finding a suboptimal feasible solution of SSPP based on the current fractional solution $(\hat{\boldsymbol{u}}, \hat{\boldsymbol{x}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{w}}, \hat{\rho})$. In effect, the current upper bound ρ^{best} is the best feasible solution obtained so far, either as a binary solution of a B&B leaf or by the heuristic. In general, the heuristic is not invoked at every B&B node as this could be too time consuming (the number of visited B&B nodes is usually enormous) with respect to gains related to the value of the upper bound.

In fact, an upper bound heuristic can be an important element of the B&B framework as heuristics providing feasible solutions of good quality can significantly improve the efficiency of B&B. Using problem-dependent user-defined heuristics can improve effectiveness of B&B, because commercial MIP solvers are equipped only with heuristics working for general MIP problems.

The second means to improve B&B is to use the so called *valid inequalities* (or simply *cuts*) – this leads to an enhancement of B&B known as B&C (branch-and-cut). The idea is to insert one or more additional inequalities to problem $P(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1)$ at a B&B node. Such an inequality must not be violated by any integral (feasible) solution, and at the same time must be violated by the current solution $(\hat{\boldsymbol{u}}, \hat{\boldsymbol{x}}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{w}}, \hat{\rho})$ with fractional $\hat{\boldsymbol{u}}$. If properly done, this can substantially strengthen the lower bound $\hat{\rho}$ and, consequently, lead to more frequent bounding. Many authors consider B&C to be potentially the best exact (and also approximate, if we do not require to reach strict optimum) method for resolving MIP problems, SSPP in particular. Valid inequalities based on the so called shortest path routing necessary condition can be found in [26] and [27]. Other types of inequalities are derived along similar lines in [9], [20], [28], [29], [30], [31], [32], [33]. We note that some of such inequalities hold only for networks with undirected links and demands, as for example inequality (12) in [20] or the 3-node condition in [33].

```
Input:   $\mathcal{G}(\mathcal{V},\mathcal{E})$, capacities $\boldsymbol{c}$, demands $\boldsymbol{h}$
  procedure $BBB(\mathcal{U},\mathcal{U}_0,\mathcal{U}_1)$
  begin
  $solution(\mathcal{U},\mathcal{U}_0,\mathcal{U}_1,\hat{\boldsymbol{u}},\hat{\boldsymbol{x}},\hat{\boldsymbol{r}},\hat{\boldsymbol{w}},\hat{\rho})$;   {subproblem}
  if $\mathcal{U} = \emptyset$ or $\forall\ (e,t) \in \mathcal{E} \times \mathcal{V}, \hat{u}_{et} \in \{0,1\}$ then
    if $\hat{\rho} < \rho^{\text{best}}$ then
      $(\rho^{\text{best}}, \boldsymbol{x}^{\text{best}}, \boldsymbol{w}^{\text{best}}) := (\hat{\rho}, \hat{\boldsymbol{x}}, \hat{\boldsymbol{w}})$
    end if
  else
    if $\hat{\rho} \ge \rho^{\text{best}}$ then
      return;   {bounding}
    else
      begin   {branching}
      choose $(e,t) \in \mathcal{U}$ such that $\hat{u}_{et}$ is fractional;
      $BBB(\mathcal{U}\backslash\{(e,t)\},\mathcal{U}_0 \cup \{(e,t)\},\mathcal{U}_1)$;
      $BBB(\mathcal{U}\backslash\{(e,t)\},\mathcal{U}_0,\mathcal{U}_1 \cup \{(e,t)\})$;
    end if
  end if
  end   {procedure}
Output:   $\boldsymbol{w}^{\text{best}}$
```

Algorithm 1: RECURSIVE BRANCH-AND-BOUND FOR SSPP

In our implementation of B&C, we use two types of problem-dependent valid inequalities generated for routing variables $\boldsymbol{u}$. Such inequalities are specified for problems $P(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1)$ with continuous routing variables $u_{et} \in \mathcal{U}$ and must be satisfied by all feasible binary $\boldsymbol{u}$, at the same time cutting off as many fractional $\boldsymbol{u}$ as possible.

We introduce two properties [18], called *transit* and *cycle* (cf. Figure 1), and the corresponding valid inequalities (based directly on the properties of the shortest paths) of the form:

$$\sum_{(e,t)\in\mathcal{I}^1} (1 - u_{et}) + \sum_{(e,t)\in\mathcal{I}^0} u_{et} \ge 1 \qquad (2)$$

where $\mathcal{I}^0, \mathcal{I}^1 \subseteq \mathcal{E} \times \mathcal{V},\ \mathcal{I}^0 \cap \mathcal{I}^1 = \emptyset$.

To explain the transit property consider two shortest paths starting at node s: one destined to node v, and the other destined to node t (as depicted in the left part of Figure 1).

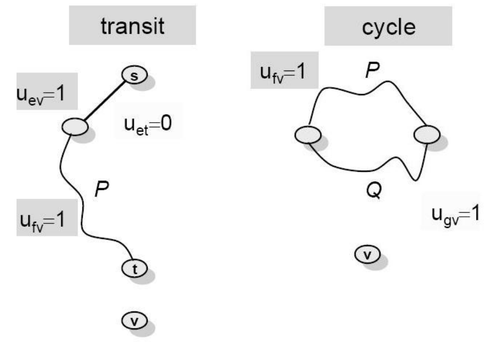

Fig. 1. TWO TYPES OF VALID INEQUALITIES

Assume that the path from s to v traverses t. Then, according to the shortest path condition, the sub-path from s to t for both considered paths must be the same (otherwise, no weight system can generate these two paths as unique shortest paths). Hence, we can formulate the transit property: if t is a transit node on a shortest path from s to v, then all links between s and t on the path to v must belong to a shortest path from s to t. Clearly, if there were a shorter path from s to t then it would have to be used to v as well. Assuming that e is the first link of path $\mathcal{P}_{b(e)t}$ from s to v we can formulate a valid inequality separating vectors $\boldsymbol{u}$ which contradict the transit property:

$$\sum_{f \in \mathcal{P}_{b(e)t}} (1 - u_{fv}) + (1 - u_{ev}) + u_{et} \geq 1. \tag{3}$$

To find the most violated inequality (3) for given s, v, t, and e, it is sufficient to find a shortest path $\mathcal{P}_{b(e)t}$ from $b(e)$ to t using the values $1-u_{ev}$ for the link weights. We note that this inequality is stronger than the one used in [9], because it does not consider the values of the variables defining the shortest paths from t to v, and considers the paths from s to t but not from s to v.

The cycle property expresses a relation between shortest paths to a single destination. Since the values of link weights $\boldsymbol{w}$ are positive, the distances of the consecutive nodes to that destination are decreasing on the shortest path to a destination. Thus, the segments $\mathcal{P}_{st}$ and $\mathcal{Q}_{ts}$ of two such paths cannot form a cycle. The following inequality separates vectors $\boldsymbol{u}$ which contradict this property:

$$\sum_{f \in \mathcal{P}_{st}} (1 - u_{fv}) + \sum_{g \in \mathcal{Q}_{ts}} (1 - u_{gv}) \geq 1. \tag{4}$$

To find the most violated inequality (4) for given s, t, and v, it is sufficient to find a pair of shortest paths, one from s to t, and one from t to s, using the values $1 - u_{ev}$ as link weights.

To generate all violated inequalities of type (3) and (4) for a given vector $\boldsymbol{u}$, it is thus sufficient to determine, for each destination node v, the shortest paths between all pairs of nodes, using values $1 - u_{ev}$ for the link weights; thus, the entire generation has the overall complexity of $\mathcal{O}(|\mathcal{V}|^4)$.

Now, we describe a heuristic that generates feasible solutions (upper bounds ρ^{best}) during our B&C process. The heuristic is based on a method discussed in [16]. The idea of this method reflects the fact that link weights of the form $w_e = 2^e,\ e \in \mathcal{E} = \{1, 2, \dots, E\}$ generate routing patterns with a unique shortest path between each pair of nodes. This is pretty obvious that for these specific link weights all simple paths have mutually different lengths (see [27]).

Suppose that $\mathcal{V} = \{1, 2, \dots, V\}$ (each node is identified by an integer) and consider a somewhat different form of the link weights: $\boldsymbol{w}^o = (w^o_e :\ e \in \mathcal{E})$ (where $w^o_e = \hat{w}_e \cdot 2^{|\mathcal{V}|} + 2^{a(e)}$), formed for a given vector of integral weights $\hat{\boldsymbol{w}} = (\hat{w}_e : e \in \mathcal{E})$. It can be shown that the weight system $\boldsymbol{w}^o$ always generates unique shortest paths.

For $\hat{\boldsymbol{w}}$ we can use a vector of rounded weights obtained from a fractional solution of the linear relaxation of a B&C subproblem $P(\mathcal{U}, \mathcal{U}_0, \mathcal{U}_1)$. Clearly, the numbers $\hat{w}_e \cdot 2^{|\mathcal{V}|}$ can assume large values, exceeding the assumed bound W. To alleviate this, a simple compression of values of vector $\hat{\boldsymbol{w}}$ is made in the following way. Suppose there are K different values in vector $\hat{\boldsymbol{w}}$. Then, the smallest values in $\hat{\boldsymbol{w}}$ are set to 1, the next smallest values—to 2, and so on, until the largest values in $\hat{\boldsymbol{w}}$ are set to K.

Additionally, to better explore the feasible solution space, the described heuristic generates several weight systems $\hat{\boldsymbol{w}}$ by adding some small integers (as $\{0, 1, 2\}$) to the compressed weights $\hat{\boldsymbol{w}}$. Then, vector $\boldsymbol{w}^o$ is computed for each perturbed weight vector $\hat{\boldsymbol{w}}$. Moreover, for each such obtained feasible solution, the heuristic performs several additional steps trying to improve it. The heuristic iteratively identifies all links with the maximum link utilization ρ_e, increases the value of the corresponding original weights, and recomputes the solution. This aims at decreasing loads of the overloaded links (as in general an increased weight of a link makes the number of the shortest paths traversing the link smaller). Similarly, links with the minimum loads are expected to be under-loaded, so the heuristic occasionally decreases the values of original weights related to such links.

B. Constraint Programming

In this section, we describe an exact optimization method for SSPP based on *constraint programming* (CP, see [22] for details). In CP, problems are modeled using variables and constraints. Each variable has a domain (typically discrete) of possible values. Each constraint is imposed on a set of variables, and restricts the combinations of values that can be assigned to these variables. An assignment to all variables which is consistent with every constraint is a feasible solution to the problem.

The process of searching for solutions is systematically organized by traversing a binary search tree, in which each node maintains its own version of the variable domains. The root node uses the original variable domains, while the domains at other nodes may be subsets of the original domains, reflecting search decisions that have been made on the path to the node. Nodes are expanded by doing domain splitting. First, the domains from the expanded node are copied to its children. Then a branching variable, v, is chosen, and finally half of v's domain is removed from the left child and the other half from the right child. A node in which every variable has a single domain defines a full assignment, and if this assignment is consistent with every constraint, then the node represents a solution. In this way, the entire solution space is explored. So, if there is a feasible solution, the method is guaranteed to find it. For optimization problems, it is common to let one variable represent the value of the objective function, and use B&B search (described in Section III-A) to find an optimal solution. Thus, like B&C, CP optimization is *exact* in the sense that it finds provably optimal solutions.

The main difference compared to B&C lies in how the constraints are used to prune the search space. During search, constraint-specific algorithms are used to remove inconsistent values from the domains of the variables on which a constraint is defined. These domain reductions take place in the domains belonging to the current search node. Thus, variable domains are pruned based on one constraint at a time, but with constraint-specific pruning methods, removing values that cannot be part of any solution allowed by the constraint. A reduction of a variable domain based on one constraint may make it possible to make further domain reductions based on other constraints. This process in known as *constraint propagation*, and will go on until a fixpoint has been reached where no more domain reductions can be inferred. At this point, the current search node is expanded, triggering new rounds of constraint propagation at its child nodes.

Here, we describe the main features of the CP-based method. For details, we refer to [34], where a similar version of the method was described. The CP method basically handles capacity constraints in the same way as the B&C method, by embedding a linear program for the multi-commodity flow problem, constraints (1b)-(1d) in a CP constraint. The main difference between the CP and B&C models lies in how the *admissibility* of the path system is enforced, i.e., how constraints are used to make sure that the chosen path set is realizable as single shortest paths w.r.t. some weights.

In [35], a necessary condition for admissibility is described, identifying structures called *valid cycles* that cannot occur in any admissible path set. This condition is conveniently expressed on the routing variables, $\boldsymbol{u}$, defined in Section II. These variables are included in the CP model, together with the *valid cycles constraint* which enforces the condition.

We add an alternative set of routing variables, $\boldsymbol{q}$ to the basic problem formulation (1), where $q_{svt} = 1$, if and only if, traffic from s to t is routed through node v. These variables are linked to the $\boldsymbol{u}$-variables by a channeling constraint, to keep the two representations consistent. On the $\boldsymbol{q}$-variables, we impose the *four-node constraint* which restricts routing decisions for all groups of four nodes. For each such group, the constraint looks at the possible assignments to the 24 $\boldsymbol{q}$-variables that are defined only on these four nodes. Out of the $2^{24} \approx 16 \cdot 10^6$ assignments to these variables, it turns out that only 3225 are possible in admissible path systems, and the four-node constraint will enforce that one of these assignments is chosen. This constraint can be implemented efficiently – it is not necessary to check against all 3225 assignments each time the consequences of a variable assignment is propagated.

We also include in the model the *inverse shortest paths constraint* that embeds an LP formulation of the *inverse shortest paths problem*, i.e., the problem of finding a set of weights that define a given set of paths as shortest paths. This LP formulation can include decisions about partial path systems, so it can be applied at any point in the search. Since this constraint is rather time-consuming, it is not applied at every search node, but it is always run at nodes that correspond to complete assignments, where the path system is fully defined and, thus, a solution candidate. This is necessary, since this is the only constraint that excludes *all* inadmissible solutions.

A standard B&B search only provides an upper bound on the objective function: the best solution found. To estimate the quality of such a solution, it is necessary to also have a lower bound. One way of achieving this is to guess a lower bound, z_{LB}, enforce z_{LB} as an upper bound on the CP model, and solve this problem. If the problem has no solutions, then z_{LB} is a proven lower bound. In the CP method, we use a search scheme in which we run a standard B&B search in parallel with a search that mainly looks for a lower bound. This second search iteratively guesses a bound, and runs a search with a limited number of backtracks using this bound. If a solution was found during an iteration, then the upper bound of the B&B search is updated. If infeasibility was proved for the guessed bound, then it is recorded as a new lower bound. After each iteration, the search is restarted with a new bound guess. The technique of restarting a search after a number of backtracks is known as *random restarts* [36].

C. Hill Hopping Heuristic

In the previous subsections we have presented two exact methods for SSPP. These methods are based on B&B and because of that they can exhibit excessively long runtimes to find provably optimal solutions, even for relatively small networks. Because of $\mathcal{NP}$-hardness of SSPP, B&B is virtually the only exact approach to the problem, so the time issue is unavoidable. Therefore, we need to consider more time-efficient heuristic methods for SSPP that scale with the network size and do not necessarily scan the whole solution space.

In this subsection, we present such a heuristic approach, called *hill hopping* (HH), to tackle the SSP problem in a time efficient way also for large networks. The price paid for time efficiency is that HH does not guarantee optimal solutions. A formal description of HH is given by Algorithm 2.

The gist of the heuristic, introduced in [11], is to first pick up, in a random way, a vector $\boldsymbol{w}$, and then determine, by a simple algorithm, a set of its neighboring vectors that are better than $\boldsymbol{w}$ in terms of the objective function. This step is iteratively repeated until an assumed limit of successive iterations with no improvement is exhausted.

HH starts with a random selection of a weight vector $\boldsymbol{w} \in \{1, 2, \ldots, W\}^{|\mathcal{E}|}$ (weights are assumed to be integers between 1 and W). At this point, vector $\boldsymbol{w}$ becomes the current best weight vector $\boldsymbol{w}^{\text{best}}$. Then, we define a neighborhood set $\mathcal{N}(\boldsymbol{w}^{\text{best}})$ of the current best weight vector $\boldsymbol{w}^{\text{best}}$.

The neighborhood is generated by the so called *Random Neighborhood Generation* (RNG) algorithm, already presented (together with two other algorithms for neighborhood generation) in [11]. RNG has two parameters, C and Δw. It chooses a non-empty subset $\mathcal{E}' \subseteq \mathcal{E}$ with at most C links and modifies their weights by at most Δw. Mathematically,

$$\mathcal{N}(\boldsymbol{w}^{\text{best}}) = \{\boldsymbol{w} : |w_e - w_e^{\text{best}}| \leq \Delta w,\ e \in \mathcal{E}'\}, \quad (5)$$

where $1 \leq w_e, w_e^{\text{best}} \leq W$. Since we solve the SSP problem, we have to eliminate all vectors $\boldsymbol{w}$ which generate multiple shortest paths and, therefore, to use an SSPP-feasible neighborhood $\mathcal{N}_{\text{SSPP}}(\boldsymbol{w}^{\text{best}})$ defined by

$$\mathcal{N}_{\text{SSPP}}(\boldsymbol{w}^{\text{best}}) = \{\boldsymbol{w} \in \mathcal{N}(\boldsymbol{w}^{\text{best}}) : \boldsymbol{w} \text{ is SSPP-feasible}\}. \quad (6)$$

To produce $\mathcal{N}_{\text{SSPP}}(\boldsymbol{w}^{\text{best}})$ we consecutively perturb the current best vector w^{best} and check the SSPP-feasibility of the resulting vectors (i.e., whether the vector generates unique shortest paths for all node pairs) inside the routing calculation. Vectors that generate multiple shortest paths for some node pair are immediately discarded and not considered for the neighborhood.

Maximum link utilization $\rho(\boldsymbol{w})$ is calculated for each new $\boldsymbol{w} \in \mathcal{N}_{\text{SSPP}}(\boldsymbol{w}^{\text{best}})$. The basic feature of HH is the threshold $T, T \geq 1$, allowing a vector $\boldsymbol{w}$ to be a successor of $\boldsymbol{w}^{\text{best}}$, if $\rho(\boldsymbol{w}) \leq T \cdot \rho(\boldsymbol{w}^{\text{best}})$. HH is motivated by the general *hill climbing* local search heuristic (e.g., [11, Section 3.1]). However, HH does not get stuck that easily at a local minimum as it can "hop" over the local minima while wandering around the solution space. The steps of HH are iteratively repeated until the algorithm is unable to improve during N iterations.

```
Input:    $\mathcal{G}, W, C, T, N, \Delta w$
    randomly pick $\boldsymbol{w} \in \{1, 2, \ldots, W\}^{|\mathcal{E}|}$ feasible for SSPP;
    $\boldsymbol{w}^{\text{best}} \leftarrow \boldsymbol{w}$; $\boldsymbol{w}^{\text{new}} \leftarrow \boldsymbol{w}$; $n \leftarrow 0$;
    while $n < N$ do
        randomly select $\boldsymbol{w} \in \mathcal{N}_{\text{SSP}}(\boldsymbol{w}^{\text{new}})$;    {cf.Equation (6)}
        $n \leftarrow n + 1$;
        if $\rho(\boldsymbol{w}) \leq T \cdot \rho(\boldsymbol{w}^{\text{best}})$ then
            $\boldsymbol{w}^{\text{new}} \leftarrow \boldsymbol{w}$;    {to prevent continuous decrease}
            if $\rho(\boldsymbol{w}) < \rho(\boldsymbol{w}^{\text{best}})$ then
                $\boldsymbol{w}^{\text{best}} \leftarrow \boldsymbol{w}$; $n \leftarrow 0$
            end if
        end if
    end while
Output:    $\boldsymbol{w}^{\text{best}}$
```

Algorithm 2: Hill Hopping Heuristic for SSPP

IV. Numerical Results

In this section, we present and compare the results of applying the three methods described in the Section III to a common set of network instances. The instances are characterized in Table I. In each case, every node has a separate demand to all other nodes. The individual demand volumes are in general different, and so are the link capacities.

A. Branch-and-Cut

The B&C method described in Subsection III-A was implemented through the use of Callable Library API 1.2 to program SSPP, and then to resolve it with CPLEX 9.1 (cf. [37]). In effect, the problem is resolved by means of the CPLEX built-in B&C solver. The solver calls our externally defined cut-generation procedure at each B&B node. The procedure uses the current fractional solution obtained at a B&C node and searches for the violated transit and cycle valid inequalities which are generated using a shortest path algorithm. If one (or more) violated valid inequality is found, CPLEX re-optimizes the current linear relaxation of the basic problem with new valid inequalities. The procedure is repeated until no violated valid inequality is found.

B&C invokes the upper bound heuristic procedure every 60 seconds. If a linear relaxation subproblem cannot be solved in this time (what is often the case for large networks), the heuristic is invoked at every visited B&B node. The heuristic extracts the integral weights $hat\boldsymbol{w}$ from the fractional solution provided by the last visited B&B node. Then, the compressed weights are increased by a random integer from the set $\{0, 1, 2\}$. After that, the heuristic assigns a random order of indices to the network nodes $\mathcal{V}$ and calculates the resulting feasible weight vector $\boldsymbol{w}^o$ ($w_e^o = \hat{w}_e \cdot 2^{|\mathcal{V}|} + 2^{a(e)} : e \in \mathcal{E}$). Further, the heuristic continues to improve each such solution $\boldsymbol{w}^o$ in $|\mathcal{E}|$ steps. Weights corresponding to the most heavily loaded links are increased with probability 0.9, and the values of weights corresponding to the least heavily loaded links are decreased with probability 0.1.

The experiments were performed on a machine running Windows XP on an Intel Pentium 4 (3.0 GHz) processor with 1.96 GB of RAM. The CPU was used up to 50%.

B. Constraint Programming

The CP-based method was implemented in using the constraint solver JaCoP [38], a constraint solver implemented in Java. The embedded linear programs were solved using lp_solve. The experiments were performed on a machine running Linux on an Intel Pentium 4 3.0 GHz processor with 1 GB of RAM.

To improve the performance of CP, the constraint model would have to be stronger. Thus, a more thorough pruning of the search space would have been allowed. One weakness of the CP model, compared to B&C, is that the admissibility constraints interact poorly with the capacity constraints. The main capacity constraint, based on the multi-commodity flow, does not incorporate any admissibility constraints, so there is no mechanism similar to the cuts of B&C that can strengthen lower bounds based on the admissibility requirement.

For the two labnet problems, the CP based method fails to find lower bounds. The reason is that none of the randomly restarted iterations has guessed a bound for which the search space was exhausted. One way of achieving a lower bound in such a situation would be to use the lower bound given by the multi-commodity flow constraint at the root of the search

TABLE I
NETWORK DATA AND RESULTS FOR THE THREE METHODS APPLIED ON THE NETWORK INSTANCES SORTED BY CATEGORIES

	Number of		Lower bound / estimated		Upper bound / best found			Time (s)			Gap (%)	
Network	nodes	links	B&C	CP	B&C	CP	HH	B&C	CP	HH	B&C	CP
network_6	6	18	1.2414	1.2414	1.2414	1.2414	1.2414	0.2	1.2	1.0	0.00	0.00
network_8	8	24	1.3134	1.3134	1.3134	1.3134	1.3134	3.8	8.6	2.2	0.00	0.00
network_10	10	30	1.2014	1.2014	1.2014	1.2014	1.2014	25.8	39.0	4.6	0.00	0.00
network_12	12	36	1.0400	1.0451	1.1310	1.1310	1.1310	10800.0	10800.0	11.8	8.04	7.59
network_14	14	42	1.0000	0.8512	1.2748	1.9901	1.2714	10800.0	10800.0	21.8	21.56	57.23
network_16	16	48	1.0064	1.2356	1.2832	1.3746	1.2832	10800.0	10800.0	28.5	21.57	10.11
artificial_6n	6	28	0.8846	0.8846	0.8846	0.8846	0.8846	0.3	0.6	1.1	0.00	0.00
polska_12n	12	36	0.9955	0.9955	0.9955	0.9955	0.9955	18.6	13.2	8.0	0.00	0.00
polska_28n	28	80	0.9942	0.9671	1.0088	1.1468	1.0000	10800.0	10800.0	160.7	1.45	15.67
cost239_defaultTM	11	52	0.4863	0.3682	0.6274	0.8820	0.6274	10800.0	10800.0	8.5	22.49	58.25
cost239_ecmpScaledTM	11	52	0.5135	0.3888	0.6625	0.9314	0.6625	10800.0	10800.0	8.6	22.49	58.25
cost239_sprScaledTM	11	52	0.3043	0.2304	0.3926	0.5519	0.3926	10800.0	10800.0	8.6	22.49	58.25
geant_defaultTM	19	60	0.2972	0.2958	0.3421	0.4944	0.3456	10800.0	10800.0	67.2	13.13	40.17
labnet_ecmpScaledTM	20	106	0.4155	0.0000	0.4477	0.8055	0.4516	10800.0	10800.0	172.2	7.19	100.00
labnet_sprScaledTM	20	106	0.2836	0.0000	0.3055	0.5499	0.3083	10800.0	10800.0	164.2	7.17	100.00
nobel_ecmpScaledTM	28	82	0.5605	0.4664	0.6042	0.9968	0.5651	10800.0	10800.0	281.2	7.23	53.21
nobel_sprScaledTM	28	82	0.5664	0.4713	0.6106	1.0073	0.5711	10800.0	10800.0	263.9	7.23	53.21

tree. However, it is likely that this bound would be weak, for the same reason that the current method fails.

C. Hill Hopping Heuristic

For the evaluation of HH, we used a Java tool implemented for [11] with extended functionality specific to SSPP problem. The application ran on a PC with an Intel Core2 CPU 6600 with 2.4 GHz and 1 GB of RAM using Linux. Only one single CPU core was used to run HH, so the second core of the CPU was not used. Full capacity of the utilized CPU was used whereas RAM was not a limiting factor ($\leq 3\%$ were used).

Each weight vector is an element of $\{1, 2, \dots, W\}^{|\mathcal{E}|}$ so the probability of generating a weight vector which induces multiple shortest paths decreases with increasing W. Hence, we set W to 100, a value higher than in [11]. The maximum number N of consecutive unsuccessful iterations was set to 5000. The parameter C for the maximum cardinality of the set $\mathcal{E}' \subseteq \mathcal{E}$ of the links for which the weights are modified was set to 5. The maximum change Δw in a single link weight was set to 5. Since we choose a weight vector $\boldsymbol{w}$ for initialization of the process at random, we perform 20 runs with different random seeds for all networks in Table I. The best result out of 20 runs and the average time per run is shown in Table I.

Table I reveals that HH is capable of solving large networks in a reasonable time. Nevertheless, the heuristic occasionally is not able to solve SSPP faster than the exact approaches, especially for small networks, because of the large number of weight vectors that always have to be examined, the more that some weight vectors are rejected due to multiple shortest paths. Calculation times of HH could be reduced by decreasing N, but this would increase the probability that the heuristic gets stuck in a local optimum. In contrast to the exact approaches, the HH heuristic does not estimate the lower bound.

D. Discussion

The main merit of the comparison presented in Table I is three-fold.

First, to our best knowledge, no such exhaustive numerical comparison of different optimization approaches to SSPP has been published so far.

Second, we compare two substantially different exact methods, B&C and CP (both based on B&B). As both methods are aimed at exploring the entire solution space, they are usually not able to find a (provable) optimum within a given 3-hour time limit already for medium size networks. In fact, in many cases the ultimately best solution is found quite early in the resolution process, still the methods are not able to assess the optimality in a reasonable time. Observe that within the 3-hour time interval, B&C typically yields better results than CP.

Finally, we compare the exact approaches with a heuristic (HH). Contrary to B&C and CP, HH delivers results of seemingly good quality in a reasonable time, also for large networks. For small networks the quality of the HH results is validated through the exact methods. In general, there is no guarantee for the optimality of the HH results (it does not provide the gap, i.e., the difference between the best upper and lower bound, either). Nevertheless, the HH results are usually as good as, and frequently even better than, the upper bounds achieved by the exact methods.

V. CONCLUSIONS

We have presented three methods for optimizing administrative link weights for intra-domain routing with single-shortest paths (SSP), assuming minimization of the maximum link utilization as the optimization objective. The SSP problem (SSPP) is $\mathcal{NP}$-hard. It can be formulated as a mixed-integer program (MIP). Branch-and-cut (B&C) and constraint programming (CP) are applied to solve the MIP to yield exact results through a systematic search through the solution space, while the third investigated method HH (hill hopping) is a heuristic that explores the solution space randomly.

The comparison of the three methods shows that B&C and CP find, as they should, the same optimal results for small networks. However, both methods are not able to complete computation in the 3 hours time limit even for medium-size networks. In the considered cases, B&C shows better performance than CP. The HH heuristic is fast and often finds optimal solutions for small networks, i.e., in the cases when exact algorithms are able to finish their computation and the optimal solution is known. For medium-size networks, solutions of HH are often as good as, and frequently even better than, the upper bounds (i.e., the best feasible solutions of SSPP) obtained with the exact methods.

The exact approaches can most likely be improved by adding new upper-bound heuristics, valid inequalities, and constraint propagation procedures, so their range in finding optimal solutions can be extended. One such important improvement would be to use HH as the upper bound heuristic in B&C (and in CP). Besides, the exact approaches can be used as approximate methods, e.g., if solutions 10% off the optimum are of interest. The HH heuristic is a promising approach, still, it has to be validated for more examples. All these issues will be investigated in our future research, which will also be extended to resilient optimization of weight systems robust to link failures and traffic fluctuations.

Acknowledgement

The authors thank Matthias Hartmann, Krzysztof Kuchcinski, and Artur Tomaszewski for enlightening discussions. Also, M. Hartmann provided excellent programming support.

References

[1] D. Oran, "RFC1142: OSI IS-IS Intra-Domain Routing Protocol," Feb. 1990.

[2] R. Martin, M. Menth, and M. Hemmkeppler, "Accuracy and Dynamics of Multi-Stage Load Balancing for Multipath Internet Routing," Glasgow, Scotland, UK, Jun. 2007.

[3] B. Fortz and M. Thorup, "Internet Traffic Engineering by Optimizing OSPF Weights," in *IEEE Infocom*, Tel-Aviv, Israel, 2000, pp. 519–528.

[4] B. Fortz, J. Rexford, and M. Thorup, "Traffic Engineering with Traditional IP Routing Protocols," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 118–124, 2002.

[5] B. Fortz and M. Thorup, "Robust Optimization of OSPF/IS-IS Weights," in *International Network Optimization Conference (INOC)*, Paris, France, Oct. 2003, pp. 225–230.

[6] A. Nucci, B. Schroeder, S. Bhattacharyya, N. Taft, and C. Diot, "IGP Link Weight Assignment for Transient Link Failures," in 18^{th} *International Teletraffic Congress (ITC)*, Berlin, Sep. 2003.

[7] D. Yuan, "A Bi-Criteria Optimization Approach for Robust OSPF Routing," in 3^{rd}*IEEE Workshop on IP Operations and Management (IPOM)*, Kansas City, MO, Oct. 2003, pp. 91 – 98.

[8] A. Sridharan and R. Guerin, "Making IGP Routing Robust to Link Failures," in *IFIP-TC6 Networking Conference (Networking)*, Ontario, Canada, May 2005.

[9] L. D. Giovanni, B. Fortz, and M. Labbe, "A Lower Bound for the Internet Protocol Network Design," in *Proceedings of the International Network Optimization Conference INOC'2005, Lisbon*, 2005, pp. B.2.401–408.

[10] M. Dzida, M. Zagożdżon, and M. Pióro, "Optimization of resilient ip networks with shortest path routing," in *6th International Workshop on Design and Reliable Communication Networks, DRCN 2007, La Rochelle, France*, 2007.

[11] M. Menth, M. Hartmann, and R. Martin, "Robust IP Link Costs for Multilayer Resilience," in *Networking*, Atlanta, GA, USA, 2007.

[12] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufman, 2004.

[13] A. Riedl, "A Hybrid Genetic Algorithm for Routing Optimization in IP Networks Utilizing Bandwidth and Delay Metrics," in *IEEE Workshop on IP Operations and Management (IPOM)*, Dallas, TX, Oct. 2002.

[14] M. Ericsson, M. Resende, and P. Pardalos, "A Genetic Algorithm for the Weight Setting Problem in OSPF Routing," *Journal of Combinatorial Optimization*, vol. 6, pp. 299–333, 2002.

[15] E. Mulyana and U. Killat, "A Hybrid Genetic Algorithm Approach for OSPF Weight Setting Problem," in *Proc. 2nd Polish-German Teletraffic Symposium, PGTS'2002*, 2000, gdansk, Poland.

[16] A. Bley, "Finding Small Administrative Lengths for Shortest Path Routing," in *Proceedings INOC'2005, Lisbon*, 2005, pp. B.1.121–128.

[17] K. Holmberg and D. Yuan, "Optimization of Internet Protocol Network Design and Routing," *Networks*, vol. 43, no. 1, pp. 39–53, 2004.

[18] A. Tomaszwski, M. Pióro, M. Dzida, M. Mycek, and M. Zagożdżon, "Valid Inequalities for a Shortest-Path Routing Optimization Problem," in *International Network Optimization Conference INOC2007, Spa, Belgium*, 2007.

[19] A. Tomaszewski, M. Pióro, M. Dzida, and M. Zagożdżon, "Optimization of Administrative Weights in IP Networks Using the Branch-and-Cut Approach," in *The Second International Network Optimization Conference INOC 2005, Lisbon*, March 2005.

[20] N. Bourquia, W. Ben-Ameur, E. Gourdin, and P. Tolla, "Optimal Shortest Path Routing for Internet Networks," in *Proceedings of the 1st International Network Optimization Conference INOC'2003, Evry-Paris*, 2003, pp. 119–125.

[21] M. Pióro, A. Szentesi, J. Harmatos, A. Jüttner, P. Gajowniczek, and S. Kozdrowski, "On OSPF Related Network Optimization Problems," *Performance Evaluation*, vol. 48, pp. 201–223, 2002, (A preliminary version of this paper appeared in the proc. IFIP ATM IP 2000, Ilkley, England, July 2000).

[22] R. Dechter, *Constraint Processing*. Morgan Kaufmann, 2003.

[23] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[24] A. Bley, M. Grötschel, and R. Wessäly, "Design of Broadband Virtual Private Networks: Model and Heuristics for the B-WiN," in *Proc. DIMACS Workshop on Robust Communication Networks and Survivability, AMS-DIMACS Series, 53*, 1998, pp. 1–16.

[25] P. Broström and K. Holmberg, "Multiobjective design of survivable IP networks," in *P. Broström PhD Dissertation, Optimization Models and Methods for Telecommunication Networks Using OSPF*. Linkoeping University, 2006.

[26] W. Ben-Ameur and B. Liau, "Design a Reliable Telephone Network," in *Proceedings IFIP Workshop on Traffic Management*. Montreal, 1999.

[27] A. Farago, A. Szentesi, and B. Szviatovszki, "Allocation of Administrative Weights in PNNI," in *Proc. Networks'98*, 1998, pp. 621–625, sorrento, Italy.

[28] E. Gourdin, "Optimizing Internet Networks," *OR/MS Today*, pp. 46–49, 2001.

[29] A. Bley and T. Koch, "Integer Programming Approaches to Access and Backbone IP-network Planning," Tech. Rep. ZIB-Report 02-41, 2002, zuse Institut Berlin.

[30] W. Ben-Ameur and E. Gourdin, "Internet Routing and Related Topology Issues," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 1, pp. 18–49, 2002.

[31] M. Prytz and A. Forsgren, "Dimensioning of a Multicast Network that Uses Shortest Path Routing Distribution Trees," in *M. Prytz PhD Thesis, On Optimization in Design of Telecommunication Networks with Multicast and Unicast Traffic*. Royal Institute of Technology, Stockholm, 2002.

[32] S. Köhler, D. Staehle, and U. Kohlhaas, "Optimization of IP Routing by Link Cost Specification," in 15^{th}, Würzburg, Jun. 2002, pp. 167 – 175.

[33] M. P. Pettersson, R. Szymanek, and K. Kuchcinski, "A CP-LP Approach to Network Management in OSPF Routing," *SAC 2007*, 2007.

[34] ——, "A CP-LP Hybrid Method for Unique Shortest Path Routing Optimization," in *International Network Optimization Conference INOC2007*, Spa, Belgium, 2007.

[35] P. Broström and K. Holmberg, "Valid Cycles: A Source of Infeasibility in OSPF Routing," in *P. Broström PhD Dissertation, Optimization Models and Methods for Telecommunication Networks Using OSPF*. Linkoeping University, 2006.

[36] C. P. Gomes, B. Selman, and H. A. Kautz, "Boosting Combinatorial Search Through Randomization," in *AAAI/IAAI*, 1998, pp. 431–437.

[37] CPLEX, *CPLEX User's Manual*. ILOG, 1999.

[38] K. Kuchcinski, "Constraints-Driven Scheduling and Resource Assignment," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 8, no. 3, pp. 355–383, Jul. 2003.