On Assessing Robustness in Transportation Planning *

Apostolos Bessas and Christos Zaroliagis

[1] R.A. Computer Technology Institute, N. Kazantzaki Str., Patras University Campus, 26504 Patras, Greece
[2] Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece
Email: {mpessas,zaro}@ceid.upatras.gr

Abstract. We consider a fundamental problem, called *QoS-aware Multicommodity Flow*, for assessing robustness in transportation planning. It constitutes a natural generalization of the weighted multicommodity flow problem, where the demands and commodity values are elastic to the Quality-of-Service (QoS) characteristics of the underlying network. The problem is also fundamental in other domains beyond transportation planning. In this work, we provide an extensive experimental study of two FPTAS for the QoS-aware Multicommodity Flow Problem enhanced with several heuristics, and show the superiority of a new heuristic we introduce here.

Keywords: QoS-ware Multicommodity Flow, Robust Planning, Demand Elasticity, Packing LP.

1 Introduction

One of the key issues that planners of transport operators in public transportation networks have to deal with concerns the routing of various commodities (customers with common origin-destination pairs) to meet certain demands [13]. A customer, when provided with a non-optimal path (route) due to unavailable capacity, s/he will most likely switch to another operator or even other means of transport and the probability in doing so increases as the QoS (quality of service) drops – actually, as a result of statistical measurements over several years, major European railway companies know quite accurately the percentage of customers they lose in such cases as a function of the path's QoS [8, 13]. To minimize the loss of customers, the value charged for the requested service is usually reduced to make the alternative (worse in QoS) path, offered for that service, attractive. Alternatively, improvements in QoS may increase customer demand and also incur an analogous increase in the pricing policy. Consequently,

* This work was partially supported by the Future and Emerging Technologies Unit of EC, under contracts no. FP6-021235-2 (FP6 IST/FET Open/Project ARRIVAL), and no. ICT-215270 (FP7 ICT/FET Proactive/Project FRONTS).

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2146

transportation planners would like to determine the robustness of their planning models towards such fluctuation of customer demands.

In an earlier work [11, 12] we introduced and studied a combinatorial optimization problem, called *QoS-aware Multicommodity Flow* (MCF), that is fundamental to address robustness issues in transportation planning, as those mentioned above. In the QoS-aware MFC problem, a capacitated directed network $G = (V, E)$ is given, in which we wish to route k commodities to meet certain initial demands. Each commodity i is associated with a specific origin-destination pair (s_i, t_i), a demand d_i and a value v_i representing the *profit* of routing one unit of flow from that commodity. Also, for each commodity i, a weight $wt_i : E \to \mathbb{R}_0^+$ is defined that quantifies the provided *quality of service (QoS)*, when this commodity is routed along an edge e or a path p, where $wt_i(p) = \sum_{e \in p} wt_i(e)$. Smaller weight means better QoS. When a commodity is not routed along its shortest w.r.t. wt_i (optimal w.r.t. QoS) path due to capacity restrictions, then (i) a portion of the demand d_i drops (the worse the QoS of the path, the larger the portion d_i that is lost), and (ii) its value v_i is reduced (the worse the QoS, the larger the reduction). In other words, demands and values are *elastic* to the provided QoS. The objective is to compute the maximum weighted multicommodity flow (sum over all commodities and over all paths of the flow routed from every commodity on each path multiplied by the commodity's value) subject to the QoS-elastic demands and values.

To determine the robustness of their models against fluctuations of customer demands, transportation planners are typically confronted with the following robustness issues in network and line planning:

(i) Which is the maximum profit obtained with the current capacity policy that incurs certain QoS-elastic demands and values?
(ii) How much will this profit improve if the capacity is increased?
(iii) Which is the necessary capacity to achieve a profit above a certain threshold?

A fast algorithm for the QoS-aware MCF problem would allow transportation planners to assess effectively the aforementioned robustness issues by identifying capacity bottlenecks and proceed accordingly.

It is worth mentioning that the QoS-aware MCF problem is also fundamental in applications beyond the transportation domain. For instance, in networking (e.g., multimedia) applications over the internet, or in information dissemination over various communication networks [3]. In such a setting, a "server" (owned by some service provider) sends information to "clients", which retrieve answers to queries they have posed regarding various types of information. Common queries are typically grouped together. Answering a query incurs a cost and a data acquisition time that depends on the communication capacity. When a "client" is provided with a non-optimal service (e.g., long data acquisition time due to capacity constraints), s/he will most likely switch to another provider. On the other hand, the provider may reduce the cost of such a service in order to minimize the loss.

In [11, 12] it was shown that the QoS-aware MCF problem can be formulated as a fractional packing linear program (LP) and a FPTAS for its approximate

solution was provided. The algorithm builds upon the Garg & Könemann (GK) Langrangian relaxation method for fractional packing LPs [5], combined with the phases technique introduced by Fleischer [4], and a new approximation algorithm for the non-additive shortest path (NASP) problem developed in [11, 12], which constitutes the required oracle that identifies the most violated constraint of the dual LP.

In this paper, we present a comparative experimental study for the QoS-aware MCF problem. In particular, we have implemented and compared the following algorithms:

- The FPTAS described in [11, 12] for solving the QoS-MCF problem, using as oracle the FPTAS for NASP developed in the same work.
- The GK approach [4, 5] enhanced with the heuristic methods presented in [2], using as oracles the exact (pseudopolynomial) NASP algorithm in [10] and the approximate NASP in [11, 12].
- The FPTAS in [11, 12] incorporating some of the heuristics in [2], as well as the GK approach, and enhanced both with a new heuristic that we develop.

Our comparative experimental study on synthetic and real-world data shows that the new heuristic method leads to a dramatic improvement in the running time over the original algorithms in [4, 5, 11, 12]. Moreover, the use of the exact NASP routine in the GK approach is considerably faster than the version of the approximate NASP.

The rest of the paper is organized as follows. In Section 2, we define the QoS-aware MCF problem formally and formulate it as a packing linear program. In addition, we present the method proposed by Garg & Könemann [5], its modification by Fleischer [4], as well as an exact and an approximate algorithm for the Non-Additive Shortest Path (NASP) problem that constitutes a fundamental subroutine for solving the QoS-aware MCF problem. In Section 3, we present the algorithms implemented for the QoS-aware MCF problem, and in Section 4 we present the experimental results obtained. We conclude in Section 5.

2 Preliminaries

2.1 The QoS-aware MCF Problem

To formally define the QoS-aware Multicommodity Flow Problem, we have adopted the exposition in [11, 12]. In particular, we are given an n-vertex, m-edge digraph $G = (V, E)$ along with a capacity function $u : E \rightarrow \mathbb{R}_0^+$ on its edges. We are also given a set of k commodities. A commodity i, $1 \leq i \leq k$, is a tuple $(s_i, t_i, d_i, wt_i(\cdot), \mathrm{f}_i(\cdot), \mathrm{v}_i(\cdot))$, where s_i and t_i are the source and sink nodes for the commodity i respectively, $d_i \in \mathbb{R}_0^+$ is the demand of the commodity and $wt_i : E \rightarrow \mathbb{R}_0^+$ is the weight function for commodity i. The weight function quantifies the *Quality of Service* for commodity i (smaller weight means better QoS). For any s_i-t_i path p, $wt_i(p) = \sum_{e \in p} wt_i(e)$. Let $\delta_i(s_i, t_i)$ be the length of the shortest path for commodity i with respect to the weight function $wt_i(\cdot)$.

The non-decreasing function $\mathrm{f}_i : [1, +\infty) \to [0, 1]$ is the elasticity function that determines the portion $\mathrm{f}_i(x)$ of the commodity's demand d_i that is lost, if a path that is x times worse than the shortest path with respect to the weight function $wt_i(\cdot)$ is used; that is, if a units of d_i were supposed to be sent in case the provided path was shortest (optimal), then only $(1 - \mathrm{f_i}(x))a$ units will be shipped through the actually provided (non-optimal) path, while $\mathrm{f_i}(x)a$ units will be lost. Commodity i is also associated with a non-increasing profit function $\mathrm{v}_i : [1, +\infty) \to \mathbb{R}_0^+$, which gives the profit $\mathbf{v}_i(x)$ from shipping one unit of flow of commodity i through a path that is x times worse than the shortest path with respect to the weight function $wt_i(\cdot)$. The objective is to maximize the total profit, i.e., the sum over all commodities and over all paths of the flow routed for every commodity on each path multiplied by the commodity's profit subject to the capacity and demand constraints and with respect to the QoS-elasticity of demands and profits. The above is called the *QoS-aware Multicommodity Flow* problem.

Let $P_i = \{p : p \text{ is a } s_i\text{-}t_i \text{ path}\}$ be the set of candidate paths along which flow of commodity i can be sent and let $X_i(p) \in \mathbb{R}_0^+$ denote the flow of commodity i sent along path p. The definition of the elasticity function implies that for each unit of flow of commodity i routed along p, there are $\frac{1}{1-\mathrm{f_i}(x)}$ units *consumed* from the demand of the commodity. Thus, we define a consumption function $\mathrm{h}_i : [1, +\infty) \to [1, +\infty)$ with $\mathrm{h}_i(x) = \frac{1}{1-\mathrm{f_i}(x)}$. Since $\mathrm{f_i}$ is non-decreasing, $\mathrm{h_i}$ is also non-decreasing. Accordingly, the *consumption* $h_i(p) \geq 1$ of a path p is defined as the amount of demand consumed for each unit of flow routed along p, i.e., $h_i(p) = \mathrm{h_i}\left(\frac{wt_i(p)}{\delta_i(s_i,t_i)}\right)$. Similarly, the *value* $v_i(p)$ of a path p is defined as the profit from routing one unit of flow of commodity i through p, i.e., $v_i(p) = \mathrm{v}_i\left(\frac{wt_i(p)}{\delta_i(s_i,t_i)}\right)$.

Consequently, the QoS-aware MCF problem can be described by the following LP:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{k} \sum_{p \in P_i} v_i(p) X_i(p) \\
s.t. \quad & \sum_{i=1}^{k} \sum_{e \in p, p \in P_i} X_i(p) \leq u(e), \quad \forall e \in E \\
& \sum_{p \in P_i} X_i(p) h_i(p) \leq d_i, \quad \forall i = 1, \ldots, k \\
& X_i(p) \geq 0, \quad \forall i = 1, \ldots, k, \forall p \in P_i
\end{aligned}
$$

The dual LP is as follows:

$$\min \quad D = \sum_{e \in E} l(e)u(e) + \sum_{i=1}^{k} \phi_i d_i \tag{1}$$

$$\begin{aligned} s.t. \quad & l(p) + \phi_i h_i(p) \geq v_i(p), \quad \forall i = 1, \ldots, k, \forall p \in P_i \\ & l(p) \geq 0, \quad \forall p \in P_i, \forall i = 1, \ldots, k, \\ & \phi_i \geq 0, \quad \forall i = 1, \ldots, k \end{aligned} \tag{2}$$

The above primal problem is a *packing linear program*; that is, an LP of the form $\max\{c^{\mathrm{T}}x | Ax \leq b, x \geq 0\}$, where A, b and c are $(M \times N)$, $(M \times 1)$ and $(N \times 1)$ matrices, respectively, the entries of which are all positive.

2.2 The Garg-Könemann Method and its Modification by Fleischer

Garg and Könemann in [5] present an efficient algorithm for approximately solving packing linear programs, based on the assumption that $A(i,j) \leq b(i)$, $\forall i, j$ – which can be achieved by appropriate scaling. They use the dual problem $\min\{b^{\mathrm{T}}y | A^{\mathrm{T}}y \geq c, y \geq 0\}$ to identify the most violated constraint. Then, they increase the corresponding primal variable so as to decrease this violation. The most violated constraint is identified by using an exact oracle.

The algorithm works as follows. Let the length of a column j with respect to the dual variables y be $\texttt{length}_y(j) = \sum_i A(i,j)y(i)/c(j)$ and let $\alpha(y)$ denote the length of the column with the minimum $\texttt{length}$; i.e., $\alpha(y) = \min_j \texttt{length}_y(j)$. Additionally, let $D(y) = b^{\mathrm{T}}y$. Then, the dual problem is equivalent to finding a variable assignment y such that $D(y)/\alpha(y)$ is minimized. Let $\beta = \min_y D(y)/\alpha(y)$ as well.

The algorithm proceeds in iterations. Let y_{k-1} be the dual variables and f_{k-1} be the primal solution at the beginning of the k-th iteration. Let q denote the minimum length column of A (i.e., $\alpha(y_{k-1}) = \texttt{length}_{y_{k-1}}(q)$) and p be the "minimum capacity" row (i.e., $p = \arg\min_i \frac{b(i)}{A(i,q)}$). Then, we increase the primal variable $x(q)$ by an amount $\frac{b(p)}{A(p,q)}$ so that $f_k = f_{k-1} + c(q)\frac{b(p)}{A(p,q)}$. The dual variables are updated as

$$y_k(i) = y_{k-1}(i)\Big(1 + \epsilon \frac{b(p)/A(p,q)}{b(i)/A(i,q)}\Big)$$

where $\epsilon > 0$ is a constant, the value of which depends on the desired approximation ratio. The initial values of the dual variables are $y_0(i) = \delta/b(i)$, where $\delta = (1+\epsilon)\Big((1-\epsilon)M\Big)^{-1/\epsilon}$. For brevity, we denote $\alpha(y_k)$ and $D(y_k)$ by $\alpha(k)$ and $D(k)$ respectively. Thus, $D(0) = M\delta$. The algorithm stops at the first iteration t such that $D(t) \geq 1$.

In [4], Fleischer introduced the concept of phases (for the special case of the Maximum Multicommodity Flow problem, but this technique can be extended to all packing linear programs), where the commodities are considered in a round

robin manner and flow is routed for commodity j, until the length of the shortest s_j-t_j path exceeds $\alpha(1+\epsilon)$. Then, the running time is reduced by a factor of k, since it avoids the k shortest path computations required by [5] for every routing of flow.

2.3 NASP routines

The approximation algorithms for solving the QoS-aware MCF problem that we study in this work identify the most violated constraint of the dual LP by repeatedly calling a subroutine that solves the so-called Non-Additive Shortest Path (NASP) problem. NASP is a generalization of the classical shortest path problem, in which the additivity assumption of the edge costs along paths does not hold. More formally, in NASP, we are given a digraph $G = (V, E)$ and a d-dimensional cost vector $\mathbf{c} : E \to [\mathbb{R}^+]^d$ associating each edge e with a vector of attributes $\mathbf{c}(e)$ and a path p with a vector of attributes $\mathbf{c}(p) = \sum_{e \in p} \mathbf{c}(e)$. We are also given a d-attribute non-decreasing and *non-linear* utility function $U : [\mathbb{R}^+]^d \to \mathbb{R}$. The objective is to find a path p^*, from a specific source node s to a destination t, that minimizes the objective function, i.e., $p^* = \text{argmin}_{p \in P(s,t)} U(\mathbf{c}(p))$, where $P(s,t)$ denotes the set of all s-t paths. It is easy to see that in the case where U is linear, NASP reduces to the classical single-objective shortest path problem. For the general case of non-linear U, it is not difficult to see that NASP is NP-hard. For the case of the QoS-aware MCF problem, it turns out that we need a biobjective ($d = 2$) version of NASP, for which both exact and approximate algorithms are known.

Exact NASP. In [10], a pseudopolynomial algorithm for solving exactly the biobjective version of NASP is presented. This algorithm handles the case where every edge (and hence every path) is associated with two attributes (e.g., cost and resource) and the objective function is of the form $U([x_1, x_2]^T) = U_1(x_1) + U_2(x_2)$, where U_1, U_2 are any two non-linear, convex and non-decreasing functions.

The algorithm consists of three phases:

1. It computes upper and lower bounds of the optimal solution using the Extended Hull Algorithm [10]. The running time of the Extended Hull Algorithm is $O(\log(nRC)(m + n \log n))$, where n is the number of nodes of the graph, m the number of edges and R and C the maximum values of the resource and cost respectively.
2. It prunes the graph by eliminating those nodes and edges that do not lie on the optimal path.
3. It closes the gap between the upper and lower bounds and finds the optimal solution by enumeration.

Although this is a pseudopolynomial algorithm (due to the 3rd phase), the experimental study in [10] revealed that, in the vast majority of instances (98%), Phases 2 and 3 are seldomly executed and the optimal solution is found after

the first phase. Hence, for the vast majority of input instances, the running time of the exact algorithm is bounded by the running time of the Extended Hull algorithm.

Approximate NASP. In [12] an algorithm for finding an approximate solution to the d-objective version of the NASP problem was given, for any $d \geq 2$ and for a very broad class of utility functions. For the biobjective case of NASP we are interested in this work, the algorithm in [12] boils down to the following result, which is an immediate consequence of [12, Theorem 4].

Theorem 1. *[12] Let the utility function of NASP be of the form* $U([x_1, x_2]^T) = x_1 U_1(x_2) + U_2(x_2)$*, where* U_1, U_2 *are any non-negative and non-decreasing functions. Then, for any* $\varepsilon > 0$*, there is an algorithm that computes an* $(1+\varepsilon)$*-approximation to the optimum of NASP in time* $O(n^2 m \frac{\log(nC_1)}{\varepsilon})$*, where* $C_1 = \frac{\max_{e \in E} c_1(e)}{\min_{e \in E} c_1(e)}$.

3 Implemented Algorithms

We have implemented a host of algorithms for the QoS-aware MCF problem. In particular: (1) The FPTAS in [11, 12], using as oracle the FPTAS for NASP developed in [11, 12]. (2) The original GK approach [5] and its modification with phases as suggested by Fleischer [4], using as oracles both the exact algorithm for NASP in [10] and the FPTAS for NASP in [11, 12], enhanced with the heuristics in [2] that were proposed for the classical MCF problem. (3) The FPTAS in [11, 12] incorporating some of the heuristics in [2], as well as the GK approach enhanced with the heuristics in [2], and enhanced both with a new heuristic that we develop. In the rest of this section, we provide a description of these algorithms.

3.1 The FPTAS

The FPTAS in [11, 12] requests that $u(e) \geq 1, \forall e \in E$ and $d_i \geq h_i(p), i = 1, \ldots, k, p \in P_i$. This is enforced by scaling the capacities of the edges and the demands for the commodities by $\min\left\{ \min_{e \in E} u(e), \min_{1 \leq i \leq k} \frac{d_i}{h_i^{max}} \right\}$, where $h_i^{max} = \mathrm{h_i}\Big(\frac{(n-1) \max_{e \in E} wt_i(e)}{\delta_i(s_i, t_i)} \Big)$ is an upper bound for the maximum value of the function $h_i(\cdot)$.

Given an assignment (l, ϕ) for the dual variables, we define the length of a dual constraint as $\texttt{length}_{(l,\phi)}(i,p) = \frac{l(p) + \phi_i h_i(p)}{v_i(p)}$. Then, the most violated constraint of the dual problem is the path of the shortest `length`. We define the length of this path as $\alpha(l, \phi) = \min_{1 \leq i \leq k} \min_{p \in P_i} \texttt{length}_{(l,\phi)}(i,p)$. Initially, $l(e) = \frac{\delta}{u(e)}, \forall e \in E$ and $\phi_i = \frac{\delta}{d_i}, i = 1, \ldots, k$, where $\delta = (1+\epsilon)\Big((1+\epsilon)(m+k)\Big)^{-\frac{1}{\epsilon}}$.

The algorithm is iterative. Initially, all flows are equal to zero. In each iteration the algorithm makes a call to an oracle that returns a commodity i' and

a path $p \in P_{i'}$ that approximately minimizes the function $\texttt{length}_{(l,\phi)}(i,q)$ over all $1 \leq i \leq k$ and $q \in P_i$; that is, $\texttt{length}_{(l,\phi)}(i',p) \leq (1+\epsilon)\alpha(l,\phi)$. Then, the algorithm augments $\Delta = \min\left\{\frac{d_{i'}}{h_{i'}(p)}, \min_{e\in p} u(e)\right\}$ units of flow for the commodity i' along path p and updates the corresponding dual variables l and ϕ by setting $l(e) = l(e)(1+\epsilon\frac{\Delta}{u(e)}), \forall e \in p$ and $\phi_{i'} = \phi_{i'}(1+\epsilon\frac{\Delta h_{i'}(p)}{d_{i'}})$. D is updated accordingly.

The algorithm terminates at the first iteration in which $D = \sum_{e\in E} l(e)u(e) + \sum_{i=1}^{k} \phi_i d_i > 1$. During the course of algorithm it can happen that more flow is sent along an edge than its capacity. It can be proved [4, 5, 11, 12] that the final flow has to be scaled by a factor of $\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$ in order to be feasible. The ratio of the flow sent along an edge and its capacity, during the course of the algorithm, is called the *congestion* of the edge.

The (approximate) oracle that has to be called by the algorithm, in order to find the most violated constraint of the dual, has to (approximately) minimize the function

$$\frac{l(q)+\phi_i h_i(q)}{v_i(q)} = \frac{l(q)+\phi_i \mathrm{h}_i\left(\frac{wt_i(q)}{\delta_i(s_i,t_i)}\right)}{\mathrm{v}_i\left(\frac{wt_i(q)}{\delta_i(s_i,t_i)}\right)}.$$

For a fixed i this requires the solution of a NASP instance with objective function

$$U([x_1,x_2]^{\mathrm{T}}) = \frac{x_1+\phi_i \mathrm{h}\left(\frac{x_2}{\delta_i(s_i,t_i)}\right)}{\mathrm{v}_i\left(\frac{x_2}{\delta_i(s_i,t_i)}\right)}$$

and cost vector $\mathbf{c} = [l, wt_i]^{\mathrm{T}}$. Clearly, the utility function is of the form required by Theorem 1 and hence the approximate algorithm for solving NASP instances can be used.

The calls to this oracle proceed in phases, following the technique introduced in [4]. A lower bound estimation on the current length of the shortest path $\bar{\alpha}$ is maintained. Initially, $\bar{\alpha} = \frac{1}{1+\epsilon}\min_{1\leq i\leq k}\left\{\frac{l(p_i)+\phi_i h_i(p_i)}{v_i(p_i)}\right\}$, where p_i is the path returned from the NASP routine for the specific commodity i. In each phase, the oracle examines the commodities one by one and for each commodity i it returns a path p such that $\frac{l(p)+\phi_i h_i(p)}{v_i(p)} < \bar{\alpha}(1+\epsilon)^2$. As long as there is such a path for commodity i, the oracle sticks to this commodity. When no such path can be found, the algorithm proceeds to the next commodity. After all commodities have been considered in the current phase, it holds that $\alpha(l,\phi) \geq (1+\varepsilon)\bar{\alpha}$ and the algorithm proceeds to the next phase by setting $\bar{\alpha} = \bar{\alpha}(1+\epsilon)$.

We call the above algorithm **TZ-aNASP**. Its complexity is given by the following theorem.

Theorem 2. *[11, 12] There is an algorithm that computes a $\frac{(1+\epsilon)^2}{(1-\epsilon)^2}$-approximation to the QoS-aware Multicommodity Flow problem in time $O((\frac{1}{\epsilon})^3(m+k)\log(m+k)mn^2(\frac{1}{\epsilon}\log(m+k)+\log(nU)))$, where n is the number of nodes, m is the number of edges, k is the number of commodities and $U = \frac{\max_{e\in E} u(e)}{\min_{e\in E} u(e)}$*

3.2 Approximate Algorithms using Heuristic Methods

The second algorithm follows the GK approach for approximately solving packing LPs [5] improved with a few other techniques and heuristic methods. Its main difference with Algorithm *TZ-aNASP* is that now we can use an exact (and not only an approximate) oracle by employing the exact NASP algorithm described in Section 2.3. Moreover, the algorithm terminates as soon as the ratio of the dual solution to the primal is smaller than $1+\omega, \omega < 1$ (it can be proved that this is a valid termination criterion). In addition, we adapt and use a few heuristic methods that were originally proposed in [2] for the classical MCF problem. In the following, let $v^{\max} = \max_i \left\{ \mathrm{v_i}\left(\frac{(n-1)\max_{e \in E} wt_i(e)}{\delta_i(s_i,t_i)} \right) \right\}$ be the upper bound of the maximum value of the functions $\mathrm{v_i}$, over all commodities $1 \leq i \leq k$. We have implemented three methods of updating the best so far dual solution β (recall its definition from Section 2.2).

- We use the best D/α ratio obtained so far.
- We consider the union of all s_i-t_i cuts to obtain an upper bound on the capacity of the multicut (the cut separating all s_i from all t_i), which, when multiplied with $v^{\max}$ is in turn an upper bound on β.
- We keep track of the capacity and the s_i-t_i pairs separated by all cuts encountered in the course of shortest path computations, and run the greedy algorithm for the set cover problem on the collection of cuts. In this reduction, the sets are the cuts, their cost is the capacity of the cut and the elements they cover are the s_i-t_i pairs separated by the cut. The value returned multiplied with $v^{\max}$ is a tighter upper bound for β.

At each time, the smallest value obtained by these three methods is used to update β, if necessary. Furthermore, the amount of flow augmented along a path is equal to $\max\{f_1, \min\{f_2, f_3\}\}$, where f_1, f_2, f_3 are the amounts of flow which, when routed along this path, would cause the length of the path to exceed $\alpha(1+\epsilon)$, the congestion to exceed the maximum congestion, and the length of the path to exceed D/β, respectively. We call this algorithm **GK-H**.

Apart from the above heuristic methods, we can take advantage of the structure of the QoS-aware MCF problem to obtain another upper bound on the dual solution β. In the QoS-aware MCF problem, we are interested in augmenting d_i units of flow for commodity i, $i = 1, \ldots, k$. That is, we want to augment $\sum_{i=1}^{k} d_i$ units of flow in total at most (in case every commodity can use its shortest path w.r.t. $wt_i(\cdot), i = 1, \ldots, k$). Hence, we can use the sum of demands of each commodity multiplied by $v^{\max}$ as an upper bound of the best dual solution (because this is the maximum flow we are interested in sending). We extend the previous algorithm with this method and call the resulted algorithm **GK-HD**.

Additionally, we added the heuristic methods of algorithm *GK-HD* (except for the methods involving cut computations, due to the fact that these computations cannot be added to the approximate NASP routine without incurring extra overhead) to algorithm *TZ-aNASP*, and call the resulting algorithm **TZ-aNASP-HD**.

All the aforementioned algorithms work for the case that the profit function is constant (e.g., $\mathrm{v_i}(x) = 1$, $i = 1, 2, \ldots, k$). In the general case, in which the profit function is non-increasing, only algorithms *TZ-aNASP* and *TZ-aNASP-HD* are applicable. This is due to the fact that the other algorithms use the exact NASP routine, which works, only if the utility function is of the particular form described in Section 2.3.

4 Experimental Results

All algorithms were implemented in `C++` using `g++` (version 3.4.6). Additionally, the LEDA library (version 5.2) was used. The experiments were performed on a computer with two hyper-threaded Intel Xeon processors clocked at 2.8GHz. The total RAM was 4GB.

Two sets of experiments were conducted. In the first set, the profit function was $\mathrm{v_i}(x) = 1$. All algorithms are compared for this first set of data and we want to see, the way that using an approximate NASP routine affects the execution of the algorithms. In the second set the profit function was $\mathrm{v_i}(x) = \frac{1}{x}$ and, so, only algorithms *TZ-aNASP* and *TZ-aNASP-HD* are considered. With this set of experiments, we evaluate the performance of the original algorithms as well as those obtained by incorporating the heuristic methods already described. For all experiments the elasticity function was $\mathrm{f_i}(x) = 1 - \frac{1}{x^2}$, and so the consumption function was $\mathrm{h_i}(x) = x^2$. The total approximation ratio was set to 10%.

4.1 Synthetic Data Sets and Constant Profit

In the first set of experiments, three types of graphs were used to test the above algorithms:

GRID(n, k) These are $n \times n$ (i.e., n^2 nodes) grid graphs with k commodities. These were generated by the corresponding grid generator provided by LEDA. Results were taken for graphs of sizes from 10×10 to 20×20. For the 10×10 to 14×14 graphs the number of commodities was 5. For the rest of the graphs the number of commodities was 10. The capacities of the edges were randomly selected in $[20, 30]$ and the weights of the edges in $[1, 10]$. The demand for each commodity was randomly selected from the range $[1, 10]$. The source nodes were randomly selected from the nodes in the top row and leftmost column of the grid, while the target nodes were selected from the nodes in the bottom row and rightmost column of the grid in such a way that a path connecting the source with the corresponding target node always existed.

GENRMF(α, β) These are graphs consisted of β grid graphs of size $\alpha \times \alpha$. The nodes of each grid graph are connected with nodes of another grid in a random way. Experiments were performed for $(5, 5)$ up to $(15, 10)$ graphs and for 10 commodities. The capacities of the edges were randomly selected in the range $[6, 16]$ and the weights in $[1, 10]$. The demand for each commodity was in $[1, 10]$. Details for the particular graph generator can be found in [6].

NETGEN(n, m, k) These are graphs produced by the netgen generator, which is described in [7]. The generated graphs had n nodes and m edges. In addition, k commodities were used for the graph. The capacities of the edges, the weights of the edges and the demand for each commodity were randomly selected in $[5, 14]$, $[1, 10]$ and $[1, 10]$, respectively.

An initial set of experiments revealed two interesting outcomes: (i) The dominating factor with respect to the running time was the calls to the NASP routines. (ii) There is a huge difference in performance between the exact NASP (Section 2.3) and the approximate NASP routine (Section 2.3), especially for large sizes of graphs, in favor of the former. This difference is justified by the theoretical running times of the two algorithms in combination with the chosen numerical values and the form of the utility function. Moreover, the implementation of the exact NASP algorithm uses a few heuristics methods that considerably speed up its execution. However, the approximate algorithm handles a broader selection of instances w.r.t. numerical values and utility functions.

In view of the above, we will report our experimental results with respect to the number of NASP calls (exact or approximate) performed by the algorithms.

To investigate the influence of the phases technique in [4], we start by comparing the original algorithm of Garg and Könemann (using the exact NASP routine), referred to as **GK-orig**, and the same algorithm enhanced with the phases technique, referred to as **GK-F**. The results for the case of grid graphs are shown in Table 1. Similar results were obtained with the other graph families (GENRMF and NETGEN).

Graph(n, k)	Algorithm GK-orig	Algorithm GK-F
GRID(10, 5)	60450	78880
GRID(11, 5)	61770	82200
GRID(12, 5)	64380	85030
GRID(13, 5)	66170	85953
GRID(14, 5)	68110	89352
GRID(15, 10)	277690	181874
GRID(16, 10)	283920	185770
GRID(17, 10)	289950	190001
GRID(18, 10)	296340	192066
GRID(19, 10)	300360	195570
GRID(20, 10)	305730	200118

Table 1. Comparison of algorithms GK-orig and GK-F in GRID graphs with all profit functions set to 1. The number of NASP calls is presented.

We observe that for small graphs *GK-orig* is faster than *GK-F*. This happens, because, in order to achieve the same total approximation error, a smaller value of ϵ is used for the second algorithm, since the use of the phases introduces another factor of error. That is, the approximation ratio of the first algorithm is $\frac{1}{(1-\epsilon)^2}$,

while the approximation ratio of the second algorithm is $\frac{1+\epsilon}{(1-\epsilon)^2}$. However, when the size of the graph and the number of commodities increase, we can see that the second algorithm is quite faster than the first one, because the improvement gained from the technique of phases is more significant than using a smaller value for ϵ for the total running time, resulting in a decrease in the required NASP calls. This is expected, as the number of NASP calls in the original GK approach is $O(\frac{1}{\epsilon^2}km\log n)$ [5] and the use of the phases technique reduces the number of NASP calls to $O(\frac{1}{\epsilon^2}m\log n)$ [4].

In Table 2 the number of NASP calls is presented for algorithms *GK-F*, *TZ-aNASP*, *GK-H*, *GK-HD* and *TZ-aNASP-HD* for graphs of type GRID for sizes up to 14×14 and 5 commodities. Experiments were also performed for larger grid graphs (up to 20×20) with 10 commodities and for graphs of type NETGEN and GENRMF and we obtained similar results.

Graph(n, k)	GK-F	TZ-aNASP	GK-H	GK-HD	TZ-aNASP-HD
GRID(10, 5)	78880	90023	3426	1036	1105
GRID(11, 5)	82200	93389	4018	1909	2130
GRID(12, 5)	85030	97000	3781	856	877
GRID(13, 5)	85953	99036	2813	360	489
GRID(14, 5)	89352	102090	3084	337	340

Table 2. Comparison of algorithms in GRID graphs with all profit functions set ot 1. The number of NASP calls is presented.

One can see that *TZ-aNASP* is inferior to *GK-F*. This is due to the smaller value of the constant ϵ that has to be selected for the first algorithm, in order for the total error to be the same in the two algorithms (the approximation ratios are $\frac{(1+\epsilon)^2}{(1-\epsilon)^2}$ and $\frac{1+\epsilon}{(1-\epsilon)^2}$ respectively). On the other hand, *TZ-aNASP* can handle a broader range of problem instances.

A second crucial observation from Table 2 is that the algorithms *GK-H*, *GK-HD* and *TZ-aNASP-HD* that use the heuristic methods described in Section 3.2 outperform dramatically algorithms *GK-F* and *TZ-aNASP*. Applying the heuristic methods has a beneficial effect on the number of NASP calls required to find an approximate solution, since a path is used to send flow for as long as possible, approaching faster the optimal solution.

A third important observation concerns the impact of the new heuristic introduced in Section 3.2 and is based on the demands. We do not only observe a dramatic improvement in the performance of *TZ-aNASP*, but also in that of *GK-H*. This is due to the fact that by taking advantage of the extra knowledge of demands in the problem a better upper bound can be computed faster, resulting in more flow being sent along a path per NASP computation.

To further elaborate on the effect of using the heuristic based on the demands, we report, in Tables 3, 4 and 5, the experimental results of algorithms *GK-H* and *GK-HD* on all synthetic data used, when $\mathrm{v_i}(x) = 1$, $i = 1, 2, \ldots, k$. We can see

that in all cases, the heuristic based on the demands results in an improvement in the number of NASP calls required. The improvement depends on the structure of the graph (e.g., for grid graphs the improvement is greater than for graphs of type netgen) as well as the numerical data used.

Graph(n, k)	GK-H	GK-HD
GRID(10, 5)	3426	1036
GRID(11, 5)	4018	1909
GRID(12, 5)	3781	856
GRID(13, 5)	2813	360
GRID(14, 5)	3084	337
GRID(15, 10)	7252	1549
GRID(16, 10)	6383	1388
GRID(17, 10)	6746	1345
GRID(18, 10)	6874	955
GRID(19, 10)	6850	1644
GRID(20, 10)	5880	1391

Table 3. Comparison of algorithms GK-H and GK-HD in GRID graphs with all profit functions set to 1. The number of NASP calls is presented.

Graph(α, β)	GK-H	GK-HD
GENRMF(5, 5)	5937	1572
GENRMF(6, 5)	6445	4817
GENRMF(7, 5)	6367	2203
GENRMF(8, 5)	7057	5718
GENRMF(9, 5)	7963	4611
GENRMF(10, 10)	6403	1894
GENRMF(11, 10)	6841	3102
GENRMF(12, 10)	7183	2513
GENRMF(13, 10)	8095	4751
GENRMF(14, 10)	6998	3073
GENRMF(15, 10)	7878	3249

Table 4. Comparison of algorithms GK-H and GK-HD in GENRMF graphs with $k = 10$ commodities and all profit functions set to 1. The number of NASP calls is presented.

4.2 Synthetic and Real-world Data Sets with Non-increasing Profit

The second set of experiments was conducted on grid graphs of sizes 10×10 to 14×14 with 5 commodities, and on real-world data from the German railways comparing algorithms *TZ-aNASP* and *TZ-aNASP-HD*, which are the only ones

Graph(n, m, k)	GK-H	GK-HD
NETGEN(100, 1000, 10)	14272	13561
NETGEN(200, 1300, 15)	21892	21709
NETGEN(200, 1500, 15)	19621	18985
NETGEN(200, 2000, 20)	32024	30766
NETGEN(300, 4000, 15)	23246	21330
NETGEN(500, 3000, 30)	43104	41260
NETGEN(700, 30000, 50)	76092	70018

Table 5. Comparison of algorithms GK-H and GK-HD in NETGEN graphs with all profit functions set to 1. The number of NASP calls is presented.

that apply to this case of profit functions. The underlying network in the first set of real-world data (R1) has 280 nodes and 354 edges, in the second set (R2) 296 nodes and 393 edges and in the third set (R3) 319 nodes and 452 edges. The data are taken from the software platform LinTim [9]. For all sets of real-world data, demands were in [4000, 10000], the wt functions corresponded to the length of the edges of the train network ranging from a few hundred meters to more than 100 Km and the capacity of an edge was in [800, 1600] . All profit functions were set to $\frac{1}{x}$. The results are presented in Tables 6 and 7.

Again one notes the significant drop in the number of NASP calls required, when the heuristic methods are used. We observe that *TZ-aNASP-HD* is from 14 up to 54 times faster than *TZ-aNASP*. This is, because the heuristic methods allow for a path to be used multiple consecutive times in order to send flow, resulting in considerably fewer NASP calls by the algorithm, and hence achieving a huge speedup.

5 Conclusions

In this paper an experimental study for the QoS-aware MCF problem was presented. Algorithms for this problem that follow the Garg & Könemann method have to rely on solving an instance of a NASP problem. Using the exact NASP routine results in fewer NASP calls than by using an approximate one (in order to obtain the same approximation ratio for the algorithms). However, the algorithms that use the approximate NASP routine are more general and enforce less restrictions on the form of the problem. The results show clearly that incorporating the described heuristic methods, and especially the new heuristic based on the demands, yields significant improvements in the running time of the algorithms. The difference in NASP calls of algorithms *TZ-aNASP* and *TZ-aNASP-HD*, or *GK-orig* and *GK-HD* is dramatic and, since the bottleneck in the running time is the computation of the non-additive shortest path, there was an accordingly great decrease in the running time of the corresponding algorithms.

Graph(n, k)	TZ-aNASP	TZ-aNASP-HD
GRID$(10, 5)$	90538	1630
GRID$(11, 5)$	93389	2128
GRID$(12, 5)$	97000	930
GRID$(13, 5)$	99036	624
GRID$(14, 5)$	101801	605
GRID$(15, 10)$	208412	2356
GRID$(20, 10)$	228131	9291

Table 6. Comparison of algorithms TZ-aNASP and TZ-aNASP-HD in GRID graphs with all profit functions set to $\frac{1}{x}$. The number of NASP calls is presented.

Data Set	Commodities	TZ-aNASP	TZ-aNASP-HD	Speedup
R1	5	68336	2022	33
	10	120500	3229	37
	15	185171	6984	26
	20	216902	12615	17
R2	5	61241	1598	38
	10	119855	4059	29
	15	162239	5354	30
	20	235181	16832	14
R3	5	74563	1357	54
	10	165782	3894	42
	15	247540	6548	37
	20	247540	5911	41

Table 7. Comparison of algorithms TA-aNASP and TZ-aNASP-HD on the available sets of real-world data with all profit functions set to $\frac{1}{x}$. The number of NASP calls and the speedup is presented.

References

1. Ravindra K. Ahuja, Thomas L. Magnati, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.
2. Garima Batra, Naveen Garg, and Garima Gupta. Heuristic improvement for computing maximum multicommodity flow and minimum multicut. In *Algorithms — ESA 2005*, volume 3669 of *Lecture Notes in Computer Science*, pages 35–46. Springer Berlin / Heidelberg, 2005.
3. A. Datta, D. Vandermeer, A. Celik, and V. Kumar. Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users. *ACM Transactions on Database Systems*, 24(1):1–79, 1999.
4. Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. volume 14, pages 505–520. Society for Industrial and Applied Mathematics, 2000.
5. Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 300–309. IEEE Computer Society, 1998.
6. Donald Goldfarb and Michael D. Grigoriadis. A computational comparison of the dinic and network simplex methods for maximum flow. *Annals of Operations Research*, 13(1):81–123, December 1988.
7. D. Klingman, A. Napier, and J. Stutz. NETGEN — A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–821, 1974.
8. PIN project (Projekt Integrierte Netzoptimierung). Deutsche Bahn AG, 2000.
9. Michael Schachtebeck and Anita Schöbel. Lintim — a toolbox for the experimental evaluation of the interaction of different planning stages in public transportation. Technical Report ARRIVAL-TR-0206, ARRIVAL Project, February 2009.
10. George Tsaggouris and Christos Zaroliagis. Non-additive shortest path. In *Algorithms — ESA 2004*, volume 3221 of *Lecture Notes in Computer Science*, pages 822–234. Springer Berlin, 2004.
11. George Tsaggouris and Christos Zaroliagis. QoS-aware Multicommodity Flows and Transportation Planning. In *Proc. 6th Workshop on Algorithmic Methods and Models for Optimization of Railway — ATMOS 2006*, 2006.
12. George Tsaggouris and Christos Zaroliagis. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Theory of Computing*, 45(1):162–186, 2009.
13. F. Wagner. Challenging Optimization Problems at Deutsche Bahn. AMORE Workshop (invited talk), 1999.