Virtual private network provisioning in the hose model

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and Networks Laboratory

Virtual Private Network Provisioning in the Hose Model

Maurice Rüegg

DIPLOMA THESIS
DA-2003.02

Winter 2002/2003

Tutor and Supervisor: Prof. Dr. Thomas Erlebach

Contents

Abstract

A virtual private network (VPN) provides private network connections over a publicly accessible shared network. Provisioning quality of service and flexibility, but also saving cost through link multiplexing are some of the main goals when identifying suitable models for VPNs. A very promising approach to achieve these goals is given by the *hose model* introduced four years ago. In this report, we focus on the different possible variants of this model and provide ideas, implementations, and simulation results for many aspects. We formulate and solve the problem of building VPN hoses as well as validating them. To do this, we distinguish between different routing approaches. Previous to this report, many tree routing issues were discussed. We add algorithms and linear programming formulations for unsplittable and splittable routing; for the latter, our algorithm runs in polynomial time. Finally, we compare performance issues of different hose routings and put the hose model in contrast to a more traditional approach of setting up a directly linked VPN.

1 Introduction

A virtual private network (VPN) has become a popular way to set up a private network over an existing physical network. Because of its omnipresence and ease of accessibility, the global Internet is an ideal such network. This project presents ways to provision VPNs as to guarantee quality of service and flexibility while saving cost through link multiplexing. We apply the concept of the *hose model* presented in [DGG+99] and add to the ideas and results found in the last couple of years of research in this area.

Our work depends heavily on theoretical aspects of communication networks, graph theory, and optimization problems solved through linear programming. We look at a VPN as an undirected graph $G = (V, E)$ with a set of nodes V that represents computers, servers, routers, or any other devices that accept, process, or produce traffic in the network over their communication links in E. We consider only small, straightforward examples of such networks with no more than 10 nodes.

Because large parts of this project are of an implementational nature and also because it is our opinion that many problems and algorithms become easier to grasp this way, we often include pseudo code similar in its semantics to C++. Listings of code are restrained to single algorithms or procedures and always explained and described in the accompanying text.

This report starts with an overview of what we mean when talking about the hose model in the following section 2. Examples help to understand the advantages of a hose. Also, the possible variations of provisioning a hose are listed and explained together with an overview of known results.

Section 3 shows ideas, algorithms, and implementational details when checking a given hose. We look at the basic question whether a given routing definition for a hose is valid on all links and does not violate any capacity constraints. We see that, in order to answer the question, we must consider matchings and flow algorithms in graphs in various forms.

After being able to verify hoses, section 4 talks about building hoses. We give algorithms based on linear programming and integer linear programming, and we see that some algorithms may run faster than others that give better results. Theoretical aspects of building hoses include techniques such as cutting plane algorithms and path stripping.

In section 5, we discuss practical as well as some theoretical results. We see how well the hose model performs compared to a simpler model that uses direct links and to a minimum reservation in terms of overall cost. Then, we look at different variants of hose routing, namely tree, unsplittable, and splittable routing and discuss their performance. Finally, there's a not so obvious difference between the representation of the network as a bidirected and as an undirected graph when calculating hoses.

Finally, we recapitulate our findings in section 6.

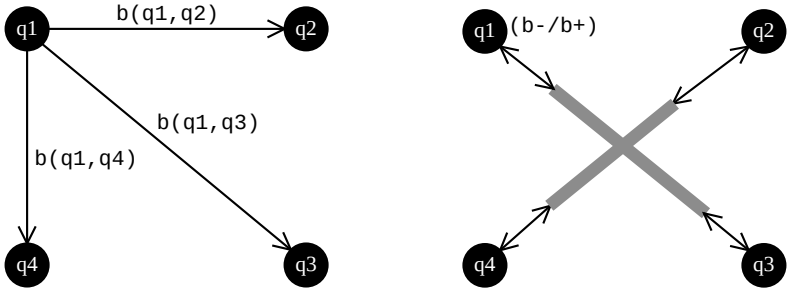
Acknowledgement This project would not have been possible without the help of Prof. Dr. Thomas Erlebach.

2 The Hose Model

A virtual private network is a virtual network set up on top of an existing physical network such as the global Internet. It acts like a separate, private network, providing security and privacy while saving the cost of installing physical links as in a traditional private network. In order to implement such a VPN, various techniques such as IPsec, tunneling, and special routing protocols are used. Resource allocation and management may be done by the classical approach of emulating private lines from one terminal of a VPN to all other terminals. This is called the *pipe model*. A more elegant method called the *hose model* was introduced in [DGG+99].

2.1 Basic Ideas

The hose model may be seen as a VPN service interface as well as a performance abstraction. It simplifies specification aspects of a VPN. A "hose" ensures bandwidth limits and, while it provides a link into the network, it allows to send and receive traffic without the need of predicting point to point loads. Figure 1 illustrates the difference between a pipe VPN and a hose VPN.



(a) Pipe Model (b) Hose Model

Figure 1: (a) The pipe model provides a link and bandwidth reservation from a terminal to all other terminals. (b) The hose model offers the possibility to specify a hose into the VPN with one single ingress and egress bandwidth.

In the hose model, the communication network is given as an undirected graph $G = (V, E)$. Each edge $e \in E$ has a per-unit reservation cost c_e and a capacity limit C_e. In this report, c_e is considered to be equal to 1 while C_e is set to be infinite if we do not explicitly mention different values. The VPN terminals Q are given as a subset of the node set V, $Q \subseteq V$.

As mentioned above, no complete traffic matrix between all terminals has to be specified. Instead, for each $v \in Q$, two bounds b_v^- and b_v^+ are given. The bound b_v^- specifies the maximum amount of traffic that v may receive, and the bound b_v^+ specifies the maximum amount of traffic that v may send.

The idea of the hose model is to reserve bandwidth in the network and to route traffic between VPN terminals such that the reserved bandwidth supports every valid traffic matrix. A traffic matrix is valid if and only if it maps every pair (u, v) of terminals in Q to a non-

negative demand d_{uv} such that, for every v, we have

$$\sum_{u \in Q} d_{uv} \leq b_v^- \qquad \text{and} \qquad \sum_{u \in Q} d_{vu} \leq b_v^+ \tag{1}$$

Let us assume $d_{vv} = 0$ for all $v \in Q$.

The bandwidth reserved for the VPN on edge $e \in E$ is denoted by y_e. The total cost or weight of a reservation is $W = \sum_{e \in E} c_e \cdot y_e$, and the goal is to find a reservation of minimum cost.

The example in figure 2 shows how the hose model is able to save bandwidth by sharing links in a given network G (figure 2(a)). The three VPN terminals q_1, q_2, and q_3 have symmetric ingress and egress bandwidth requirements of 1, 2, and 2 units, respectively. Figure 2(b) depicts the bandwidth reserved on relevant links of the network when an independent shortest paths approach is used to connect VPN terminals. One may see this as the classical approach used in the pipe model. Here, the shortest path between q_1 and q_2 passes through B, while the shortest path between q_1 and q_3 passes through A. This causes an overall network reservation of 16 units. In figure 2(c), the hose model lets all paths pass through C, resulting in a multiplexing gain and an overall reservation of only 12 units. Note that the path from q_1 to q_2 passing through C in (c) is longer than the path through B in (b). However, since terminal q_1 may not receive or send more than one unit of traffic, the bandwidth reserved on each of the links from q_1 to C is one unit (in each direction) and is shared between two paths.

To recapitulate, VPN provisioning in the pipe model is somewhat simpler since traffic between every pair $(u, v) \in Q$ of terminals is fixed and is input to the provisioning algorithm. The hose model trades off provisioning simplicity for ease of specification (figure 1) and multiplexing gains (figure 2).

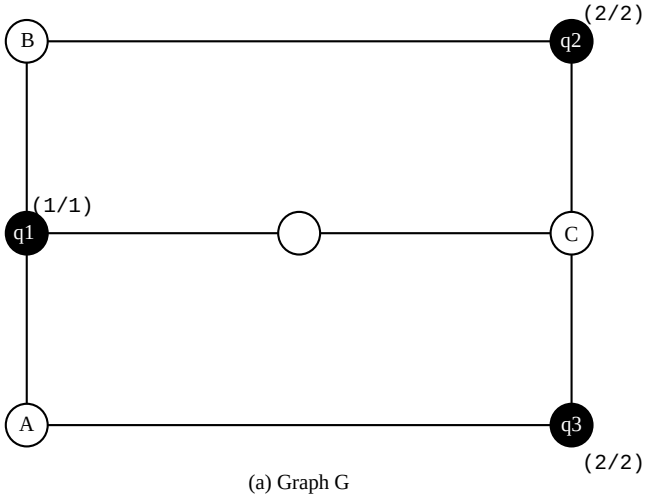
2.2 Variants

The basic problem of finding a reservation of minimum cost in the hose model may be subject to variable conditions. First, ingress and egress bandwidth b_v^-, b_v^+, $v \in Q$ may be defined in three different ways.

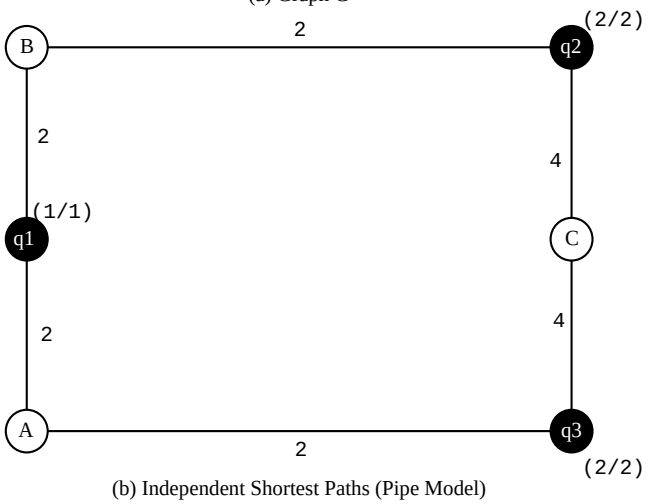
- The symmetric case: $b_v^- = b_v^+ \quad \forall v \in Q$.
- The sum-symmetric case: $\sum_{v \in Q} b_v^- = \sum_{v \in Q} b_v^+$.
- The asymmetric or general case: b_v^- and b_v^+ are arbitrary values.

Additional variations arise from routing constraints.

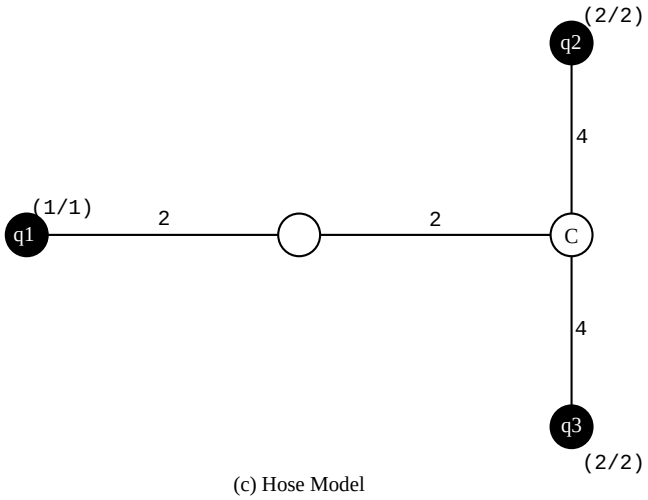
- Tree routing: The edges on which bandwidth is reserved ($y_e \neq 0$) form a tree T. The traffic from $u \in Q$ to $v \in Q$ is routed along the unique path from u to v in T.
- Unsplittable routing: For every pair (u, v) of distinct nodes in Q, the traffic from u to v is routed along a single path P_{uv}. P_{uv} does not depend on the traffic matrix and may also be different from P_{vu}.



(a) Graph G



(b) Independent Shortest Paths (Pipe Model)



(c) Hose Model

Figure 2: Comparing the pipe model (b) to the hose model (c). The hose model is able to share links and save bandwidth.

- Splittable routing: For every pair (u, v) of distinct nodes in Q, the traffic from u to v may be split arbitrarily among several paths. The routing does not depend on the traffic matrix, and paths from u to v may be different from those from v to u.
- Splittable dynamic routing: For every pair (u, v) of distinct nodes in Q, the traffic from u to v may be split arbitrarily among several paths. The routing may depend on the traffic matrix.

Another variant of the problem is obtained if the VPN is required to be fault tolerant. For example, one may require that a VPN has sufficient capacity to support every valid traffic matrix even after an arbitrary link of the network fails ([IRY02]).

In this report, our main emphasis lies on the symmetric and asymmetric cases of tree, unsplittable and splittable routing.

2.3 Survey of Known Results

Before us, several authors have considered the problem of computing a VPN reservation with minimum cost. The following is a short summary of previous work and known results (not necessarily complete; please refer to the original papers).

- [DGG+99] introduces the hose model for VPN provisioning.
- [KRSY01] gives algorithms and results for the problem of computing a reservation with minimum cost
  - for tree routing with infinite capacities, symmetric bounds $b_v^- = b_v^+$ for all nodes $v \in V$, and cost $c_e = 1$ for all edges $e \in E$: a polynomial algorithm for computing the optimal reservation and
  - for tree routing with infinite capacities, asymmetric bounds $b_v^- \neq b_v^+$ for all nodes $v \in Q$, and cost $c_e = 1$ for all edges $e \in E$: It is $\mathcal{NP}$-hard to compute the optimal reservation. The problem may be formulated as an integer linear program. A 10-approximation algorithm may be obtained by solving the linear program relaxation and rounding the fractional solution. Other approximation algorithms are obtained using the primal-dual technique and a breadth-first-search based approach.

  The algorithms are evaluated using simulation experiments.
- [KRSY02] is the journal version of [KRSY01]. In particular, section 5 proposes modified algorithms of VPN reservations for finite capacities.
- [GKK+01] extends the work of [KRSY01] and gives the following results:
  - For asymmetric tree routing with infinite capacities, the approximation ratio of the approach based on the linear program relaxation and rounding is improved to 9.002.
  - For symmetric tree routing with infinite capacities, it is shown that the cost of the optimal tree is at most twice as large as the cost of the optimal reservation in the model with splittable dynamic routing.

  - For symmetric routing with infinite capacities, the optimal cost for the case where each terminal must route its traffic along a tree—while different terminals could use different trees—is equal to the optimal cost for the case where tree routing is used.
  - For asymmetric routing with infinite capacities and $b_v^- = \infty$ for all $v \in Q$, the optimal cost with unsplittable routing is equal to the optimal cost with tree routing and at most twice as large as the optimal cost with splittable dynamic routing.
  - For symmetric routing with finite capacities, it is $\mathcal{NP}$-hard to check whether there is a feasible solution with tree routing or unsplittable routing. A polynomial algorithm is given to compute a solution with unsplittable routing whose cost is within a constant factor of the optimum and that violates edge capacities at most by a constant factor.

- [ILO02] shows that for sum-symmetric tree routing with infinite capacities, the optimal solution may be computed in polynomial time and is within a factor of three of the optimal cost with unsplittable routing. Additionally, a new integer linear program formulation is proposed.

- [dVPSW02] lists spatial and temporal multiplexing aspects of VPN routing (dynamic routing) and discusses the issue of minimizing cost versus reducing congestion in a network.

- [IRY02] considers the problem of making a VPN fault-tolerant for single edge failures. The paper suggests a 16-approximation for the problem of minimizing the total capacity of the edges in backup paths for a given VPN tree reservation.

When adding the results we are going to give in this report, a summary of known results may be given as shown in table 1. References are given as to where in the paper we discuss each case.

ROUTING	ASYMMETRIC	SYMMETRIC	SUM-SYMMETRIC
TREE	**$\mathcal{NP}$-hard** [KRSY01] [GKK+01] section 4.1	**polynomial** [KRSY01] [GKK+01] section 4.1.4	**polynomial** [ILO02]
UNSPLITTABLE	**?** [GKK+01] section 4.3	**?** [GKK+01]	**?**
SPLITTABLE	**polynomial** section 4.2	**polynomial**	**polynomial**
DYNAMIC	***co*-$\mathcal{NP}$-hard** *for directed graphs* [GKK+01]	**?**	**?**

Table 1: Summary of known results.

3 Validating a Hose

From the discussion in section 2, it follows that the hose model provides a simple mechanism for specifying bandwidth requirements in a VPN and enables the utilization of network bandwidth more efficiently. In order to make use of these benefits, efficient algorithms must be devised for provisioning hoses as will be done in section 4. At least as important, we must be able to verify that a hose is valid for every traffic matrix in a given network $G = (V, E)$. Figure 3 shows what data a *hose checker* will need to validate a given hose and how such data is best grouped together. A network representation is best done via a graph with edge capacities and eventually edge costs. Then, a set of VPN terminals must be given and to each terminal v a traffic specification pair of b_v^- and b_v^+. Third, there must be a VPN reservation, meaning the definition of a hose. For tree routing, only edge reservations y_e for every edge e in the tree T are needed. For the more general unsplittable and splittable routing, additional information about routing paths between all pairs of terminals $(u, v) \in Q$ is required. For one possible format of these three sets of data shown in figure 3, see appendix A.2.

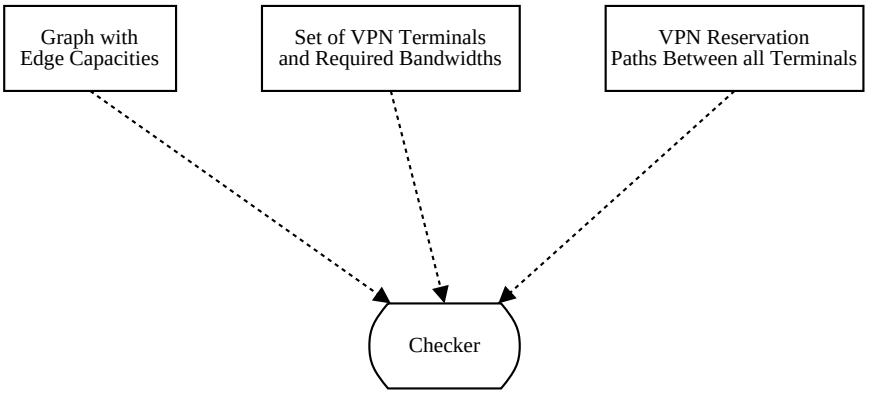


Figure 3: Validating the data of a hose with a checker.

In [ILO02], the problem of validating a given reservation vector y_e on every edge $e \in E$ for the network G and for all valid traffic matrices is identified as being a

Separation Problem Does the network G with capacities y_e support every valid traffic matrix $M = (m_{u,v})$ as stated in (2) where, $m_{u,v,e}$ denotes the traffic from terminal u to terminal v passing through e?

$$y_e \geq \sum_{u \in Q, v \in Q} m_{u,v,e} \tag{2}$$

For all four routing models (tree, unsplittable, splittable, and splittable dynamic routing), the separation problem (2) belongs to $\mathcal{C}o - \mathcal{NP}$, and, if the answer is negative, then there exists a valid traffic matrix M, such that G with capacities y_e does not support M.

The separation problem is solved for a hose with tree routing in section 3.1. In section 3.2 and 3.3, the problem is solved for unsplittable and splittable routing, respectively. It is shown that, for the latter two, approaches using b-matching, maximum flow, and minimum cost flow may be used, while checking tree routing reduces to one single equation.

3.1 Tree Routing

3.1.1 Idea

Dealing with routing in a tree T, or more precisely tree routing in a general graph $T(y)$, validating a given edge reservation vector y_e is not complicated. [GKK+01] and [ILO02] show that for every pair of nodes u and v of a tree routing along the unique path P_{uv}, an edge e only belongs to P_{uv} if u and v belong to different components of $T(y) \setminus e$. Given a valid traffic matrix M, the amount of flow that has to be routed through e is then given as

$$\sum_{u \in Q \cap L_e, v \in Q \cap R_e} d_{uv} + d_{vu} \leq \min\Big\{ \sum_{v \in Q \cap L_e} b_v^-, \sum_{v \in Q \cap R_e} b_v^+ \Big\} + \min\Big\{ \sum_{v \in Q \cap L_e} b_v^+, \sum_{v \in Q \cap R_e} b_v^- \Big\} \quad (3)$$

where (L_e, R_e) are the node sets of the components of $T(y) \setminus e$. It is possible to define a valid traffic matrix M for every edge e such that the inequality holds tight. The meaning of (3) is illustrated in a graph in figure 4.

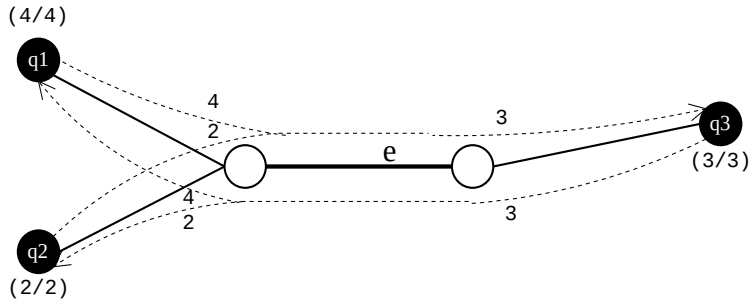


Figure 4: Example illustration of (3).

It follows that a tree reservation vector y_e is feasible, meaning that it supports all valid traffic matrices M, if and only if

$$y_e \geq \min\Big\{ \sum_{v \in Q \cap L_e} b_v^-, \sum_{v \in Q \cap R_e} b_v^+ \Big\} + \min\Big\{ \sum_{v \in Q \cap L_e} b_v^+, \sum_{v \in Q \cap R_e} b_v^- \Big\} \quad (4)$$

for every $e \in E(T(y))$. Thus, a reservation y_e on a tree $T(y)$ is *tight* if for every edge $e \in E(T(y))$ (4) holds with equality. In the example in figure 4, the tight reservation on edge e would be $\min\{(4+2), 3\} + \min\{(4+2), 3\}$, which is equal to 6.

3.1.2 Algorithm and Implementational Aspects

A possible implementation to check a given hose for tree routing is described in listing 1. All edges with a reservation y_e which is greater than 0 must be checked by the formula given in (4).

Note that it is not only possible to check for a feasible solution as done in listing 1, but also, whether a solution is tight as mentioned above.

In order to speed up an implementation of validating a tree hose, it is a good idea to remove all edges e with a reservation $y_e = 0$ from the network T. Because of this, there may be some isolated nodes v with a degree of 0 that may safely be removed, too. However, it is important that no terminal node $v \in Q$ is removed.

```
// Input: tree T, terminals Q, bandwidth bIn and bOut, reservation y
// Output: feasibility of the reservation

forall_edges (e, T)
    hide edge e;
    compute components L_e and R_e of T\e;
    forall (v, Q)
        if (v belongs to L_e)
            bInL[v] += bIn[v];
            bOutL[v] += bOut[v];
        else
            bInR[v] += bIn[v];
            bOutR[v] += bOut[v];
    restore edge e;

    if (y[e] < (min(bInL, bOutR) + min(bInR, bOutL)))
        feasible [e] = false;
    else
        feasible [e] = true;
return feasible [];
```

Listing 1: checkTree()—Checking a tree routing reservation.

The given algorithm for checking tree routing runs in $\mathcal{O}(m + n^2)$, where m is the number of edges and n the number of nodes in the network T because we have m possible edges to hide and then check for every node v in what component of T it is. Even though an optimal algorithm would achieve a running time of $\mathcal{O}(m)$, $\mathcal{O}(m + n^2)$ is good enough because of the only small graphs we consider in our work.

3.2 Unsplittable Routing

As the name implies, unsplittable routing refers to hoses in general graphs $G = (V, E)$, where every pair of terminals $(u, v) \in Q$ is linked over a single, explicitly specified path. This results in edge reservations y_e for all $e \in E$ that must allow for a bandwidth equal to or larger than the sum of capacity requirements of all routings over e. Note that routing paths are completely independent from each other, and a path from u to v may use e while the reverse path from v to u does not.

Since the problem of unsplittable routing in the hose model may be redefined as a b-matching problem, section 3.2.1 gives a short overview of matchings and b-matchings, before an algorithm to validate unsplittable routing is presented in section 3.2.2.

3.2.1 Matchings and b-Matchings

A matching $\mathcal{M}$ in a graph $G_{Bi} = (V, E)$ is a subset of E such that any two edges in $\mathcal{M}$ are disjoint. Most matching problems arise in situations that may be modeled by bipartite graphs. Thus, in the following, it is assumed that G_{Bi} is bipartite.

Example A practical example of a matching $\mathcal{M}$ may be a set of jobs a company offers on one side of G_{Bi} and a number of applicants on the other side. Each candidate has the skills to

perform a certain subset of the jobs. Assign suitable candidates to as many jobs as possible. ■

The above example is called a maximum matching and, if all jobs could be assigned and all applicants employed, a perfect matching. For an illustration, see the bipartite graph on the left of figure 5. There, three applicants would be available for two jobs, and a maximum matching assigns all available jobs (solid edges).

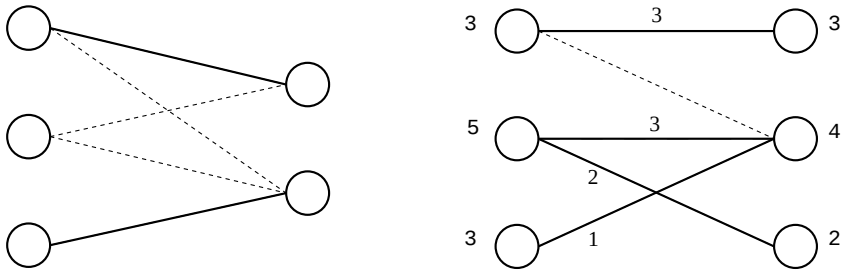


Figure 5: A maximum matching $\mathcal{M}$ (a) and a maximum b-matching $\mathcal{M}_b$ (b).

The right side of figure 5 illustrates the problem of a maximum b-matching $\mathcal{M}_b$, also known as a transportation problem. $\mathcal{M}_b$ is a matching where all nodes v on the left side of G_{Bi} are assigned a supply and those on the right side a demand. The goal is to assign values f to edges e such that the sum of all f becomes maximal. More formally, an instance of $\mathcal{M}_b$ is a graph $G_{Bi} = (V, E)$ and a function $b : V \to \mathbb{N}_0$. The problem is to find a function $f : E \to \mathbb{N}_0$ satisfying $\sum_{v \in \text{adj}(u)} f(\{u, v\}) \leq b(u)$ for all $u \in V$ such that $\sum_{e \in E} f(e)$ is maximized. A perfect b-matching is achieved if $\sum_{v \in \text{adj}(u)} f(\{u, v\}) = b(u)$.

Example A maximum b-matching is desirable when considering the problem of assigning client computers to server stations. Each server has a maximum range of service and a limit on the number of clients that it may support. Clients must each be assigned to a server that is reachable and not fully utilized. Assign suitable servers to as many clients as possible. ■

Algorithms for finding maximum matchings try to extend a given matching $\mathcal{M}$ to a matching $\mathcal{M}'$ with a cardinality that is one higher, i. e. $|\mathcal{M}'| = |\mathcal{M}| + 1$. These improvement algorithms are based on the concept of alternating and augmenting paths via labeling algorithms. A good overview on matchings may be found in chapter 2 of [The]. For b-matchings, see chapter 5.5 of [CCPS98] or [PT00].

The problem of finding a maximum b-matching $\mathcal{M}_b$ may be reduced to a maximum matching problem. We duplicate every node u on the left of G_{Bi} $b(u)$ times and v on the right of G_{Bi} $b(v)$ times. If u and v are connected, we add edges from all nodes $u_1, \ldots, u_{b(u)}$ to $v_1, \ldots, v_{b(v)}$. Figure 6 illustrates this process of reducing $\mathcal{M}_b$ to $\mathcal{M}$ for three nodes. It is obvious that this technique will produce quite large graphs, when b-values become large. Additionally, non-integer b-values present a problem. Thus, in the following sections 3.2.2 and 3.3.1, a different approach of solving $\mathcal{M}_b$ is presented. First, though, the idea of redefining a hose routing as a b-matching $\mathcal{M}_b$ will be presented.

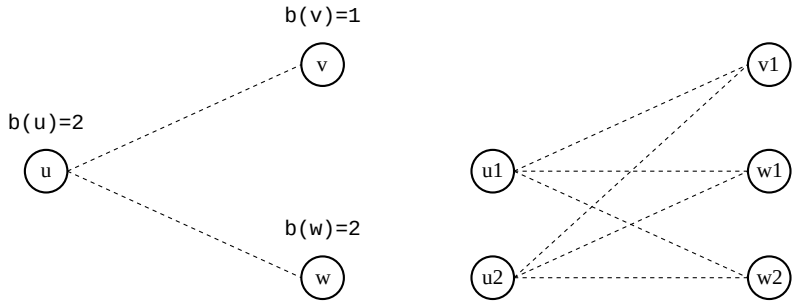


Figure 6: A possible way to solve a b-matching problem: reducing it to a matching problem.

3.2.2 Edge Reservations as b-Matchings

The idea of checking edge reservations y_e for all edges e in a general graph G according to the hose model is based upon a b-matching problem $\mathcal{M}_b$ as illustrated in figures 7 and 8. First, all routing paths that use an edge e are analyzed and their sending and receiving terminals u and $v \in Q$ identified. From this, a bipartite graph $G_{Bi,e}$ is built with all sending terminals in the left and receiving terminals in the right column of $G_{Bi,e}$. Edges represent the paths between any two terminals passing through e. It may well be, that a terminal v is a sending terminal in the left column of $G_{Bi,e}$ as well as a receiving terminal in the right one at the same time.

In order to represent traffic specifications of a path between u and v, every node u and v of $G_{Bi,e}$ may be given a capacity, or more accurately, a supply of b_u^+ and a demand b_v^-. Consequently, an instance of a b-matching $\mathcal{M}_b$ is created. The sum of all f-values assigned to edges of $G_{Bi,e}$ in a maximum b-matching $\mathcal{M}_b$ is then equal to the maximum edge reservation y_e required over e for a given routing. Correctness of this transformation is made clear by understanding that in a tight solution for $\mathcal{M}_b$, there will be no traffic overtaxing any terminal while all possible capacities are being used.

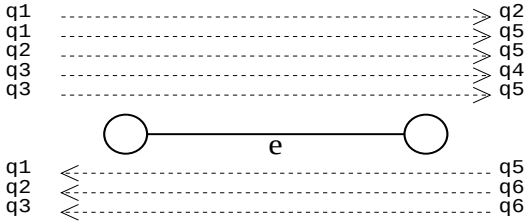


Figure 7: For every edge e in a graph G, find out what routing paths use e.

One possible way to solve a b-matching $\mathcal{M}_b$ by transforming it into a simple matching $\mathcal{M}$ has been mentioned in the previous section 3.2.1. A more efficient solution uses flows to determine the maximum amount of traffic that may pass from all sending terminals u to all receiving terminals v. Therefore, a single source node s is introduced in $G_{Bi,e}$, together with a sink t, resulting in the new graph $G'_{Bi,e}$. In $G'_{Bi,e}$, there are edges from s to all sending terminals u with a capacity of b_u^+. Analogously, $G'_{Bi,e}$ must include edges from all receiving terminals v to t with a capacity of b_v^-. This way, it is assured that s may send no more

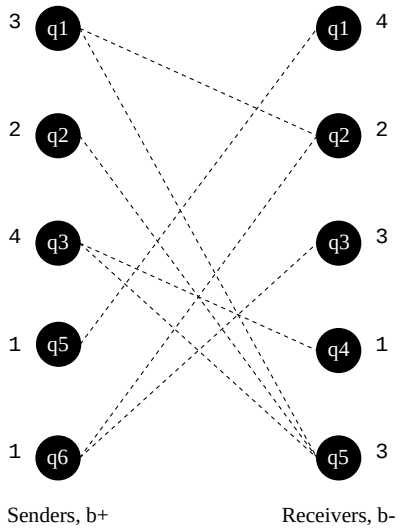


Figure 8: Build a bipartite graph $G_{Bi,e}$ out of the routing information over edge e.

traffic via u, than u itself may send and t may receive no more via v than what v itself would receive. The resulting flow problem is depicted in figure 9. Obviously, any edge between two terminals need not have any capacity limits because the constraints on edge capacities from s and on those to t assure feasible traffic values on all other edges.

A maximum flow algorithm for $G'_{Bi,e}$ will now deliver the maximum possible flow from s to t, equal to the largest edge reservation y_e needed on edge e in G for a given routing. For a maximum flow algorithm description, see [Max]. For a complete discussion of maximum flow algorithms, see chapter 7.10 of [MN99].

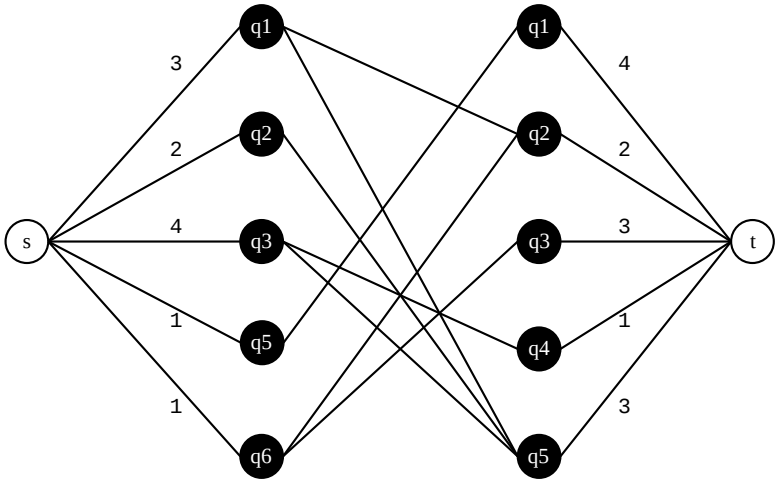


Figure 9: Solving a maximum s-t-flow problem in $G'_{Bi,e}$. Edges between terminals have an infinite capacity.

3.2.3 Implementational Aspects

An implementation of checking an unsplittable routing in the hose model is shown in listing 2. One must only pay attention that the special case when there is no routing between any terminals is not neglected.

```
// Input: graph G, terminals Q, bandwidth bIn and bOut, reservation y,
//        routing R
// Output: feasibility of the reservation

forall_edges (e, G)
    build a bipartite Graph G_Bi_e from R;
    add nodes s, t to G_Bi_e;
    add edges from s and to t to G_Bi_e;
    forall_edges (f, G_Bi_e)
        if (source(f) == s)
            cap[f] = bOut[target(f)];
        else if (target(f) == t)
            cap[f] = bIn[source(f)];
        else
            cap[f] = INFINITY;

    flow = MAX_FLOW(G_Bi_e, s, t, cap);

    if (y[e] < flow)
        feasible [e] = false;
    else
        feasible [e] = true;

return feasible [];
```

Listing 2: checkUnsplit()—Checking an unsplittable routing reservation.

The worst case running time of checking an unsplittable routing reservation depends on two things:

- Construction of a bipartite graph $G_{Bi,e}$ for every edge e of G. — This may depend on how the given routing information is represented (see also appendix A.2). In an uncomplicated approach, all start and end terminals u and $v \in Q$ must be processed for every edge e. Thus, construction of one graph $G'_{Bi,e}$ is achieved in $\mathcal{O}(n^3)$ where n is the number of nodes of G.

- A good max flow algorithm. — Implementational variations are given in chapter 7.10 of [MN99], ranging from $\mathcal{O}(n^2\sqrt{m})$ to $\mathcal{O}(n^3)$ with m the number of edges in G.

3.3 Splittable Routing

Splittable routing is a variant where a routing between terminals u and $v \in Q$ may be split up between multiple paths. Figure 10 gives an example of this. While 10% of traffic from terminal q_1 to q_2 will choose the way over A, 90% will use the one over B. Similarly, routing from q_1 to q_3 is split, with half the traffic using node B and half node C. There is no limit

as to how many paths may exist for a routing between two terminals u and v in a graph G as long as together, they route 100% of traffic. Thus, if a path from u to v uses an edge e of G, there exists a split factor $S(u, v, e)$ with $0 < S(u, v, e) \leq 1$, indicating the percentage of traffic from u to v over e.

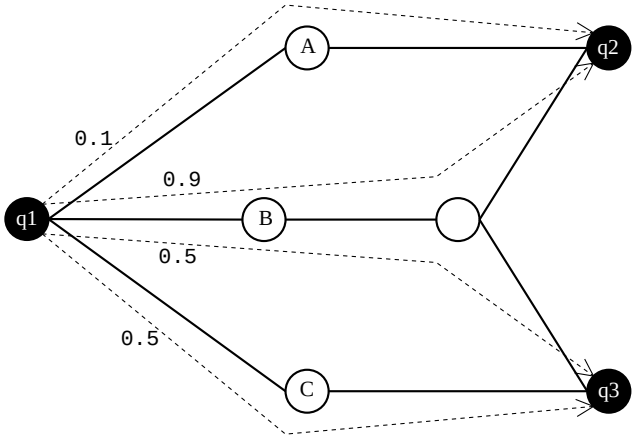


Figure 10: Splittable routing in the hose model.

3.3.1 A Minimum Cost Flow Problem

Checking splittable routing in a hose may be achieved very similarly to unsplittable route checking. As in unsplittable routing, every edge e of G is looked at independently. Figure 11 shows how all routing paths over e are listed. The difference to unsplittable routing (see figure 7) lies in split factors $S(u, v, e)$ given for every path over e.

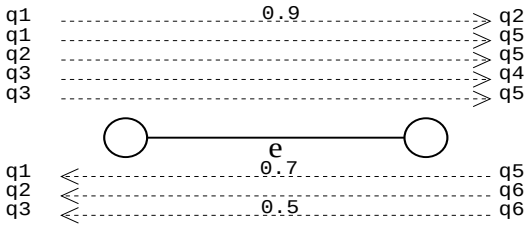


Figure 11: For every edge e in a graph G, find out what routing paths use e and by what percentage (indicated via the split factor). Edges without an indicated split factor are assumed to have one equal to 1.0.

Just as for unsplittable routing, a bipartite graph $G_{Bi,e}$ is built from the given routing information over e, and a start node s and end node t are introduced to get $G'_{Bi,e}$. Edges from s to $u \in Q$ are given capacity values of b_u^+ and edges from $v \in Q$ to t b_v^-. All other edges between start and end terminals have infinity capacity.

The difference and main idea of checking splittable routing is to introduce a cost $c_{Bi,e}$ for every edge in $G'_{Bi,e}$. While edges from s and to t have a cost $c_{Bi,e} = 0$, those between terminals and representing paths receive a *negative* cost according to their split factor. This means that

using edges with higher split factors is cheaper. In figure 12, the resulting minimum cost flow graph $G'_{Bi,e}$ is shown. For a description of minimum cost flow algorithms, see [Min], for a complete discussion, see chapter 7.11 of [MN99].

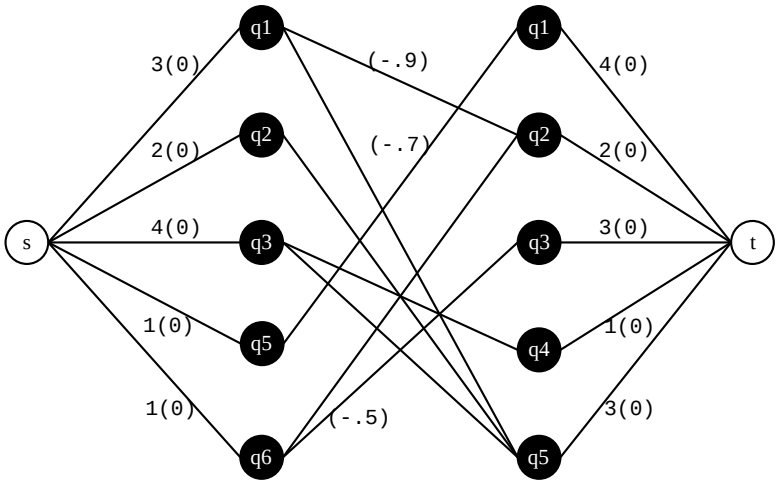


Figure 12: Generating a minimum cost flow graph. Edges between terminals have an infinite capacity. Those without an indicated split factor in parentheses are assumed to have one equal to 1.0.

Checking for large enough edge reservations y_e on all edges e in G, leads to the following theorem:

Theorem 3.1 *There exists a traffic matrix M, generating traffic y_e on edge e of $G \iff$ there is a flow f from s to t in $G'_{Bi,e}$ with cost of $-y_e$.*

Proof Let us split the proof to the above theorem into two parts:

- There exists a traffic matrix M, generating traffic y_e on edge e of $G \Longrightarrow$ there is a flow f from s to t in $G'_{Bi,e}$ with cost of $-y_e$:
  Let $M = (m_{u,v})$ be the matrix that generates traffic y_e from terminal u to v on edge e.
  Let f be a flow defined as

$$\begin{aligned} s \xrightarrow[e]{} u : f(e) &= \sum_{v \in Q} m_{u,v} \\ v \xrightarrow[e]{} t : f(e) &= \sum_{u \in Q} m_{u,v} \\ u \xrightarrow[e]{} v : f(e) &= m_{u,v} \end{aligned} \tag{5}$$

  f is valid because

  - no edge capacity limits C_e in $G'_{Bi,e}$ are violated and
  - flow conservation in $G'_{Bi,e}$ is kept for all nodes $u \neq \{s, t\}$.

The overall cost of a valid flow over edge e may be defined as

$$\sum_{u\in Q}\sum_{v\in Q} m_{u,v}\cdot c(uv) \qquad c(uv)=\begin{cases} 0 & \text{if routing } u,v \text{ does not use } e \\ -c_{Bi,e} & \text{if a fraction } c_{Bi,e} \text{ of } u\to v \text{ uses } e \end{cases} \tag{6}$$

Since equation (6) is also equal to the traffic over e generated by M, this part of the proof is complete.

- There is a flow f from s to t in $G'_{Bi,e}$ with cost of $-y_e \Longrightarrow$ there exists a traffic matrix M, generating traffic y_e on edge e of G:
  Let f be a flow from s to t with cost $-y_e$.
  Define M: $M_{u,v} := f(uv)$
  M is a valid traffic matrix because for every start terminal $u \in Q$

$$f(su) \leq \text{cap}(s\to u) = b_u^+ \tag{7}$$

  and for every end terminal $v \in Q$

$$f(vt) \leq \text{cap}(v\to t) = b_v^-. \tag{8}$$

  Also, M generates traffic y_e on edge e because

$$-y_e = \sum_{u\in Q}\sum_{v\in Q} f(uv)\cdot c(uv) \qquad c(uv)=\begin{cases} 0 & \text{if routing } u,v \text{ does not use } e \\ -c_{Bi,e} & \text{if a fraction } c_{Bi,e} \text{ of } u\to v \text{ uses } e \end{cases} \tag{9}$$

□

3.3.2 Implementational Aspects

As shown above, a minimum cost flow algorithm may find the maximally needed edge capacity y_e for an edge e. Because it is not known what flow will give a minimum cost flow, all possibilities from a minimum flow of 1 to the maximum flow given by a maximum flow algorithm must be considered. However, this may be avoided by a simple trick shown in figure 13. Adding a direct edge e_{dir} from s to t with cost 0 and infinite capacity lets the minimum cost flow find an optimal solution for an infinite flow from s to t. Because of a cost of 0, e_{dir} does not influence the cost of an overall solution while making any flow from s to t possible.

With this trick, an implementation of checking splittable routing becomes very similar to checking unsplittable routing. The minimum cost flow problem generalizes the maximum flow problem (see chapter 7.11 of [MN99]). Listing 3 shows how a graph $G'_{Bi,e}$ must be constructed for every e of G, including capacities and costs for all edges of $G'_{Bi,e}$. With the additional direct edge e_{dir}, a minimum cost flow returns the maximally needed capacity on e. The mentioned infinite flow between s and t may be restrained to a value found by a maximum flow algorithm.

Once more, the worst case running time depends on how efficient the bipartite graphs $G'_{Bi,e}$ for every e in G are constructed. If m is the number of edges and n the number of nodes of $G'_{Bi,e}$, it takes $\mathcal{O}(n^3)$ to check through all pairs of terminals (u,v) for every reservation value. Because for every routing, there are at most m paths from u to v, the resulting running time

```
// Input: graph G, terminals Q, bandwidth bIn and bOut, reservation y,
//        routing R, split factor S[u][v][e]
// Output: feasibility of the reservation

forall_edges (e, G)
    build a bipartite Graph G_Bi_e from R;
    add nodes s, t to G_Bi_e;
    add edges from s and to t to G_Bi_e;
    forall_edges (f, G_Bi_e)
        if (source(f) == s)
            cap[f] = bOut[target(f)];
            cost[f] = 0;
        else if (target(f) == t)
            cap[f] = bIn[source(f)];
            cost[f] = 0;
        else
            cap[f] = INFINITY;
            cost[f] = -S[source(f)][target(f)][f];

    maxflow = MAX_FLOW(G_Bi_e,s,t,cap);
    supply(s) = maxflow;
    supply(t) = -maxflow;

    add edge e_dir between s and t;
    cap[e_dir] = INFINITY;
    cost[e_dir] = 0;

    flow = MIN_COST_FLOW(G_Bi_e,cap,cost,supply);

    if (y[e] < -flow)
        feasible [e] = false;
    else
        feasible [e] = true;

return feasible [];
```

Listing 3: checkSplit()—Checking a splittable routing reservation.

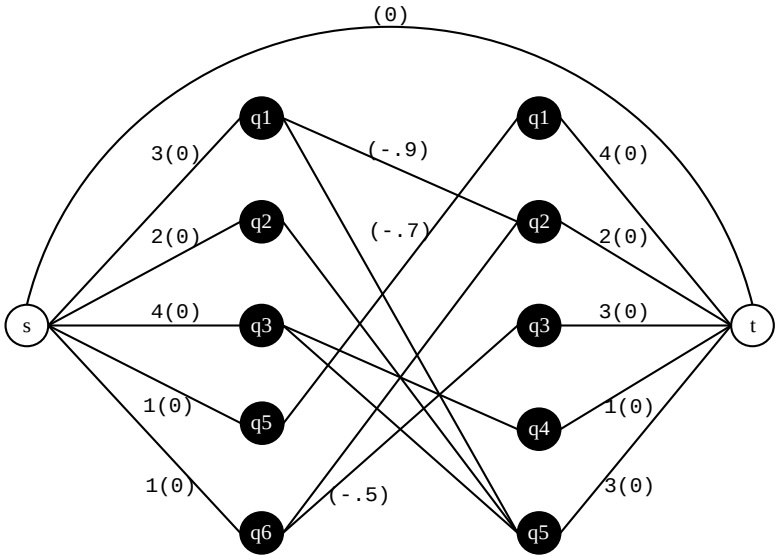


Figure 13: Solving a minimum cost flow problem by adding an edge from s to t with cost 0 and an infinite capacity.

is equal to $(\mathcal{O}(mn^3))$. The implementation of a maximum flow algorithm runs in $(\mathcal{O}(n^2\sqrt{m})$ to $\mathcal{O}(n^3))$. Rendering this problem even more complicated than unsplittable routing is an additional minimum cost flow algorithm that is described in chapter 7.11 of [MN99] and has worst case running time of $\mathcal{O}(m \log U(m + n \log n))$ where U is the largest absolute value of any capacity.

4 Building a Hose

In this section, the main problem of building valid hoses in a network $G(V, E)$ is discussed. We will see that building hoses is very closely related to checking hoses, not only to verify a solution, but also to solve the problem of calculating valid hoses directly as sections 4.2 and 4.3 will show.

To give an overview, figure 14 shows what information a *hose builder* needs in order to generate a possible VPN reservation and path allocation, and also how such a builder works in coordination with a checker that validates a calculated hose.

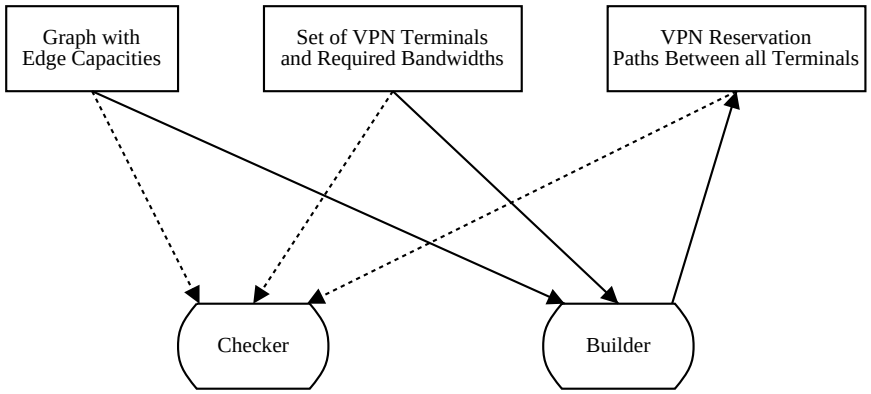


Figure 14: Calculating a hose reservation and validating the data with a checker.

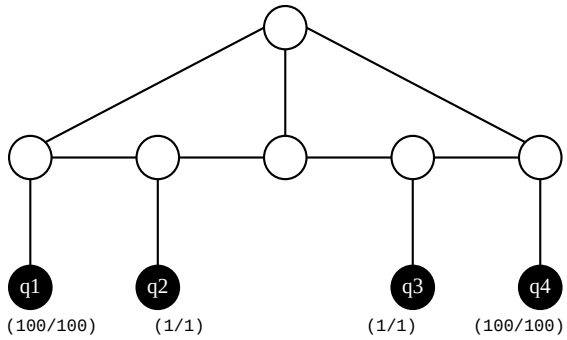
If not explicitly mentioned otherwise, all methods discussed here refer to the more general case of asymmetric routing. First, we will show how tree routing may be solved by integer programming as done in section 4 of [KRSY01] and [KRSY02]. Then we will formulate a new linear program (LP) for splittable and unsplittable routing. While, in the previous section 3, we have discussed checking unsplittable routing before checking splittable routing because the latter derives from the former, we will show a way to solve splittable routing first in section 4.2 and then adapt it to unsplittable routing in section 4.3.

4.1 Tree Routing

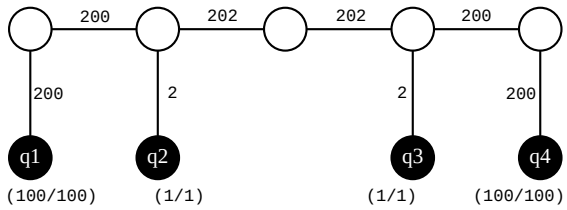
4.1.1 Problem

In [DGG+99], it has been suggested that a Steiner tree may be used to connect the VPN terminals of a network G. However, even though a Steiner tree has the smallest number of links, it may be suboptimal, as shown by [KRSY01] and [KRSY02] and illustrated in figure 15. Figure 15(a) shows a network graph with four terminals q_1, q_2, q_3, and q_4. Bandwidth requirements are given as $b^-_{q_1} = b^+_{q_1} = b^-_{q_4} = b^+_{q_4} = 100$ and $b^-_{q_2} = b^+_{q_2} = b^-_{q_3} = b^+_{q_3} = 1$. Figure 15(b) shows the Steiner tree connecting the terminals, containing 8 edges and a total bandwidth requirement of 1208. Figure 15(c) is an optimal tree hose with 9 edges and a total bandwidth requirement of only 812.

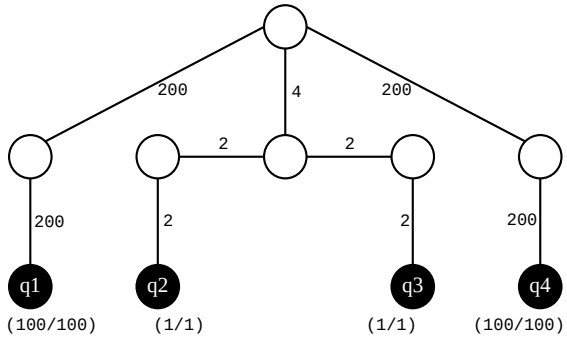
Finding an optimal VPN tree hose for terminals with asymmetric bandwidths is $\mathcal{NP}$-hard because the problem may be shown to be at least as difficult as that of computing a Steiner tree connecting the VPN terminals, and a Steiner tree computation problem for



(a) Graph



(b) Steiner Tree



(c) Optimal VPN Hose

Figure 15: Example of a suboptimal Steiner tree (b) and an optimal tree reservation hose (c).

a set of terminals is $\mathcal{NP}$-hard [Hoc97, KRSY01]. In [KRSY01] and [KRSY02], an integer programming formulation is given to solve the problem, restated briefly in the following.

4.1.2 Integer Programming Formulation

In a tree T, an edge (u, v) lets us identify $L_u^{(u,v)}$ and $R_v^{(u,v)}$ that represent the two components of $T \setminus (u, v)$ containing node u and v respectively. $Q_u^{(u,v)}$ and $Q_v^{(u,v)}$ hold the terminals contained in $L_u^{(u,v)}$ and $R_v^{(u,v)}$. [KRSY01] and [KRSY02] define an edge (u, v) to be *biased* towards u if the two following conditions hold:

1. $\left(\sum_{q\in Q_u^{(u,v)}} b_q^- < \sum_{q\in Q_v^{(u,v)}} b_q^+\right)$ or $\left(\sum_{q\in Q_u^{(u,v)}} b_q^- = \sum_{q\in Q_v^{(u,v)}} b_q^+ \text{ and } Q_v^{(u,v)} \text{ contains } \hat{q}\right)$
2. $\left(\sum_{q\in Q_u^{(u,v)}} b_q^+ < \sum_{q\in Q_v^{(u,v)}} b_q^-\right)$ or $\left(\sum_{q\in Q_u^{(u,v)}} b_q^+ = \sum_{q\in Q_v^{(u,v)}} b_q^- \text{ and } Q_v^{(u,v)} \text{ contains } \hat{q}\right)$

where $\hat{q}$ is a specially marked node. An edge (u, v) is said to be biased if it is biased towards either u or v. En edge that is not biased is said to be *balanced*. A node of T is a *core* node if a balanced edge is incident to it.

They also state and prove that the sum of bandwidths B reserved on a balanced edge (u, v) of T in both directions is $B = \min\{\sum_{q\in Q} b^-, \sum_{q\in Q} b^+\}$ and all balanced edges form a connected component. From this follows that if we delete the balanced edges from T, there is a single core node in each of the resulting connected components. If T contains no balanced edges, there is only one component, all edges must be biased, and one may show that there exists a unique node r in T, such that every edge incident to r is biased away from it. This node r is then considered to be the core node for the component. Thus, a tree T consists of a set of core nodes connected by balanced edges and connected components for each core node r containing only biased edges.

A set S consists of these core nodes and a Steiner tree may connect these nodes with balanced edges. [KRSY01] and [KRSY02] show that to compute the optimal tree T, we need to compute a set of nodes S of minimum cost with respect to edge bandwidth requirements. All nodes in S are then joined into one single supernode, and a breadth first tree rooted at the supernode connects all terminals in T.

The problem of computing the set of nodes S with minimum cost is formulated in (10) as an integer program if we know the identity of one of the nodes in S. Let x_{uv} and z_e be $\{0, 1\}$ variables where x_{uv} is 1 if terminal v is assigned to node u of S and z_e is 1 if edge e belongs to the Steiner tree connecting the nodes in S. $d_G(u, v)$ is the distance or shortest path length between nodes u and v in the network G. Also, let $\delta(\hat{V})$ denote the set of edges crossing sets $\hat{V}$ and $V - \hat{V}$ in G. Suppose we know a priori that node $r \in S$. Then, the solution to the following integer program yields the optimal set of nodes S containing r.

$$\text{minimize} \sum_{u\in V, v\in Q} d_G(u, v) \cdot (b_v^- + b_v^+) \cdot x_{uv} + B \cdot \sum_{e\in E} z_e \tag{10}$$

subject to the following constraints

$$\forall v \in Q : \sum_{u\in V} x_{uv} \geq 1$$

$$\forall \hat{V} \subset V, r \notin \hat{V}, v \in Q : \sum_{e\in\delta(\hat{V})} z_e - \sum_{u\in\hat{V}} x_{uv} \geq 0$$

$$x_{uv}, z_e \in \{0, 1\}$$

The objective function that we minimize is the cost of the set of nodes S. The first constraint states that each terminal v must be assigned to at least one node in S. The second constraint guarantees that nodes in S are connected by a Steiner tree. It achieves this by requiring that if a terminal v is assigned to a node u, then u is connected to r by Steiner tree edges.

The needed reservation y_e on all edges e of T may now be calculated. This may be done analogously to checking for a large enough tree reservation in section 3.1 by removing e from T and looking at the two components (L_e, R_e) of $T \setminus e$:

$$y_e = \min\Big\{ \sum_{v \in Q \cap L_e} b_v^-, \sum_{v \in Q \cap R_e} b_v^+ \Big\} + \min\Big\{ \sum_{v \in Q \cap L_e} b_v^+, \sum_{v \in Q \cap R_e} b_v^- \Big\}. \tag{11}$$

4.1.3 Implementational Details

In the preceding subsection, we assume that the node r that returns the minimum weight tree is previously known. In practice, this will rarely be the case. Instead, the integer program must be computed for every node $r \in V$ to find the optimal solution.

Another implementational aspect refers to connecting all terminals v to the core set S. It is not necessary to actually join all nodes of S into a single supernode before performing a breadth first search (BFS) in G to connect all v and find the final tree reservation. It is enough to initialize all nodes $r \in S$ as having distance 1 in a BFS (see listing 4). If there are no balanced edges, S contains only the node r that returned the minimum weight tree in the integer program (10).

```
// Input: graph G, queue Qu of core nodes in S, distance dist
//        dist[v] = 1, if node v belongs to S
//        dist[v] = INFINITY, else
// Output: reservation tree T

build T out of core nodes;
while (false == Qu.empty())
    v = Qu.pop();
    forall_inout_edges (e,v)
        w = G.opposite(v,e);
        if (dist[w] == INFINITY)
            dist[w] = dist[v] + 1;
            Qu.append(w);
            add e,w to T;

prune leaves of T that are not terminals;

return T;
```

Listing 4: findPathsBFS()—BFS algorithm, used to find the paths from the core to all terminals in a tree reservation.

4.1.4 Symmetric Tree Routing

For symmetric bandwidth bounds $b_v^- = b_v^+$ for each terminal $v \in Q$, [KRSY01] gives a polynomial time algorithm to compute the minimum cost VPN tree if links do not have capacity constraints. To do this, the quantity $Q(T, r)$ for a tree T and a root node $r \in T$ is defined as

$$Q(T, r) = 2 \sum_{v \in Q} (b_v^- + b_v^+) \cdot d_T(r, v) \tag{12}$$

where $d_T(r, v)$ is the unique path from r to terminal v in T.

In [KRSY01], the authors prove that there exists an optimal root node r such that $Q(T, r)$ is equal to the minimum weight of a tree reservation in a general graph G. To do this, they use a directed tree constructed from G through a BFS rooted at r. Listing 5 shows an algorithm they devised and which searches the optimal r by testing every node of G. The algorithm returns an optimal weight for symmetric tree routing and runs in $\mathcal{O}(mn)$ with m the number of edges and n the number of nodes in G.

```
// Input: graph G, terminals Q, bandwidth b
// Output: optimal tree reservation T_opt for node r_opt

forall_nodes(r, G)
    T_r = r;
    Qu = r;  // queue for BFS
    while (false == Qu.empty())
        v = Qu.pop();
        forall_inout_edges (e,v)
            w = G.opposite(v,e);
            if (w is not in T_r)
                add edge (v,w) to T_r;
                Qu.append(w);

    prune leaves of T that are not terminals;
    calculate Q(T_r,r);
    if (Q(T_r, r) < Q(T_opt,r_opt)
        T_opt = T_r;

return T_opt,r_opt;
```

Listing 5: BUILDSYMTREE()—Algorithm for computing an optimal symmetric tree routing.

4.2 Splittable Routing

Finding an optimal solution to splittable routing in the hose model means minimizing the optimization problem for a network $G = (V, E)$ given as

$$\text{minimize} \sum_{e \in E} c_e \cdot y_e \tag{13}$$

where y_e is the reserved capacity of the VPN on edge e with edge cost c_e. Because we will see in section 4.2.2 that this leads us to a separation problem solvable with cutting planes, let us first discuss the idea of cutting planes.

4.2.1 Cutting Planes

We consider a general separation problem consisting of the n inequalities

$$a_i^T \cdot x_i \leq b_i \quad (i = 1, \ldots, n). \tag{14}$$

These inequalities may be represented by a convex polyhedron P with n faces as shown in figure 16. Let $y_1, \ldots, y_n$ be nonnegative real numbers and set

$$c = \sum_{i=1}^{n} y_i \cdot a_i \qquad \text{and} \qquad d = \sum_{i=1}^{n} y_i \cdot b_i.$$

Every solution to (14) satisfies $c^T x \leq d$. Moreover, if c is integral, then we know that all integral solutions to (14) also satisfy the stronger inequality

$$c^T x \leq \lfloor d \rfloor \tag{15}$$

where $\lfloor d \rfloor$ denotes d rounded down to the nearest integer. We call (15) a cutting plane because the rounding of d cuts off part of the polyhedron P, although it does not cut off any integral vectors. Once we have derived a cut, we may add it to our system (14) and make use of it in deriving further inequalities. A sequence of such derivations is called a *cutting plane proof.*

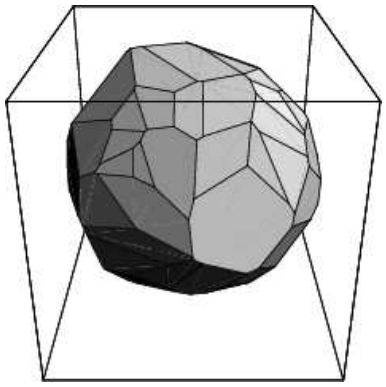


Figure 16: A convex polyhedron P.

Cutting Plane Algorithm When applying cutting plane proofs, the idea is that we have an integer programming formulation of a problem, i. e. $\max\{w^T x : x \in P; w, x \text{ integral}\}$ for some polyhedron P, together with some constraint of the form of the inequalities in (14). Using a linear programming algorithm, we find an optimal solution x^* to the linear programming relaxation $\{w^T x : x \in P; x \text{ real}\}$. If x^* is integral, it is also an optimal solution to the problem. Otherwise, we search the classes of valid inequalities to find some that are violated by x^*, that is, $c^T x^* > \lfloor d \rfloor$ (although $c^T \cdot x \leq \lfloor d \rfloor$ for all *integral* solutions x). We then add the violated inequalities to our linear programming relaxation and find a new optimal solution x^{**}. If x^{**} is integral we are done. Otherwise we search for inequalities that are violated by x^{**}, add them to our linear programming relaxation, and so on (see figure 17).

If we are lucky, we find an integral solution to one of the linear programming relaxations, but the subroutine that searches for violations may come back empty-handed. The process

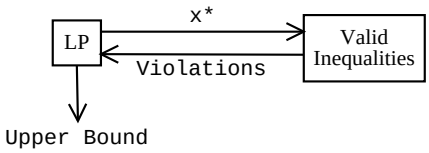


Figure 17: Cutting plane algorithm.

may terminate without reaching an integral optimal solution. In any case, the optimal value of each linear programming relaxation provides an upper bound on the optimal value of the problem that is usually better than any previous such bound and certainly no worse.

What makes a cutting plane algorithm even more powerful is the fact that it may not only be applied to *integer* linear programming. Just as well, we may apply it to any separation problem that consists of too many inequalities. We reduce this set of inequalities by "cutting off planes".

Example Consider the relaxed optimization problem

$$\text{maximize} \sum_{i=1}^{n} x_i \qquad 0 \leq x_i \leq 1 \tag{16}$$

with the constraints

$$\sum_{i \in S} x_i \leq 10 \quad \forall S \subseteq \{1, \ldots, n\}. \tag{17}$$

Let us assume that n is large, say $n = 100$. In this case, there are 2^{100} possible sets S resulting in 2^{100} constraints that must all be fulfilled. However, with a cutting plane algorithm, we will first find a solution to (16) without any of the constraints in (17), such as

$$x_1 = x_2 = \cdots = x_{100} = 1.$$

Now, we search for a violation of our constraints and we easily find for example the constraint

$$x_1 + x_2 + \cdots + x_{11} \leq 10.$$

Adding this constraint to our problem and calculating a solution to the newly created separation problem may give us for example

$$x_1 = x_2 = \cdots = x_{10} = 1, \quad x_{11} = 0, \quad x_{12} = x_{13} = \cdots = x_{100} = 1.$$

Again, we search for a constraint in (17) that is violated and which may now be $x_1 + x_2 + \cdots + x_{12} \leq 10$. Thus, we also set $x_{12} = 0$. Obviously, our algorithms runs on, until the values of all variables $x_{11}, \ldots, x_{100}$ are set to 0. We get a final optimal solution

$$x_1 = x_2 = \cdots = x_{10} = 1, \quad x_{11} = x_{12} = \cdots = x_{100} = 0.$$

Note that our cutting plane algorithm took only 91 iterations of solving the linear program and finding a violated constraint.

■

For more information, see [CCPS98] chapter 6.8 for a detailed general discussion of separation and optimization and chapter 7.4 and 6.7 for an overview of cutting planes.

4.2.2 Linear Programming Formulation

Let us return to our optimization problem

$$\text{minimize} \sum_{e \in E} c_e \cdot y_e \tag{13}$$

which will give us the optimal edge reservation y_e for all e in $G = (V, E)$ for splittable routing if we formulate it as an LP. To do this, we define traffic split values $x_{uv,e}$ as the fraction of traffic from $u \in Q$ to $v \in Q$ routed through edge e. Let $M = (m_{u,v})$ be a valid traffic matrix. We get an LP defined as:

$$\text{minimize} \sum_{e \in E} c_e \cdot y_e$$

subject to the reservation constraint

$$\forall e \in E, M = (m_{u,v}) : \sum_{u,v \in Q} x_{uv,e} \cdot m_{u,v} \leq y_e,$$

the flow constraints

$$\forall u \in Q, v \in Q : \sum_{u=\text{source}(e)} x_{uv,e} = 1 \quad \text{and} \quad \sum_{v=\text{target}(e)} x_{uv,e} = 1,$$

$$\forall u \in Q, v \in Q, w \in V \setminus \{u, v\} : \sum_{w=source(e)} x_{uv,e} = \sum_{w=target(e)} x_{uv,e},$$

and

$$\forall u \in Q, v \in Q, e \in E, u = \text{target}(e) \vee v = \text{source}(e) : x_{uv,e} = 0$$

for

$$\begin{aligned} 0 \leq x_{uv,e} \leq 1 \\ 0 \leq y_e \leq \infty. \end{aligned} \tag{18}$$

The reservation constraint of LP (18) follows from the fact that all possible traffic matrices M must be supported by our splittable routing. An edge reservation y_e must always be larger or equal to whatever traffic could be routed through e in the worst case. The first two flow constraints are illustrated in figure 18 and assure that traffic fractions sum up to 100% at u and v and that we encounter no nodes $w \in V \setminus \{u, v\}$ where flow is lost or added. Of course, we need to make sure that no flow from u is allowed into u and that no flow leaves v.

Because $0 \leq x_{uv,e} \leq 1$ and $0 \leq y_e \leq \infty$, (18) is a relaxed problem and hence polynomially solvable by the ellipsoid algorithm [GLS81]. However, if we included all possible reservation constraints for all possible traffic matrices M, we would end up with a huge problem. Fortunately, it is not necessary to include all these constraints since (18) may be identified as a separation problem solvable with cutting planes discussed in the previous section 4.2.1. Analogously to figure 17, figure 19 illustrates the cutting plane algorithm applied to our reservation constraints. The algorithm starts out with no reservation constraint and solves the LP.

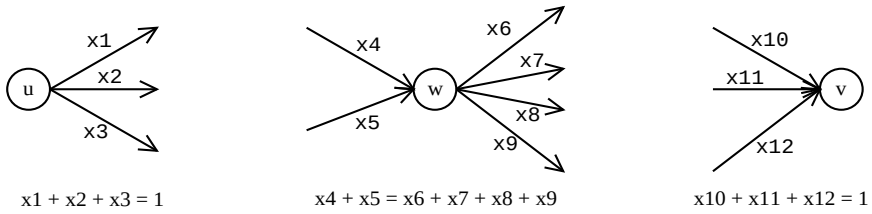


Figure 18: Illustration of the flow constraints for LP (18).

A hose checker is used to determine which edge capacities y_e encounter an overload because of some M identified by the checker. Thus, we add reservation constraints for M on one of those violated edges e to our LP. Then, the LP is solved anew. The algorithm only stops once the checker returns no more violated y_e and an optimal splittable routing is found.

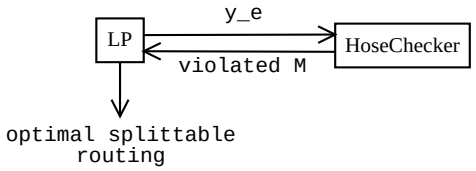


Figure 19: Cutting plane algorithm for LP (18).

For completeness, lets us mention that the above LP (18) may be extended by a constraint as given in (19), in order to allow for finite capacity limits C_e:

$$y_e \leq C_e. \tag{19}$$

4.2.3 Path Stripping

This far, we have neglected one important step when finding a splittable routing in the hose model. LP (18) may have given us edge reservation values y_e for a graph G and traffic split values or fractional flows $x_{uv,e}$. However, these split values give us only information on the edges e of G and not on paths P_{uv} between terminals u and v. What we want though, is true routing information and thus paths P_{uv}.

Figure 20 shows as an example a part G_{uv} of a graph G where $x_{uv,e} \neq 0$ for a pair of terminals u and v. In order to identify paths P_{uv}, a technique called *path stripping* is used.

In figure 20, traffic from terminal q_1 to q_2 is routed along four paths with different split percentages. The paths are identified as

50% $q_1 \longrightarrow A \longrightarrow q_2$

20% $q_1 \longrightarrow B \longrightarrow A \longrightarrow q_2$

10% $q_1 \longrightarrow B \longrightarrow D \longrightarrow q_2$

20% $q_1 \longrightarrow C \longrightarrow E \longrightarrow D \longrightarrow q_2$

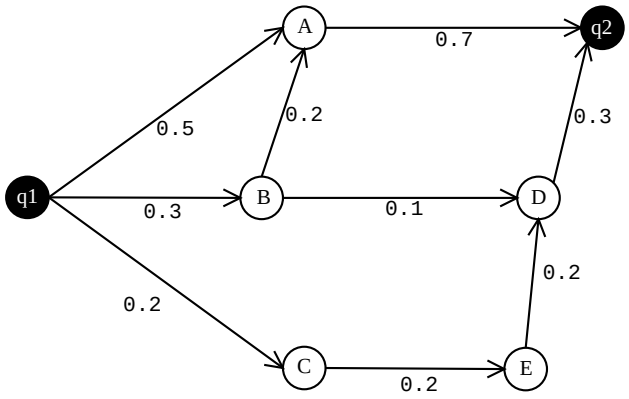


Figure 20: A graph G_{uv} with edge flow values assigned by a linear program solution from terminal q_1 to q_2. Four paths may be stripped from q_1 to q_2 in this graph. (Note that G_{uv} is shown as a directed graph to emphasize possible paths.)

How do we find these paths? Rather than looking at edge split values in isolation, we need to look at them as the splitting of traffic as it flows through G. For example at node B, incoming traffic of 0.3 from q_1 is split into traffic of 0.2 towards A and 0.1 towards D. We notice that traffic from q_1 to A is 0.2 and 0.1 from q_1 to D.

A path stripping procedure such as the one given in section 5.2 of [GD96] explicitly constructs this interpretation by assigning a flow value to each possible path of nonzero flow for every routing between two terminals u and $v \in Q$. A path P_{uv} is assigned a value $p_{x,uv}$ corresponding to the largest possible common traffic percentage across all edges of the path, i. e. the minimum traffic split percentage on any one edge in the path. A formal algorithm for path stripping is shown in listing 6.

```
// Input: graph G, terminals u and v, traffic split values x_uv_e
// Output: list of paths P_uv, fractional flows p_x for all P_uv

build a graph G_uv out of all edges e where x_uv_e != 0;
P_uv.clear();

while (aPath = findPathDFS(u,v,G_uv))
    p_x[aPath] = min(x_uv_e[e]) for all e in aPath;
    P_uv.append(aPath);
     forall_edges (e,aPath)
         x_uv_e[e] = x_uv_e[e] - p_x[aPath];
         if (x_uv_e[e] == 0)
             G_uv.del_edge(e);

return P_uv, p_x;
```

Listing 6: stripPaths()—Path stripping algorithm that makes use of a DFS in findPathDFS().

The **while** loop in listing 6 terminates as the function findPathDFS() may find at most as many paths in G as there are edges in G_{uv} because at least one edge is removed per

iteration. A good way to implement findPathDFS() is—as the name suggests—through a recursive depth first search (DFS) (see listing 7).

```
// Input: graph G, start node s, end node t
// Output: a depth first path aPath

reached[s] = true;
forall_out_edges (e,s)
  v = G.target(e);
  if (reached[v] == false)
      if (v == t)
          found = true;

      if (found == false)
          findPath(G,v,t);    // recursion
      if (found == true)
          aPath.append(e);
          return;
  else   // v has been reached before
      return;
```

Listing 7: findPathDFS()—Recursive DFS algorithm, used to find the paths in a path stripping algorithm. Please note that this implementation returns a path in reverse order, from t to s!

4.3 Unsplittable Routing

Obviously, the technique to build a splittable routing hose presented in the previous section 4.2 is easily adapted to a new LP for unsplittable routing by restricting the acceptable range of the traffic split values $x_{uv,e}$. Either, there is traffic between terminals u and v over e ($x_{uv,e} = 1$) or not ($x_{uv,e} = 0$). No splitting is allowed. Such a redefinition of LP (18) is given as

$$\text{minimize} \sum_{e \in E} c_e \cdot y_e$$

subject to the reservation constraint

$$\forall e \in E, M : \sum_{u,v \in Q} x_{uv,e} \cdot m_{u,v} \leq y_e,$$

the flow constraints

$$\forall u \in Q, v \in Q, : \sum_{u=\text{source}(e)} x_{uv,e} = 1 \quad \text{and} \quad \sum_{v=\text{target}(e)} x_{uv,e} = 1,$$

$$\forall u \in Q, v \in Q, w \in V \setminus \{u, v\} : \sum_{w=source(e)} x_{uv,e} = \sum_{w=target(e)} x_{uv,e},$$

and

$$\forall u \in Q, v \in Q, e \in E, u = \text{target}(e) \vee v = \text{source}(e) : x_{uv,e} = 0$$

for

$$\begin{aligned} x_{uv,e} &\in \{0,1\} \\ 0 \leq y_e &\leq \infty. \end{aligned} \tag{20}$$

Unfortunately, through the restraint $x_{uv,e} \in \{0,1\}$, LP (20) is not a relaxed problem any more. Instead, we now have a mixed integer program known to be $\mathcal{NP}$-hard.

To conclude this section, let us restate that we have given polynomial time solutions for symmetric tree and general splittable building of a hose. Interestingly, it seems to be easier to solve a splittable routing than an unsplittable one. This far, it is not known whether unsplittable routing is $\mathcal{NP}$-hard or not.

5 Practical and Theoretical Aspects of Routing Results

After being able to build and check hoses in an arbitrary network G, let us discuss some interesting results and find some surprising practical as well as important theoretical consequences when comparing different routing approaches.

5.1 Hose vs. Pipe Model

Naturally, we would like to know how good the hose model is. In section 2.1 we have shown the differences between a hose and a pipe. There, figure 2 illustrated how the overall cost W_T of a tree routed hose may be considerably less than the cost W_{Pipe} in a Dijkstra shortest path pipe model for a network G.

However, comparing the two models does not tell anything about how good each of them is in absolute terms. That may be achieved by comparing them to an absolute value, i. e. the cost in G when routing a single traffic matrix M. M is chosen from all possible traffic matrices that make maximal use of all b^- and b^+. This means that for $M = (m_{u,v})$

$$\sum_{u \in Q} m_{u,v} \leq b_v^- \quad \text{and} \quad \sum_{v \in Q} m_{u,v} \leq b_u^+. \tag{21}$$

This problem of finding M constrained by (21) may be solved as a b-matching or maximum flow problem between all u and v as we similarly encountered in section 3.2.2. It is depicted in figure 21. Since we assume no traffic from a node v to itself, there is no connection between v on the left to v on the right side of a bipartite graph.

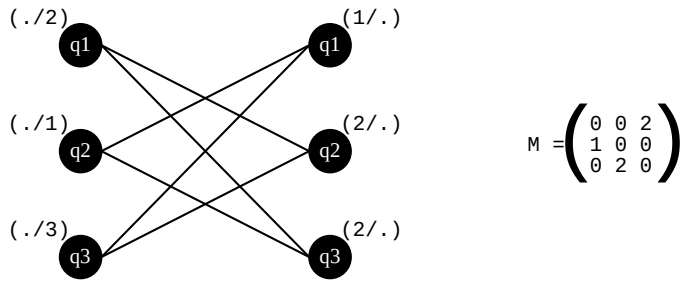


Figure 21: Generating a traffic matrix M from given b-values of a VPN with three terminals.

In the following, we study simulation results for the two models compared to the minimally required cost by M, first with symmetric bound $b_v^- = b_v^+$ for all $v \in Q$, then with asymmetric bounds $b_v^- \neq b_v^+$. The data of every simulation is based on $20,000$ generated graphs G with

$$\begin{aligned} n = |V| &\in \{3, 4, \ldots, 10\} \\ m = |E| &\in \left\{ n-1, n, \ldots, \frac{n \cdot (n-1)}{2} \right\} \\ q = |Q| &\in \{3, 4, \ldots, n\} \\ b_v^+, b_v^- &\in \{1, 2, \ldots, 5\}. \end{aligned} \tag{22}$$

For every possible configuration, there were 100 random variants chosen. Thus, there were for example 100 graphs with $n = 3$, $m = 2$, and $q = 3$ and 100 graphs with $n = 10$, $m = 45$, and $q = 10$.

5.1.1 Symmetric Routing

The following figures 22 and 23 show the results of the simulations described above for symmetric routing. Figure 22 depicts the situation for the hose model while figure 23 does so for the pipe model. Data is sorted first by increasing weight of single traffic matrices M as given by (21) for a certain G. The total reservation needed by such an M may be viewed as equal to the minimally required weight of the network reservation for given b-values. The second sort argument is the weight in the hose model and the third the weight in the pipe model. It is thus that in figure 22, the data shows a regular sawtooth curve while in figure 23, the data appears to be more random. The advantage is that for any instance of a graph on the x-axis, the data in the two figures corresponds and may be compared directly. The hose model for

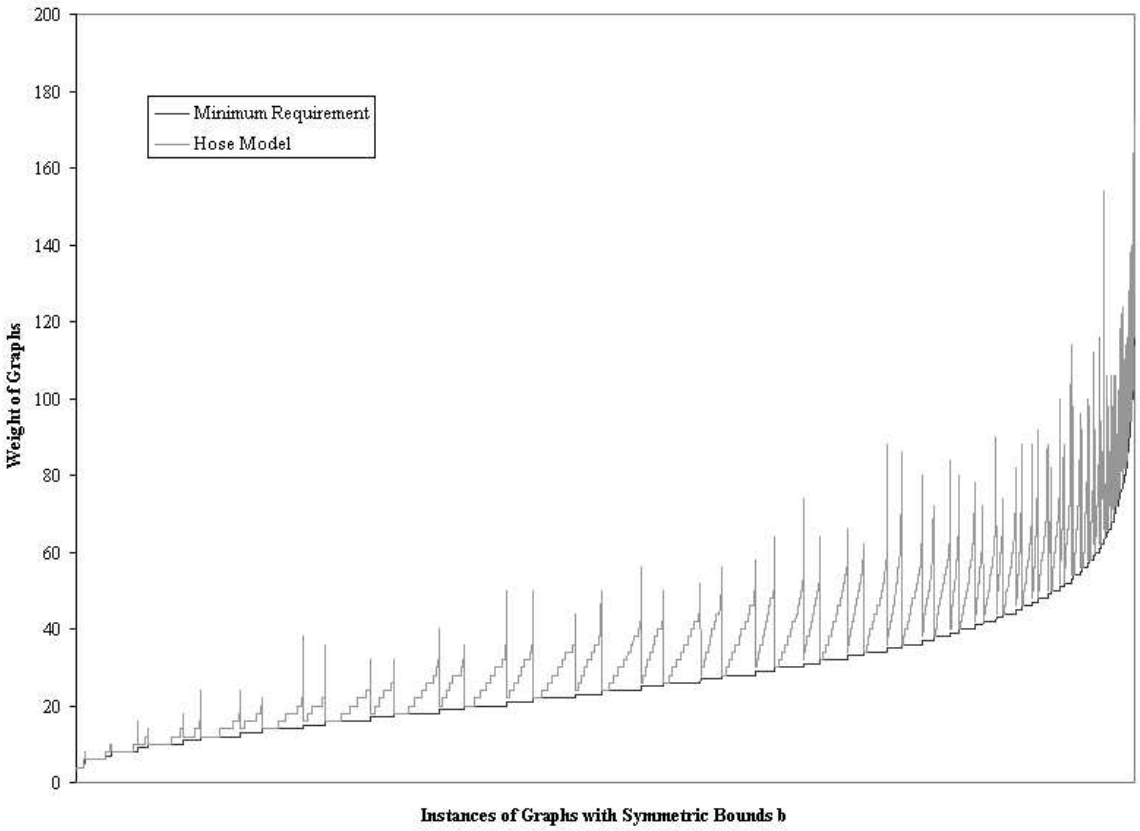


Figure 22: Comparing the overall weight of a *symmetric* bounds reservation in the *hose model* for trees to a base for $20,000$ instances of graphs G. This base is a minimally required reservation by a single traffic matrix M in G.

symmetric tree routing in figure 22 shows the following results:

- The maximum hose weight reservation equals 172. There, an optimal traffic matrix would only need a reservation of 138.
- The average hose weight reservation is 34.0 while traffic matrices need 27.0.

- The minimum hose weight reservation is 4 in a network where a traffic matrix needs at most a reservation of 3.
- Overall, the hose model returns an optimal reservation in 3837 cases. Of course, often optimal reservations are found if G is a tree. However, this result includes a lot more than just the cases where G is a tree!

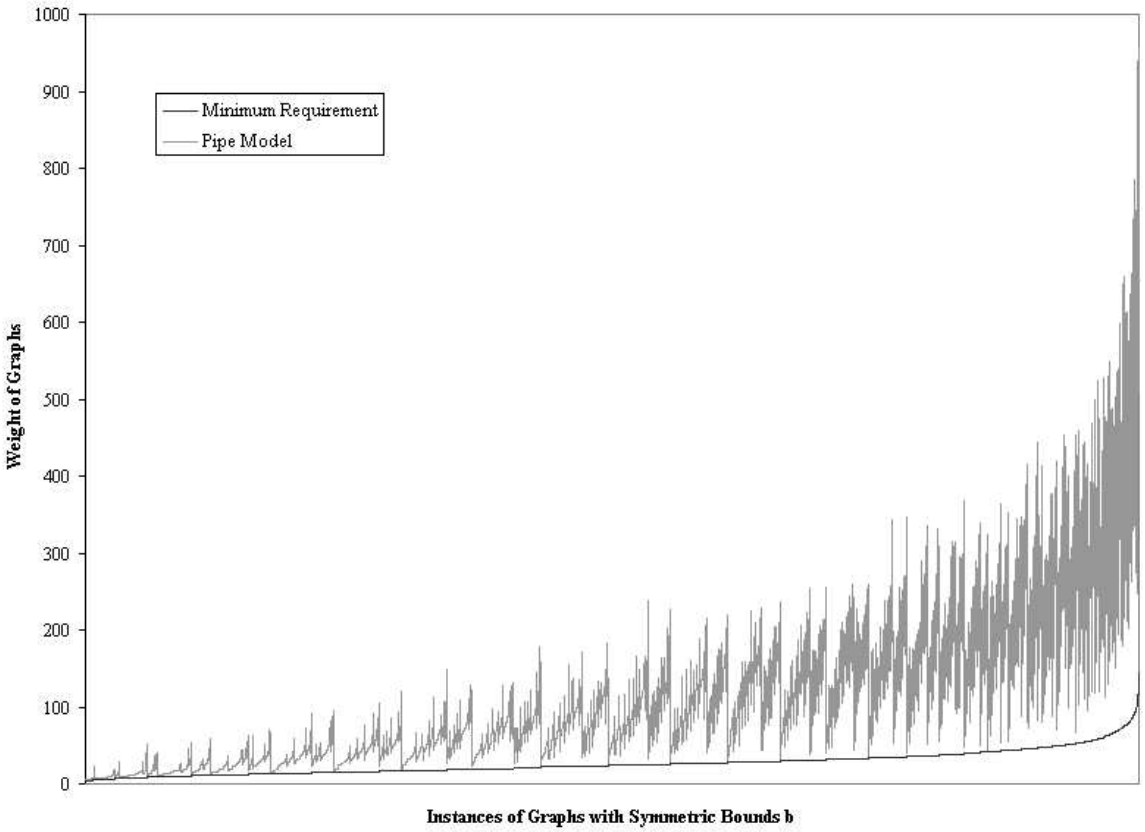


Figure 23: Comparing the overall weight of a *symmetric* bounds reservation in the *pipe model* to a best-case base for $20,000$ instances of graphs G. The base is a minimally required reservation by a single traffic matrix M in G.

For the symmetric pipes built in our $20,000$ networks G in figure 23, we find:

- The maximum pipe weight reservation is equal to 958 where an optimal traffic matrix would only need a reservation of 146!
- The average pipe weight reservation is 110.0.
- The minimum pipe weight reservation is 6 where an optimal value could be 3.
- Overall, the pipe model never returns an optimal reservation for any one of our networks.

As a direct result, we see that the pipe model returns at most a reservation that is equal to the one in the hose model. Put in other words, the pipe model never performs better than an optimal hose. This is further emphasized by the following data on the weight of symmetric reservations: The pipe model shows a minimum deviation of a factor of 1.04 from a minimum requirement. The hose model may be equal to the minimum. The maximum deviation factor for the pipe model is 10.05, while it is only 2.71 for the worst hose reservation! The pipe model has an average deviation of a factor of 3.56. For hoses, this factor is only 1.26. Thus, the hose model performs almost three times better than the pipe model for symmetric routing.

5.1.2 Asymmetric Routing

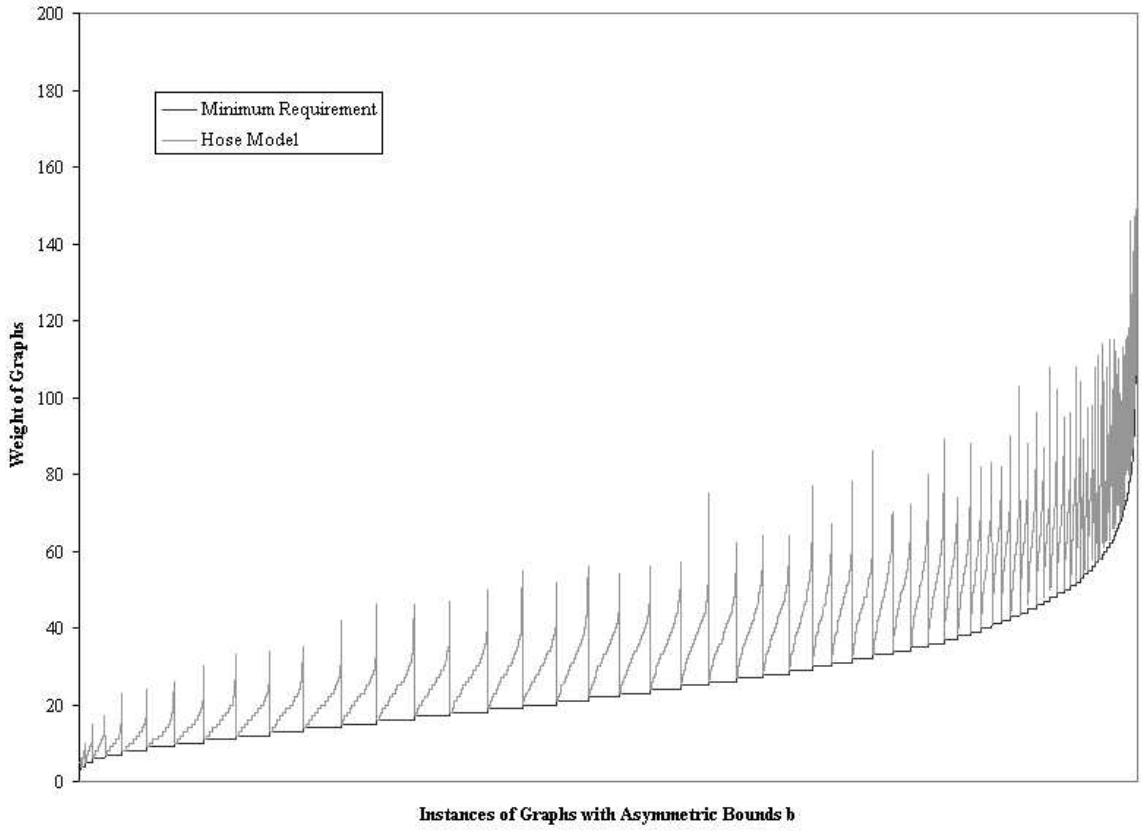


Figure 24: Comparing the overall weight of an *asymmetric* bounds reservation in the *hose model* for trees to a best-case base for $20,000$ instances of graphs G. The base is a minimally required reservation by a single traffic matrix M in G.

Results very similar to the ones of the previous section on symmetric routing are found when we look at figures 24 and 25 that compare reservation weights between the minimum requirement and the hose model for and the minimum requirement and the pipe model respectively for asymmetric routing. In the hose model for asymmetric tree routing we read out of figure 24 that:

- The maximum hose weight reservation of all 20,000 networks is equal to 149. There, an optimal traffic matrix would only need a reservation of 109.
- The average hose weight reservation is 36.0 while traffic matrices need 25.1.
- The minimum hose weight reservation is 4 which is also needed by an optimal traffic matrix.
- Overall, the hose model returns an optimal reservation in 931 cases. Obviously, some of them are found if G is a tree.

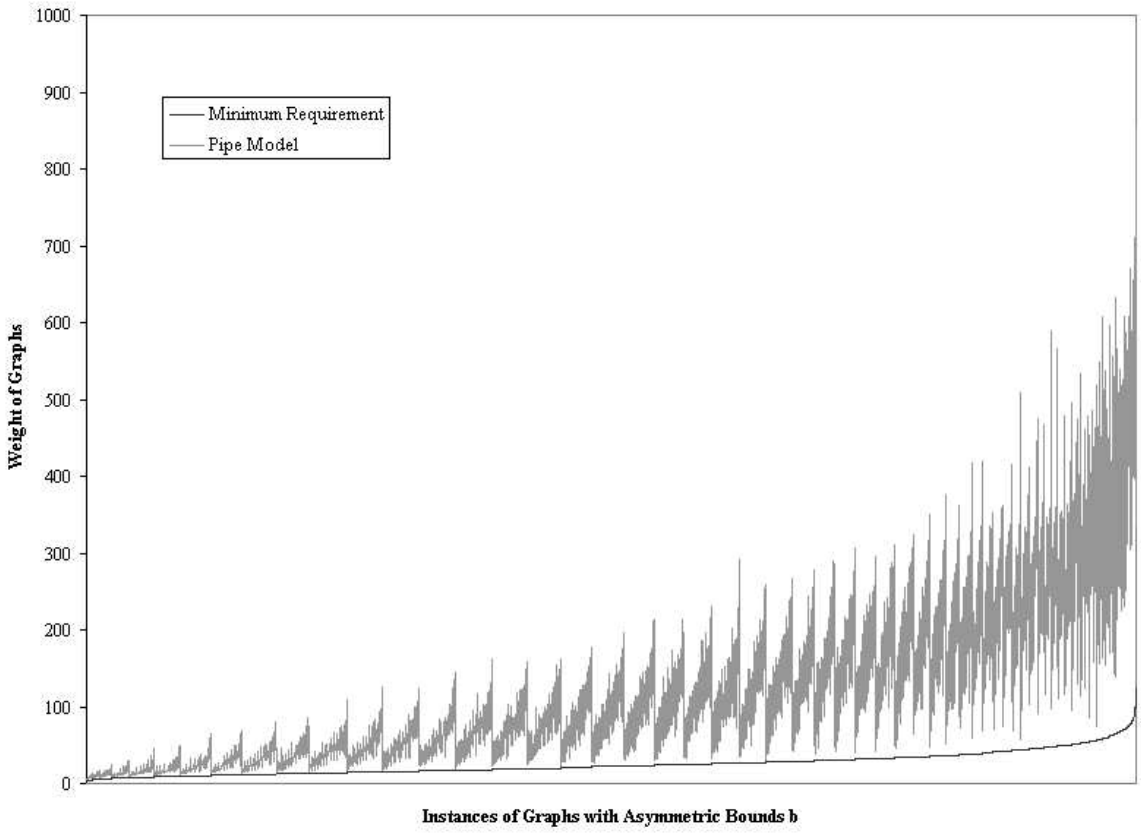


Figure 25: Comparing the overall weight of an *asymmetric* bounds reservation in the *pipe model* to a best-case base for 20,000 instances of graphs G. The base is a minimally required reservation by a single traffic matrix M in G.

For the asymmetric pipes built in our 20,000 networks G in figure 25, we find:

- The maximum pipe weight reservation is equal to 859! There, an optimal traffic matrix would only need a reservation of 126.
- The average pipe weight reservation is 116.6.
- The minimum pipe weight reservation is 6 where an optimal value could be 3.

- As already seen in symmetric routing, the pipe model never returns an optimal reservation for any one of our networks.

Collecting all asymmetric data, we find that the minimum deviation from the minimum requirement by the pipe model is by a factor of 1.07. Once more, the hose model is optimal for some graphs. The maximum deviation by the pipe model is by the huge factor of 12.53 while only 3.28 in the worst found hose. On average, the deviation factor for asymmetric routing in the pipe model is 4.02 and in the hose model 1.45. Again, the hose model performs almost three times better.

We notice the small differences between asymmetric and symmetric routing. Symmetric routing performs slightly better in general, and almost a fifth of all symmetric hoses are optimal. This may of course partially be caused by the small networks we consider. Also, when looking at the graphs, we see that the data for the hose model is less diverse for symmetric than for asymmetric routing. This is natural because there are less configurations if b^- is equal to b^+.

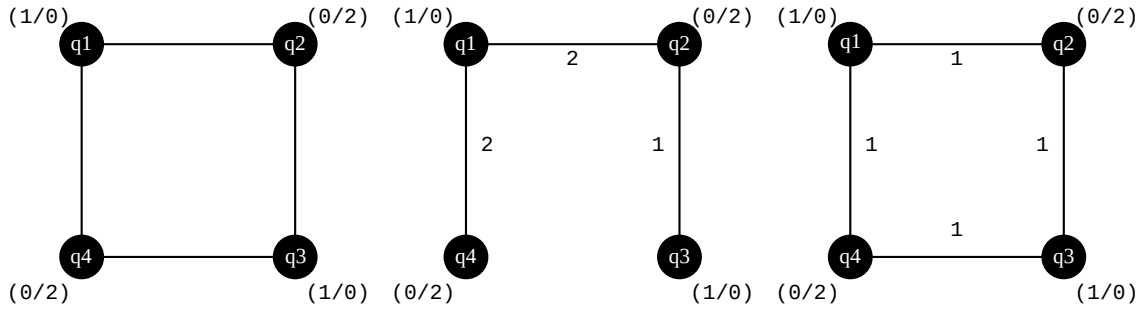
5.2 Comparing Tree, Unsplittable, and Splittable Reservations

In section 2.1, we have defined $W = \sum_{e \in E} c_e \cdot y_e$ as the overall weight or cost of any hose reservation in a graph G. Let now W_T be the cost of a tree reservation in G, W_U the cost for an unsplittable reservation, and W_S the cost for a splittable reservation. Obviously, the following comparison is always valid:

$$W_T \geq W_U \geq W_S \tag{23}$$

because an unsplittable routing reservation offers at least the same possibilities as routing in a tree, and splittable routing is at least an unsplittable routing.

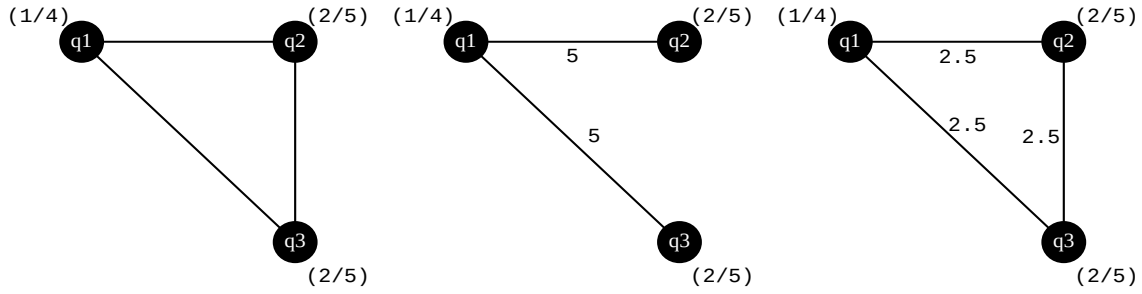
5.2.1 Asymmetric Routing



(a) Graph G (b) Tree Routing (c) Unsplittable Routing

Figure 26: For a network G in (a), tree routing in (b) has a cost of 5 while unsplittable routing in (c) has one of only 4. This example was first mentioned in [GKK+01].

In asymmetric routing, we may find manageable examples where $W_T > W_U = W_S$ or $W_T = W_U > W_S$ without too much trouble. Figure 26 illustrates the first case, figure 27



(a) Graph G (b) Tree / Unsplittable Routing (c) Splittable Routing

Figure 27: For a network G in (a), unsplittable routing in (b) has a cost of 10 while splittable routing in (c) has one of only 7.5.

the second one. Even though the graph in figure 27 has only three nodes, finding the given splittable reservation routing is not trivial and we thus list it here:

(u, v)	Splittable			
$q_1 \longrightarrow q_2$	50%:	q_1, q_2	50%:	q_1, q_3, q_2
$q_1 \longrightarrow q_3$	50%:	q_1, q_3	50%:	q_1, q_2, q_3
$q_2 \longrightarrow q_1$	50%:	q_2, q_1	50%:	q_2, q_3, q_1
$q_2 \longrightarrow q_3$	50%:	q_2, q_3	50%:	q_2, q_1, q_3
$q_3 \longrightarrow q_1$	50%:	q_3, q_1	50%:	q_3, q_2, q_1
$q_3 \longrightarrow q_2$	50%:	q_3, q_2	50%:	q_3, q_1, q_2

An example where $W_T > W_U > W_S$ is given in figure 28. Again, we list the routing paths, this time for both, unsplittable and splittable routing:

(u, v)	Unsplittable	Splittable			
$q_1 \longrightarrow q_2$	q_1, q_2	50%:	q_1, q_2	50%:	q_1, q_4, q_3, q_2
$q_1 \longrightarrow q_3$	q_1, q_4, q_3	50%:	q_1, q_2, q_3	50%:	q_1, q_4, q_3
$q_1 \longrightarrow q_4$	$q_1, 1_4$	50%:	q_1, q_2, q_3, q_4	50%:	q_1, q_4
$q_2 \longrightarrow q_1$	q_2, q_3, q_4, q_1	50%:	q_2, q_3, q_4, q_1	50%:	q_2, q_1
$q_2 \longrightarrow q_3$	q_2, q_3	50%:	q_2, q_3	50%:	q_2, q_1, q_4, q_3
$q_2 \longrightarrow q_4$	q_2, q_3, q_4	50%:	q_2, q_3, q_4	50%:	q_2, q_1, q_4
$q_3 \longrightarrow q_1$	q_3, q_4, q_1	50%:	q_3, q_4, q_1	50%:	q_3, q_2, q_1
$q_3 \longrightarrow q_2$	q_3, q_2	50%:	q_3, q_2	50%:	q_3, q_4, q_1, q_2
$q_3 \longrightarrow q_4$	q_3, q_4	50%:	q_3, q_4	50%:	q_3, q_2, q_1, q_4
$q_4 \longrightarrow q_1$	q_4, q_1	50%:	q_4, q_1	50%:	q_4, q_3, q_2, q_1
$q_4 \longrightarrow q_2$	q_4, q_3, q_2	50%:	q_4, q_1, q_2	50%:	q_4, q_3, q_2
$q_4 \longrightarrow q_3$	q_4, q_3	50%:	q_4, q_1, q_2, q_3	50%:	q_4, q_3

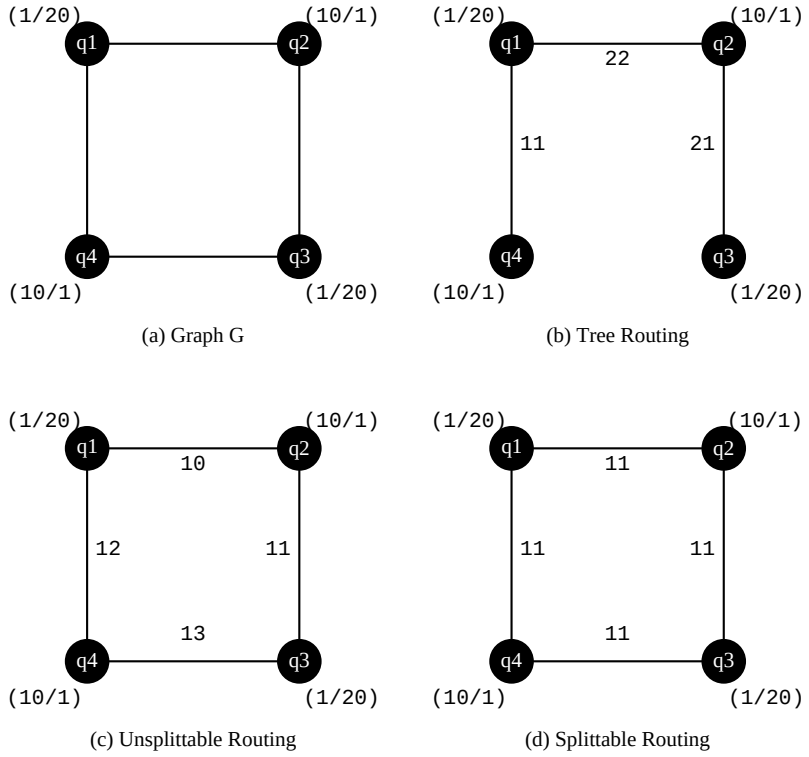


Figure 28: For a network G in (a), tree routing in (b) has a cost of 54, unsplittable routing in (c) one of 46, and splittable routing in (d) one of 44.

Let us now look at simulation results to find some interesting facts about asymmetric tree, unsplittable and splittable routing. Once more, we work with randomly generated networks. They include 6200 graphs with 3 to 5 nodes, formally described in (24). Unlike in section 5.1, bounds b are chosen less strictly here with $b_v^-, b_v^+ \in \{1, 2, \ldots, 50\}$. This allows for a wider range of solutions, especially when comparing unsplittable to splittable routing reservations where b-values have a strong influence on routing paths.

$$\begin{aligned} n = |V| &\in \{3, 4, 5\} \\ m = |E| &\in \left\{ n-1, n, \ldots, \frac{n \cdot (n-1)}{2} \right\} \\ q = |Q| &\in \{3, 4, \ldots, n\} \\ b_v^+, b_v^- &\in \{1, 2, \ldots, 50\}. \end{aligned} \tag{24}$$

Please note, that there is always the case included where $m = n - 1$ and thus where the network G itself is a tree. These cases are not interesting, but are nevertheless included in order to represent all possible networks and to prevent corruption of absolute values.

In figure 29 we show that for every fifth network G that we look at, splittable routing reserves less bandwidth than tree routing. The result is approximately the same for networks with 3 to 5 nodes. Additional experiments show the same ratio when G has 6 or 7 nodes. Interestingly, there was only two cases in all examples where unsplittable routing found a better routing than tree routing. Here, however, network size might play a role.

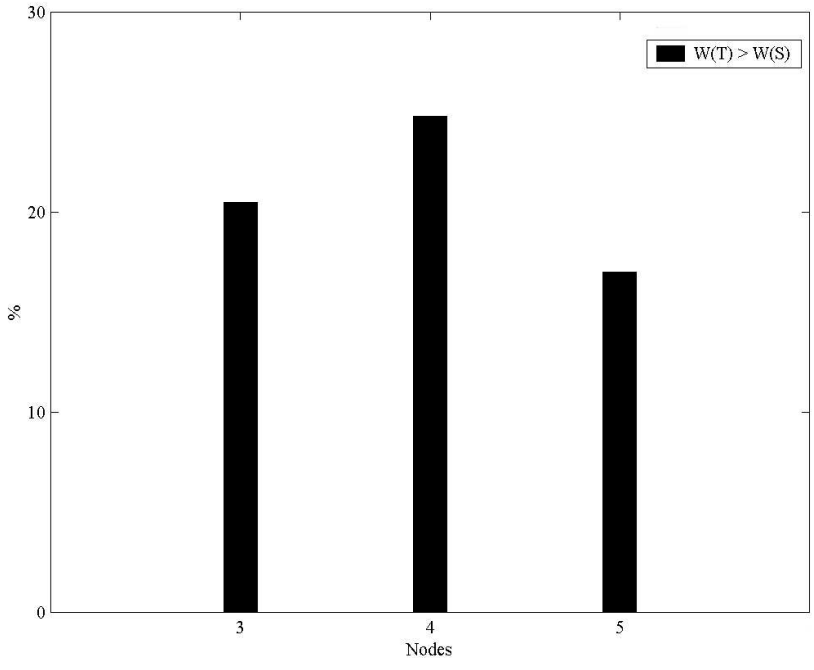


Figure 29: Percentage of networks G that have a smaller reservation for splittable routing than for tree routing.

Of course, we also want to know, how much better a splittable reservation is compared to a tree reservation. Figure 30 displays two kinds of bars for this. Black bars indicate that *if* a splittable routing is better than a tree routing, it is so by 8.6% on average and for any number of nodes. On the other hand, white bars show that, taken all studied networks G (and thus also those where $W_T = W_S$), an average gain of only 1.6% is achieved. These numbers stay approximately the same for small networks with 3, 4, or 5 nodes.

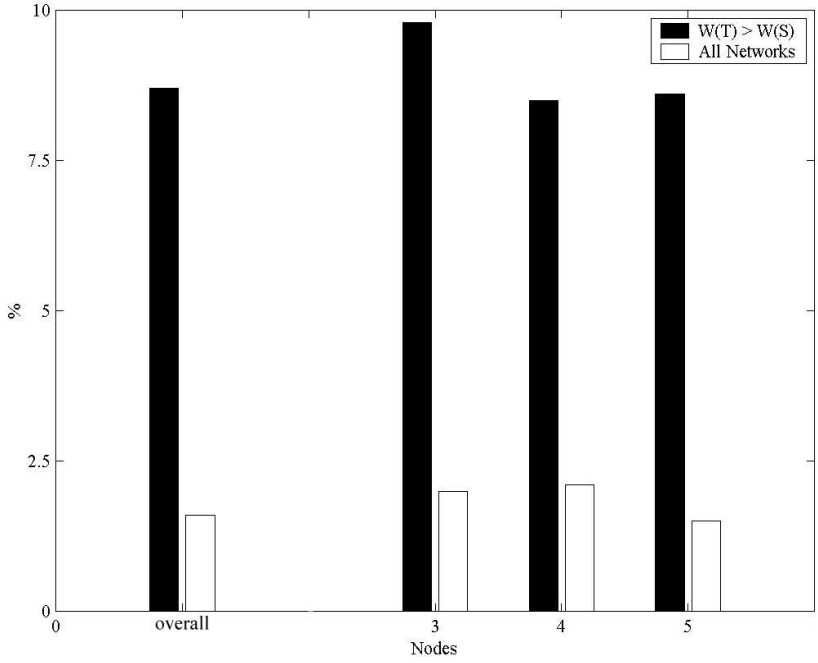


Figure 30: Gain when using splittable routing as opposed to tree routing.

We may conclude that, while splittable routing *does* perform better for small networks, the difference to tree routing and thus the gain we get is relatively small.

5.2.2 Symmetric Routing

As soon as $b_v^- = b_v^+$ for all $v \in Q$, the picture from the last section changes dramatically. In over $100,000$ graphs with 3 to 20 nodes and bounds b between 1 and up to 150, we did not find a single instance of a network where tree routing returned an inferior reservation than unsplittable or splittable routing. Thus, we state the following conjecture.

Conjecture For unsplittable and splittable routing in a general graph G where all terminals v have *symmetric* bounds $b_v^- = b_v^+$, the resulting hose reservation always forms a tree, and thus $W_T = W_U = W_S$.

The following theorems strengthen our conjecture.

Theorem 5.1 *If every terminal u routes along a tree T_u, the optimal cost W_U is equal to W_T.*

Proof In section 3.1.1 of [GKK+01], the authors give a proof to theorem 5.1 that uses auxiliary graphs $G_e(W, E_e)$ for each $e \in E$, where E_e contains the edge (u, v) if a direct path P_{uv} in G contains e. From these graphs and edge capacities y_e, fractional matchings are constructed.

□

Theorem 5.2 *The optimal cost W_U for a single terminal u to all other terminals v is always a tree T_u in unsplittable routing.*

Proof The optimal cost for a single terminal u to all other terminals v is a reformulation of the definition of a Steiner tree.

□

Figure 15 in section 4.1.1 gives additional hints towards the correctness of our conjecture. While it is clear that a Steiner tree may not always give an optimal symmetric reservation as found by a hose, the example in figure 15 is a tree nevertheless. Unfortunately, we could not prove our conjecture. So far we are missing the important link between our theorems.

5.3 Different Representations: Undirected Graphs and Bidirected Graphs

One final result that needs attention is the fact that there are two different possible network representations that seem to be equivalent, but are not when looking at hose routing:

- routing in an undirected graph and
- routing in a bidirected graph.

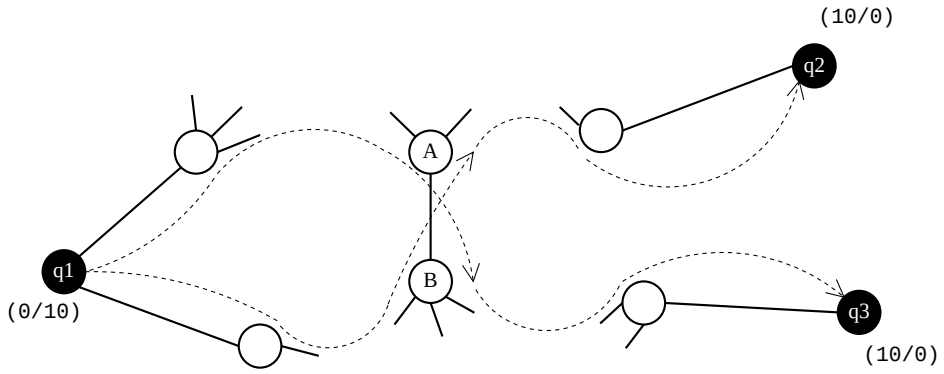


Figure 31: A network where a bidirected graph will need more bandwidth than an undirected one.

While most theory in this paper talks about undirected graphs, implementations are mostly done with bidirected graphs. The reason for this is that shortest paths and flow

algorithms may cope more easily with bidirected graphs where every edge has a direction than with undirected ones.

Figure 31 shows a part of the network G where routing results depend on whether the link between nodes A and B is undirected or bidirected. There are only two demands in the depicted part of G. One leads from q_1 to q_2 with a maximum bandwidth of 10, the other from q_1 to q_3, again with a bandwidth of 10. If for some reason, such as additional terminals with certain bounds b^- and b^+, the two possible routings are chosen as illustrated in the figure, the path $P_{q_1q_2}$ passes nodes A and B in the opposite direction of the second path $P_{q_1q_3}$. In such a situation, a bidirected network always needs a larger hose reservation than an undirected one.

As made clear in figure 32, an undirected link (A, B) in G needs a reservation of 10 while a bidirected link needs the same amount in each direction. Thus, in the worst case, a bidirected network representation may have a weight of twice that of an undirected representation.

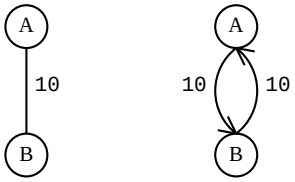


Figure 32: A close view of the two nodes A and B from figure 31. A bidirected graph needs a reservation of 20 between A and B, an undirected one of only 10.

6 Conclusion

The concept of the hose model has proven to show very interesting behaviors and phenomena. We are glad that we could give insight to a few of them and uncover others. While we constrained ourselves to tree, unsplittable, and splittable routing for symmetric and asymmetric bounds, we found the following results.

We were able to present ways to validate given hose reservations. With a checker algorithms, we could efficiently show whether a given hose routing and reservation is valid or not for all forms of routing discussed in this report. Additionally, our algorithms could determine, whether a reservation is tight and using all the given capacity or loose and wasting bandwidth.

Through linear programming formulations, it became possible to build hoses. An efficient polynomially solvable LP for splittable routing was developed. It made use of an optimization technique known as cutting planes where validating reservations suddenly played a very important role. For all three forms of routing, algorithms were formulated or, in the case of tree routing, previously known solutions adapted. This included the implementation of the special case of symmetric tree routing where a polynomial algorithm is known.

An interesting simulation we carried out compared the performance of hoses to a minimum required reservation weight and to the reservations in the pipe model. Results showed that a hose may well find an optimal solution for small networks and that it is never far from one. On the other hand, the pipe model returned reservations that are on average almost three times worse than a hose.

We also looked at differences of tree, unsplittable and splittable hose routing. For asymmetric routing, we gave three examples where weights of a tree, an unsplittable and a splittable routing differ. Simulations showed how the different models compare to one another. For small graphs and asymmetric bounds, splittable routing returns reservations that are better than those for tree or unsplittable routing in a fifth of all cases. This entails that on average, splittable routing reserves and uses less bandwidth; remarkably, the gain is only 2%. Cases where unsplittable routing is preferable to tree routing are extremely rare.

Last, but very important, we found out about the fact that hoses depend on the representation of a network. A bidirected graph could result in a reservation that is twice as much as an undirected graph.

Several problems are still open. In [ILO02], the authors asked themselves whether a tree reservation would be optimal for symmetric routing. We have run simulations including over $100,000$ graphs with not a single case where unsplittable or at least splittable routing would return a smaller reservation weight than a symmetric tree routing. Thus, we have stated a conjecture that symmetric tree routing is optimal. However, we were unable to prove it coherently.

Furthermore, we do not know whether unsplittable routing is $\mathcal{NP}$-hard. If we assume the above conjecture that for symmetric routing tree and unsplittable reservation weights are equal, we do have a polynomial symmetric tree routing algorithm. For asymmetric routing the problem of tree routing is proven to be $\mathcal{NP}$-hard, but unsplittable routing does not always return equal reservation weights as tree routing.

Additionally, no approximation algorithms for the problem of designing a minimum cost network are known for the asymmetric unsplittable case.

Finally, network size is worthy to be further analyzed. Do large networks behave in the same way as our small ones? And, together with this, a last open issue is the influence of the

range of bounds. How much does the range of bounds influence different results? It seems that there are more cases where asymmetric unsplittable routing performs better than tree routing if we allow for a wider range of bound values. A thorough future analysis will have to clear this.

A Appendix

A.1 Conceptual Thesis Formulation

The following potential tasks for the diploma thesis were given by Prof. Dr. Thomas Erlebach on October 21, 2002.

1. **Feasibility Checker.** Implement a program that checks whether a VPN reservation is feasible (supports all valid traffic matrices). The checker takes as input a network (graph G, possibly with edge capacities, VPN terminals, b^- and b^+ values) and a VPN reservation and routing (values y_e for every edge as well as routing information for all pairs of terminals). The checker should work for all routing alternatives.

2. **Optimal solutions for asymmetric demands.** Implement algorithms that compute a VPN reservation with minimum cost. For tree routing, we already have an implementation. For unsplittable and splittable routing, new algorithms need to be developed (based on linear programming and integer linear programming).

3. **Maximizing the VPN capacity.** Consider the following variant of the VPN reservation problem: The network edges have finite capacities, and we are given the VPN endpoints and their b^-, b^+ values. The goal is to find a VPN reservation and routing that can support every traffic matrix that is valid for the bounds $\lambda \cdot b^+$, $\lambda \cdot b^-$, where $\lambda \in [0, 1]$ should be as close to 1 as possible. Design a good algorithm for this problem and implement it.

4. **Tree routing vs. unsplittable routing.** According to the conclusion of [ILO02], the following question is open: In the symmetric case with infinite capacities, is the minimum cost reservation always obtained using a tree routing?

5. **Approximating unsplittable routing in the asymmetric case.** According to the conclusion of [ILO02], it is an open problem to find an approximation algorithm for the asymmetric case with infinite capacities and unsplittable routing.

6. **Implementation of algorithms.** Implement some of the new and existing algorithms for the VPN reservation problem.

7. **Computational experiments with algorithms.** Compare different algorithms for the same problem variant (and different networks) with respect to solution quality and running time.

8. **Computational experiments with the hose model.** Try to evaluate how much of the network capacity reserved for the VPN is wasted "on average" for a traffic matrix supported by the VPN. Try to evaluate the difference in optimal cost for tree routing, unsplittable routing, and splittable routing.

9. **Directed graphs.** For the problem with finite capacities, it might be interesting to consider directed graphs instead of undirected graphs.

10. **Characterization of supported b^-, b^+ values.** For a given VPN reservation, try to characterize all vectors of b^- and b^+ values for which the reservation is feasible.

11. **Fault tolerance.** Consider the problem of computing a VPN reservation that is fault-tolerant for single edge failures and has minimum cost.

A.2 Format of Data Sets

In section 3, figure 3, three data sets needed for validating a hose are introduced. These sets are again used when building a hose (section 4, figure 14). A possible (file) format of these three sets is given as:

- undirected graph with edge costs and capacities

```
v1   v2   C(v1,v2)    c(v1,v2)
v2   v4   C(v1,v4)    c(v1,v4)
v5   v1   C(v5,v1)    c(v5,v1)
v3   v5   C(v3,v5)    c(v3,v5)
[...]
```

- Set of VPN terminals Q and required bandwidths b_v^- and b_v^+, $v \in Q$.

```
v1   b-(v1)   b+(v1)
v4   b-(v4)   b+(v4)
v5   b-(v5)   b+(v5)
[...]
```

- VPN Reservation and paths between all terminals. Information is divided into three parts[1]:
  - Type of Reservation: (T)ree/(U)nsplittable/(S)plittable
  - Edge reservation y_e between the adjacent nodes u and v of e
  - Routing paths between terminals u and $v \in Q$ as a list of nodes. Appended at the end is a split factor, indicating the percentage of traffic between u and v that is routed through the specified path in a splittable routing hose.

```
S
v1   v2   y(v1,v2)
v2   v4   y(v1,v4)
v5   v1   y(v5,v1)
v3   v5   y(v3,v5)
[...]
---
v1 v2 : v1 v3 v4 v2 splitfactor v1 v5 v2 splitfactor ;
v2 v1 : v2 v5 v1 splitfactor ;
v2 v4 : v2 v4 splitfactor v2 v3 v4 splitfactor v2 v1 v4 splitfactor;
v4 v2 : v4 v2 splitfactor ;
v5 v1 : v5 v2 v4 v3 v1 splitfactor ;
[...]
```

Infinite values—for example for unlimited edge capacities—may be represented by -1. Split factors may also be given as negative values in order to be able to parse a VPN path correctly.

[1] For tree routing, only the first two parts are relevant, since a routing path between two nodes in a tree is unique

A.3 Data Sets and Source Code

The data sets discussed in section 5 and the C++ source code of this project are available for public download at `http://people.ee.ethz.ch/~mrueegg/HoseModel/`. The Code makes heavy use of two commercial libraries:

- LEDA by Algorithmic Solutions Software GmbH (`http://www.algorithmic-solutions.com/enleda.htm`)
- CPLEX by ILOG, Inc. (`http://www.ilog.com/products/cplex/`)

The available source code includes four well commented classes:

- GraphMaker includes those functions that manage a network. These include generating and storing a graph, terminals, and a reservation. Additionally, the class offers the possibility to generate a random VPN out of a given graph.
- FlowChecker is an implementation of all algorithms discussed in section 3 of this report.
- HoseBuilder offers implementations of the algorithms and linear programs described in section 4.
- PipeFinder is a small auxiliary class that was needed for some simulations whose results are shown in section 5.1.

The data sets mentioned above contain mostly preprocessed data used in section 5 and available as MS-Excel worksheets.

References

[CCPS98] William J. Cook, William H Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, New York, 1998.

[DGG+99] N.G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K.K. Ramakrishnan, and Jocobus E. van der Merwe. A Flexible Model for Resource Management in Virtual Private Networks. *Proceedings of ACM SIGCOMM*, 1999.

[dVPSW02] Gustavo de Veciana, Sangkyu Park, Aimin Sang, and Steven Weber. Routing and Provisioning VPNs Based on Hose Traffic Models and/or Constraints. *Allerton Conference on Communication, Control and Computing, Monticello, Illinois*, 2002.

[GD96] Lloyd Greenwald and Thomas Dean. Package Routing in Transportation Networks with Fixed Vehicle Schedules: Formulation, Complexity Results and Approximation Algorithms. *Networks*, 27:81–93, 1996.

[GKK+01] Anupam Gupta, Jon Kleinberg, Amit Kumar, Rejeev Rastogi, and Bülent Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, 2001.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The Ellipsoid Method and its Consequences in Combinatorial Optimization. *Combinatorica*, 1:169–197, 1981.

[Hoc97] Dorit S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, 1997.

[ILO02] Giuseppe F. Italiano, Stefano Leonardi, and Gianpaolo Oriolo. Design of Networks in the Hose Model. *Proceedings of the 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE)*, pages 65–76, 2002.

[IRY02] Giuseppe F. Italiano, Rajeev Rastogi, and Bülent Yener. Restoration Algorithms for Virtual Private Networks in the Hose Model. *Proceedings of IEEE INFOCOM*, 2002.

[KRSY01] Amit Kumar, Rajeev Rastogi, Avi Silberschatz, and Bülent Yener. Algorithms for Provisioning Virtual Private Networks in the Hose Model. *Proceedings of ACM SIGCOMM*, 2001.

[KRSY02] Amit Kumar, Rajeev Rastogi, Avi Silberschatz, and Bülent Yener. Algorithms for Provisioning Virtual Private Networks in the Hose Model. *IEEE/ACM Transactions on Networking*, 2002.

[Max] The Leda Manual - Maximum Flow Algorithms (max-flow). `http://www.algorithmic-solutions.info/leda_manual/max_flow.html`.

[Min] The Leda Manual - Min Cost Flow Algorithms (min-cost-flow). `http://www.algorithmic-solutions.info/leda_manual/min_cost_flow.html`.

[MN99] Kurt Mehlhorn and Stefan Näher. *LEDA , a Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambrigde, 1999.

[PT00] Michal Penn and Moshe Tennenholtz. Constrained Multi-Object Auctions and b-Matching. `http://iew3.technion.ac.il/~moshet/aucmichal.ps`, 2000.

[The] Cor Hurkens - Theory of Combinatorial Optimization - Lecture Notes on Discrete Optimization. `http://www.win.tue.nl/~wscor/OW/2V300/dictaat_eng.html`.