A decomposition based on path sets for the Multi-Commodity k-splittable Maximum Flow Problem

Gamst, Mette

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

A decomposition based on path sets for the Multi-Commodity k-splittable Maximum Flow Problem

Report 6.2013

DTU Management Engineering

Mette Gamst

March 2013

DTU Management Engineering
Department of Management Engineering

A decomposition based on path sets for the Multi-Commodity k-splittable Maximum Flow Problem.

M. Gamst

March 15, 2013

Abstract

The Multi-Commodity k-splittable Maximum Flow Problem routes flow through a capacitated graph such that each commodity uses at most k paths and such that the total amount of routed flow is maximized. The problem appears in telecommunications, specifically when considering *Multi-Protocol Label Switching*. In the literature, the problem is solved to optimality using branch-and-price algorithms built on path-based Dantzig-Wolfe decompositions. This paper proposes a new branch-and-price algorithm built on a path set-based Dantzig-Wolfe decomposition. A path set consists of up to k paths, each carrying a certain amount of flow. The new branch-and-price algorithm is implemented and compared to the leading algorithms in the literature. Results for the proposed method are competitive and the method even has best performance on some instances. However, the results also indicate some scaling issues.

Keywords: *Branch and price; Dantzig-Wolfe decomposition; Multi-commodity flow; k-splittable; Combinatorial optimization*

1 Introduction

The $\mathcal{NP}$-hard Multi-Commodity k-splittable Maximum Flow Problem (denoted MCkMFP) works on a capacitated graph and consists of maximizing the amount of routed flow for a given set of commodities. Edge capacities must be satisfied and each commodity can use at most k paths between its source and target vertex. The problem has application in telecommunications, specifically in *Multi-Protocol Label Switching* (denoted MPLS), see Evans and Filsfils [7] for more details on the MPLS.

The Multi-Commodity k-splittable Flow Problem (denoted MCkFP) is a well-known problem in the literature and has been solved heuristically, approximately and optimally. Baier et al. [1] presented the MCkFP. They proved that the problem is $\mathcal{NP}$-hard and proposed approximation algorithms for the Maximum Budget-Constrained Single- and Multi-Commodity k-splittable Flow Problems. Much work has been performed on non-exact solution methods for (variants of) the MCkFP, see e.g. [4, 6, 8] for a selection of heuristics and [12, 13, 14, 16, 17] for a selection of approximation algorithms.

Here we seek to solve the MCkMFP to optimality, hence we focus our literature review on exact methods. A number of Dantzig-Wolfe decompositions are proposed in the literature.

Truffot and Duhamel [18, 19] decomposed the problem into a path-based formulation, where each column represented a path for a commodity. They added an index to each variable for keeping track of which of the $h = 1, \ldots, k$ paths, the variable represented. The pricing problem was a polynomial shortest path problem. The path-based master problem allowed symmetry in the solution space, which affected the performance negatively: the number of generated columns and the size of the branch-and-price tree both grew large. Another path-based formulation by Gamst et al. [9] eliminated some of this symmetry by leaving out the variable index $h = 1, \ldots, k$. The resulting branch-and-price algorithm showed better performance, however, it suffered from complicated branching and from bounding problems on the number of used paths per commodity. Gamst and Petersen [10] improved the latter branch-and-price algorithm by proposing a new branching rule and by adding cuts to the master problem. Despite better computational results, the algorithm still suffered from the same bottlenecks: complicated branching and poor bounding on the number of used paths per commodity. A different exact solution approach from the literature consists of solving an edge-based formulation of the closely related Maximum Concurrent k-splittable Flow Problem in a branch-and-bound scheme, see Caramia and Sgalambro [3]. Branching consisted of fixed edge usage imposed by cuts. The approach outperformed standard MIP-solvers, but suffered from large branch-and-bound trees.

In this paper we propose a new Dantzig-Wolfe decomposition for the MCkMFP. The decomposition is based on path sets; a column in the master problem consists of up to k paths each carrying a fixed amount of flow for a given commodity. The pricing problem generates up to k paths for a given commodity and assigns flow to the paths – this corresponds to the $\mathcal{NP}$-hard single-commodity k-splittable flow problem. The goal of the path set-based decomposition is to remedy the bottleneck of the path-based branch-and-price algorithms from the literature by letting the pricing problem bound the number of used paths and by simplifying branching in most cases.

The paper is organized as follows. First in Section 3, the new path set-based branch-and-price algorithm is presented. The section explains the master problem, how to find an incumbent, how to solve the $\mathcal{NP}$-hard pricing problem heuristically and optimally, how to reach feasibility early through primal heuristics, and how to branch in order to ensure feasible solutions. The proposed branch-and-price algorithm is computationally evaluated and results are presented and analyzed in Section 4. Finally, Section 5 concludes the paper.

2 Mathematical formulation

Given is a graph $G = (V, E, L, k)$ consisting of a set of vertices V, a set of edges E, and a set of commodities L. Each edge $(ij) \in E$ has capacity $u_{ij} > 0$, and each commodity $l \in L$ consists of a source vertex s_l and a target vertex t_l. Finally, k is an upper bound on the number of paths, each commodity can use. Note that if $k = 1$, the problem reduces to the Multi-Commodity Unsplittable Flow Problem, see e.g. Barnhart et al. [2]. If $k \geq |E|$, the problem reduces to the Linear Multi-Commodity Flow Problem, see Baier et al. [1].

Let $x_{ij}^{hl} \geq 0$ be the amount of flow on edge $(ij) \in E$ for the h'th path of commodity $l \in L$. Note that $h \in \{1, \ldots, k\}$. Similarly, $y_{ij}^{hl} \in \{0, 1\}$ denotes whether or not edge $(ij) \in E$ is part of the h'th path of commodity $l \in L$. Finally, variable $z^{hl} \geq 0$ holds the amount of flow on the h'th path of commodity $l \in L$. The MCkMFP is formulated as:

$$\max \quad \sum_{l \in L} \sum_{h=1}^{k} z^{hl} \tag{1}$$

s. t.

$$\sum_{j \in V} x_{ij}^{hl} - \sum_{j \in V} x_{ji}^{hl} = \begin{cases} z^{hl} & \text{if } i = s_l \\ -z^{hl} & \text{if } i = t_l \\ 0 & \text{otherwise} \end{cases} \quad \begin{array}{l} \forall i \in V,\ \forall l \in L, \\ \forall h \in \{1, \ldots, k\} \end{array} \tag{2}$$

$$\sum_{j \in V} y_{ij}^{hl} - \sum_{j \in V} y_{ji}^{hl} = \begin{cases} 1 & \text{if } i = s_l \\ -1 & \text{if } i = t_l \\ 0 & \text{otherwise} \end{cases} \quad \begin{array}{l} \forall i \in V,\ \forall l \in L, \\ \forall h \in \{1, \ldots, k\} \end{array} \tag{3}$$

$$\sum_{l \in L} \sum_{h=1}^{k} x_{ij}^{hl} \leq u_{ij} \qquad \forall (ij) \in E \tag{4}$$

$$x_{ij}^{hl} - u_{ij} y_{ij}^{hl} \leq 0 \qquad \begin{array}{l} \forall (ij) \in E, \forall l \in L, \\ \forall h \in \{1, \ldots, k\} \end{array} \tag{5}$$

$$\sum_{j \in V} y_{ij}^{hl} \leq 1 \qquad \begin{array}{l} \forall (ij) \in E, \forall l \in L, \\ \forall h \in \{1, \ldots, k\} \end{array} \tag{6}$$

$$x_{ij}^{hl} \geq 0 \qquad \begin{array}{l} \forall (ij) \in E, \forall l \in L, \\ \forall h \in \{1, \ldots, k\} \end{array} \tag{7}$$

$$y_{ij}^{hl} \in \{0, 1\} \qquad \begin{array}{l} \forall (ij) \in E, \forall l \in L, \\ \forall h \in \{1, \ldots, k\} \end{array} \tag{8}$$

$$z^{hl} \geq 0 \qquad \begin{array}{l} \forall h \in \{1, \ldots, k\}, \\ \forall l \in L \end{array} \tag{9}$$

The objective function (1) maximizes the total sum of routed flow. Constraints (2) ensure flow conservation for the flow variables, and constraints (3) ensure flow conservation for the decision variables. Constraints (4) make sure that edge capacities are never violated. Whenever an edge holds flow for a path for a commodity, then both the flow and decision variables must be set. This is ensured in constraints (5). Sub-tours are eliminated in constraints (6), which forbid a path to leave a vertex more than once. The constraints also make sure that each of the k paths for a commodity is unsplittable. Finally, bounds (7)-(9) force variables to take on feasible values.

3 Solution approach

A new Dantzig-Wolfe decomposition [5] is proposed for solving the MCkMFP. The idea behind the decomposition is to use route configurations, i.e., to let the pricing problem generate and assign flow to at most k paths for each commodity. The path-based decompositions of the MCkMFP from the literature give polynomial pricing problems and complex master problems with bounding issues on the number of used paths, see [9, 10, 18]. Using path sets moves complexity from the master problem to the pricing problem: bounding the number of used paths is removed from the master problem, but the pricing problem becomes $\mathcal{NP}$-hard.

Jaumard et al. [11] considered route configurations instead of paths for the Routing and Wavelength Assignment Problem, where pairs of vertices must be connected and each edge can

carry at most W paths. They showed that a branch-and-price algorithm based on path sets eliminated symmetry in the solution space and increased performance significantly compared to the classic, path-based formulation.

Let R be the set of generated path sets and R^l the set of generated path sets for commodity $l \in L$. Each column consists of a path set $r \in R$ based on up to k paths with given flow for a given commodity. Let $x_r^l \in \{0,1\}$ be a binary variable indicating whether or not path set $r \in R$ for commodity $l \in L$ is part of a solution. Also, the constant δ_r^{ij} denotes the total amount of flow traveling on edge $(ij) \in E$ in the path set $r \in R$, and the constant δ_r denotes the total amount of flow routed by configuration $r \in R$

The master problem is:

$$\max \quad \sum_{l \in L} \sum_{r \in R^l} \delta_r x_r^l \tag{10}$$

$$\text{s.t.} \quad \sum_{l \in L} \sum_{r \in R^l} \delta_r^{ij} x_r^l \leq u_{ij} \quad \forall (ij) \in E \tag{11}$$

$$\sum_{l \in L} \sum_{r \in R^l} x_r^l \leq 1 \qquad \forall l \in L \tag{12}$$

$$x_r^l \in \{0,1\} \qquad \forall l \in L,\ \forall r \in R^l \tag{13}$$

The objective function (10) maximizes the amount of routed flow. The first constraints (11) ensure that edge capacities are never violated. The second constraints (12) say that each commodity can use at most one path set. Note that not all commodities have to be routed in order to maximize the total amount of routed flow. Finally, the bounds (13) force all variables to take on feasible values.

3.1 Incumbent

Before starting the column generation iterations, we need an initial solution. This is found using first a heuristic, and as a last resort by using *fake* edges. The heuristic is a randomized local search procedure, which iteratively finds paths in a reduced graph [8]. The heuristic is fast and often capable of finding a good solution. In the case where the heuristic is unable to find a feasible solution, fake edges are used: the source and target of each commodity are connected by a fake edge with capacity ϵ, where $\epsilon > 0$ is a suitably small number. Flow on any fake edges in the final solution is subtracted the solution value. Also, we ensure that fake edges can never be part of any column from the pricing process.

3.2 Pricing problem

The pricing problem must decide the amount of routed flow; the resulting routed flow is expressed as the constants δ_r (total amount of routed flow) and δ_r^{ij} (total amount of routed flow on edges $(ij) \in E$) in the master problem. Let $\pi_{ij} \geq 0$ and $\lambda_l \geq 0$ be the dual variables of constraints (11) resp. (12). Also, let variable $f_{ij} \geq 0$ denote the amount of flow on edge $(ij) \in E$. The total amount of flow routed by a configuration for commodity $l \in L$ is expressed as the sum of flow emanating from the source vertex $\sum_{(s_l j) \in E} f_{s_l j}$. The reduced

cost for commodity $l \in L$ is:

$$\begin{aligned} \bar{c}_r^l &= \sum_{(s_l j)\in E} f_{s_l j} - \sum_{(ij)\in E} \pi_{ij} f_{ij} - \lambda_l \geq 0 \\ \Rightarrow \quad & \sum_{(s_l j)\in E} f_{s_l j} - \sum_{(ij)\in E} f_{ij}\pi_{ij} \geq \lambda_l \\ \Rightarrow \quad & \sum_{(ij)\in E} \widetilde{\pi_{ij}} f_{ij} \geq \lambda_l \end{aligned} \tag{14}$$

where

$$\widetilde{\pi_{ij}} = \begin{cases} 1 - \pi_{ij} & \forall (ij) \in E : i = s_l \\ -\pi_{ij} & \text{otherwise} \end{cases}$$

The pricing problem consists of generating and assigning flow to up to k paths such that the reduced costs (14) are maximized. This corresponds to the $\mathcal{NP}$-hard single-commodity k-splittable maximum flow problem: consider the case where $\pi_{ij} = 0,\ \forall (ij) \in E$. Then the pricing problem is to find k paths such that $\sum_{(s_l j)\in E} f_{s_l j} > \lambda_l$, i.e., the total amount of routed flow must be maximized.

3.2.1 Heuristic

The pricing problem is solved heuristically using an algorithm very similar to the incumbent heuristic mentioned in Section 3.1. The only modification is that instead of iteratively finding any path in a reduced graph, we find the shortest path defined on the negated reduced cost edge weights $-\widetilde{\pi_{ij}}$. Edges emanating from the source vertex s_l for commodity $l \in L$ may have negative edge weight: $-(1 - \pi_{ij})$ when $\pi_{ij} < 1$, but since all other edges have non-negative edge weights $\pi_{ij}, \pi_{ij} \geq 0$, we avoid negative weight cycles by deleting all edges going into the source.

3.2.2 Exact solution approach.

When the heuristic cannot find columns with positive reduced cost, the pricing problem is solved to optimality by applying CPLEX on a mathematical formulation. Let $y_{ij}^h \in \{0, 1\}$ denote whether or not edge $(ij) \in E$ is part of the h'th path, where $h = 1, \ldots, k$, and $f_{ij}^h \geq 0$ be the amount of flow traveling on edge $(ij) \in E$ for the h'th path. Using the previously

introduced notation, the pricing problem for a commodity l is:

$$\max \quad \sum_{(ij)\in E} \widetilde{\pi_{ij}} \sum_{h=1}^{k} f_{ij}^h \tag{15}$$

$$\text{s. t.} \quad \sum_{h=1}^{k} f_{ij}^h \leq u_{ij} \qquad \forall (ij) \in E \tag{16}$$

$$\sum_{j\in V} f_{ij}^h - \sum_{j\in V} f_{ji}^h = 0 \qquad \forall i \in V \backslash \{s_l, t_l\}, \forall h \in \{1,\ldots,k\} \tag{17}$$

$$\sum_{j\in V} y_{ij}^h - \sum_{j\in V} y_{ji}^h \begin{cases} \leq 1 & \text{if } i = s_l \\ \geq -1 & \text{if } i = t_l \\ = 0 & \text{otherwise} \end{cases} \qquad \forall i \in V, \forall h \in \{1,\ldots,k\} \tag{18}$$

$$f_{ij}^h - u_{ij} y_{ij}^h \leq 0 \qquad \forall (ij) \in E, \forall h \in \{1,\ldots,k\} \tag{19}$$

$$\sum_{j\in V} y_{ij}^h \leq 1 \qquad \forall i \in V, \forall h \in \{1,\ldots,k\} \tag{20}$$

$$\sum_{(ij)\in E} \widetilde{\pi_{ij}} \sum_{h=1}^{k} f_{ij}^h \geq \lambda_l + \epsilon \tag{21}$$

$$f_{ij}^h \geq 0 \qquad \forall (ij) \in E, \forall h \in \{1,\ldots,k\} \tag{22}$$

$$y_{ij}^h \in \{0,1\} \qquad \forall (ij) \in E, \forall h \in \{1,\ldots,k\} \tag{23}$$

The objective function (15) maximizes the left hand side of the reduced cost (14). Constraints (16) make sure that edge capacities are satisfied. Constraints (17) and (18) ensure flow conservation for both the flow and the binary variables. Constraints (19) force a binary variable to be one, when flow is traveling on the corresponding edge for the corresponding path h. Constraints (20) eliminate sub-tours by ensuring that a path leaves a vertex at most once. Constraint (21) forces the solution to have positive reduced cost according to (14) and with $\epsilon > 0$ being a suitably small number. Finally, bounds (22) and (23) force variables to take on feasible values.

3.2.3 Symmetry

The h-indices in formulation (15)-(23) introduce symmetry in the pricing problem solution space. Let $p(l,h)$ be the h'th path ($h = \{1,\ldots,k\}$) for commodity $l \in L$. Path $p(l,h)$ consists of a set of connected edges going from vertex s_l to vertex t_l. Now, consider a solution for commodity $l \in L$ with paths $p^a(l,1), p^b(l,2)$ and $p^c(l,3)$. Equivalent solutions are $p^a(l,1), p^b(l,3)$, $p^c(l,2)$, and $p^a(l,2), p^b(l,1)$, $p^c(l,3)$. Obviously, more equivalent solutions exist.

To reduce the number of equivalent solutions, symmetry breaking constraints are introduced:

$$\sum_{(s_l j)\in E} \left(f_{s_l j}^{h+1} - f_{s_l j}^{h} \right) \leq 0 \quad \forall h = \{1,\ldots,k-1\} \tag{24}$$

The constraints (24) eliminate identical solutions, when paths carry different amounts of flow by forcing paths carrying more flow to take on smaller values of h. The number of constraints (24) to add is $k-1$, which is relatively small. The constraints do, however, not break symmetry in the case where paths carry the same amount of flow.

3.3 Primal heuristics

Reaching feasible solutions early may improve the bounds of the branch-and-price algorithm and hence help prune larger parts of the search tree. In the following, we propose two primal heuristics to transform fractional solutions into feasible solution.

3.3.1 Heuristic for routes with same paths

A fractional solution S may consist of fractional routes using the same paths but carrying different amounts of flow. In this case, a heuristic transforms the solution into being integer by re-using the paths and by changing the flow according to the fractional usage of the routes. This is shown in Algorithm 1, which initializes edge usage, and in Algorithm 2, which contains the overall heuristic.

Algorithm 1 initializes edge usage for each commodity $l \in L$. As input the Algorithm takes the set of edges E, the commodity to consider l, the partial heuristic solution $\bar{R}$ and the fractional solution S. The Algorithm considers each route configuration and if a path set is fully used ($x_r = 1$), then it is stored as part of the heuristic solution. If it is only partially used ($0 < x_r < 1$) then the routed flow on each edge in the fractional solution is saved. If two path sets use different paths, then the Algorithm terminates early with a fail. If the Algorithm succeeds, i.e., all fractional path sets use the same paths, then it returns the set of used edges $\bar{E}$ and a vector of edge flow in the fractional solution $\mathbf{f} = \{f_{ij}, (ij) \in \bar{E}\}$.

Algorithm 1 Init_edge_usage_first($E, l, \bar{R}, S$)

```
1: for (each edge (ij) ∈ E) do
2:    f_ij ← 0
3: end for
4: for (each route r ∈ R for commodity l) do
5:    if x_r = 1 then
6:       R̄ ← r
7:    end if
8:    if 0 < x_r < 1 then
9:       for (each (p, f_p) ∈ r with f_p > 0) do
10:         for (each edge (ij) ∈ p) do
11:            if (r is not first route for commodity l and f_ij = 0) then
12:               return (null, null)
13:            end if
14:            f_ij ← f_ij + f_p · x_r
15:         end for
16:      end for
17:   end if
18: end for
19: return (Ē, f)
```

Now Algorithm 2 tries to build a new path set using the output of Algorithm 1. The Algorithm takes as input the fractional solution S. For each commodity it initializes edge usage by calling Algorithm 1. Then it considers each of the paths for the commodity in the fractional solution and it greedily assigns as much flow as possible, using the output of

Algorithm 1. The new path set is stored as part of the heuristic solution.

In this way a new column may be generated heuristically. The column is added to the master problem, and a new branch-and-price iteration is begun.

Algorithm 2 First_primal_heuristic(S)

```
1: R̄ ← empty solution
2: for (each commodity l ∈ L) do
3:    (Ē, f) ← Init_edge_usage_first(E, l, R̄, S)
4:    if (Ē = null) then
5:       return null
6:    end if
7:    r̄ ← empty path set
8:    Let r ∈ R for commodity l be first path set with 0 < x_r < 1
9:    for (each (p, f_p) ∈ r) do
10:      f_new ← ∞
11:      for (each edge (ij) ∈ p) do
12:         f_new ← min{f_new, f_ij}
13:      end for
14:      if (f_new > 0) then
15:         r̄ ← r̄ ∪ {(p, f_new)}
16:         for (each edge (ij) ∈ p) do
17:            f_ij ← f_ij − f_new
18:         end for
19:      end if
20:   end for
21:   R̄ ← R̄ ∪ r̄
22: end for
23: return R̄
```

The heuristic has running time $\mathcal{O}(|L||\bar{R}|k|E|)$, where $|\bar{R}|$ is the number of fractional routes, whose paths combined use $\mathcal{O}(k|E|)$ edges.

3.3.2 Heuristic for routes with different paths

A fractional solution S may consist of fractional routes using different paths but edge usage sums to integers. In this case, a heuristic tries to transform the solution into being integer by finding $\leq k$ paths for each commodity, spanning all used edges from the source to the target of the commodity. The flow on the edges of the $\leq k$ paths must be equal to the flow in the fractional solution. The heuristic works as illustrated in Algorithm 3, which initializes edge usage, and in Algorithm 4, which contains the overall heuristic.

Algorithm 3 initializes edge usage for each commodity. The Algorithm takes as input the set of edges E, the commodity to consider l, the partial heuristic solution $\bar{R}$ and the fractional solution S. The Algorithm considers all route configurations for the given commodity. If a path set is fully used ($x_r = 1$), then it is stored as part of the heuristic solution. If it is only partially used ($0 < x_r < 1$) then the Algorithm saves the routed flow in the fractional solution, the degree of usage in the fractional server, and the edge itself. Finally, the Algorithm checks for integer usage; if an edges is used only partially, then the Algorithm fails. If the Algorithm

succeeds, i.e., edge usage is integer, then it returns the set of used edges in the fractional solution $\bar{E}$, a vector of edge flow in the fractional solution $\mathbf{f} = \{f_{ij}, (ij) \in \bar{E}\}$, and a vector of the degree of edge usage in the fractional solution $\mathbf{y} = \{y_{ij}, (ij) \in \bar{E}\}$.

Now back in Algorithm 4, a new path set must be built. The Algorithm takes as input the fractional solution S. For each commodity it initializes edge usage by calling Algorithm 3. A depth first search is then run on the saved edges from the source to the target of the commodity. If such a path exists, then we greedily assign as much flow as possible according to the saved flow from above. When the available flow on an edge becomes zero, then the edge is removed. The procedure repeats until either all edges are removed, until k paths are found, or until no more paths can be found. If all edges are used (i.e., all edges are removed), then the path set is saved. Otherwise the Algorithm fails.

Algorithm 3 Init_edge_usage_second$(E, l, \bar{R}, S)$

$\bar{E} \leftarrow \emptyset$
for (each edge $(ij) \in E$) **do**
  $f_{ij} \leftarrow 0$
  $y_{ij} \leftarrow 0$
end for
for (each route $r \in R$ for commodity l) **do**
  if $x_r = 1$ **then**
    $\bar{R} \leftarrow r$
  end if
  if $0 < x_r < 1$ **then**
    for (each $(p, f_p) \in r$ with $f_p > 0$) **do**
      for (each edge $(ij) \in p$) **do**
        $f_{ij} \leftarrow f_{ij} + f_p \cdot x_r$
        $y_{ij} \leftarrow y_{ij} + x_r$
        $\bar{E} \leftarrow \bar{E} \cup \{(ij)\}$
      end for
    end for
  end if
end for
for (each edge $(ij) \in \bar{E}$) **do**
  if $(0 < y_{ij} < 1)$ **then**
    return (**null**, **null**, **null**)
  end if
end for
return $(\bar{E}, \mathbf{f}, \mathbf{y})$

The heuristic has running time $\mathcal{O}(|L||\bar{R}(|k|E| + |V|))$: removing all edges not used in S requires an investigation of all edges on all paths from all routes in the fractional solution, which takes $\mathcal{O}(|\bar{R}|k|E|)$ time. Furthermore, the depth first search has running time $\mathcal{O}(|V| + |E|)$. If the heuristic succeeds, then a column with the found paths is generated and added to the master problem, and a new branch-and-price iteration is begun.

Algorithm 4 Second_primal_heuristic(S)

$\bar{R} \leftarrow$ empty solution
for (each commodity $l \in L$) **do**
  $(\bar{E}, \mathbf{f}, \mathbf{y}) \leftarrow$ Init_edge_usage_second(E, l, S)
  if ($\bar{E}$ = **null**) **then**
    return null
  end if
  $\bar{r} \leftarrow$ empty path set
  while $\bar{E}$ is non-empty and $|\bar{r}| < k$ **do**
    $p \leftarrow$ depth-first-search from s_l to t_l on edges $\bar{E}$
    if (p is empty) **then**
      break
    else
      $f_{new} \leftarrow \infty$
      for (each edge $(ij) \in p$) **do**
        $f_{new} \leftarrow \min\{f_{new}, f_{ij}\}$
      end for
      $\bar{r} \leftarrow \bar{r} \cup \{(p, f_{new})\}$
      for (each edge $(ij) \in p$) **do**
        $f_{ij} \leftarrow f_{ij} - f_{new}$
        if ($f_{ij} = 0$) **then**
          $\bar{E} \leftarrow \bar{E} \backslash \{(ij)\}$
        end if
      end for
    end if
  end while
  if ($\bar{E}$ is non-empty) **then**
    return null
  end if
  $\bar{R} \leftarrow \bar{R} \cup \bar{r}$
end for
return $\bar{R}$

3.4 Branching

In the following, we analyze situations in which fractional solutions occur and consider how the branching strategies affect the pricing problem.

The relaxed master problem solution is infeasible in the following situations:

1. A commodity is not fully routed, i.e., $0 < \sum_{r \in R} \delta_r^l x_r < 1$ for commodity $l \in L$. This is handled by the primal heuristic described in Section 3.3.1.
2. More than one path set is used for a commodity, i.e., $\delta_{r_1}^l x_{r_1} > 0$ and $\delta_{r_2}^l x_{r_2} > 0$ for commodity $l \in L$ and some path sets $r_1, r_2 \in R$

In the second case, we first consider edge usage. If an edge $(ij) \in E$ is used partially by paths for a commodity l, we systematically forbid or force edge usage:

$$\sum_{\substack{r \in R: \\ (ij) \in r}} \delta_r^l x_r = 0 \quad \text{vs.} \quad \sum_{\substack{r \in R: \\ (ij) \in r}} \delta_r^l x_r = 1 \tag{25}$$

If all edge usage is integer, the solution may still be fractional. We know that all fractional variables do not use the same paths, because this is handled by the first primal heuristic. Hence the fractional solution must use different paths. The branching strategy is then to forbid and force path paths as introduced by Gamst and Petersen [10].

Given a commodity and a sets of paths. Let a unique edge sequence (also denoted an edge sequence) be a set of connected edges emanating from the commodity source, traveling until it shares no edges with any other path. Given that the path sets do not use the same paths, then we must be able to identify unique edge sequences.

Let $b_1, b_2, \ldots b_h$ denote unique edge sequences, which are not all contained in all path sets. We generate $h + 1$ branching children. In child $i : 1 \leq i \leq h$ usage of edge sequences $b_j, j = \{1, 2, \ldots, i\}$ is forced and usage of edge sequences $b_j, j = \{i + 1, i + 2, \ldots, h\}$ is forbidden. This means that in the first child usage of edge sequence b_1 is forced and usage of edge sequences $b_j, j = \{2, 3, \ldots, h\}$ is forbidden. In the second child, usage of edge sequences b_1 and b_2 is forced and usage of edge sequences $b_j, j = \{3, 4, \ldots, h\}$ is forbidden, and so on.

The final branching child $h + 1$ forbids usage of all $b_1, b_2, \ldots b_h$ edge sequences. The branching strategy is imposed through cuts in the master problem. Let $i : 1 \leq i \leq h$ be the branching child and let the constant δ_r^{bl} denote whether or not path set $r \in R$ for commodity $l \in L$ uses edge sequence b. The cut is:

$$\sum_{j=1}^{i} \sum_{r \in R} \delta_r^{b_j l} x_r = 1 \quad \text{vs.} \quad \sum_{j=1}^{i} \sum_{r \in R} \delta_r^{b_j l} x_r = 0 \tag{26}$$

The branching restrictions must be taken into account by the pricing problem. Branching cuts (25) and (26) are added to the master problem and must also be considered by the pricing problem. Let their dual variables be $\beta_{ij} \in \mathbb{R}$ and $\beta_b \in \mathbb{R}$, respectively. The reduced cost is rewritten to:

$$\Rightarrow \sum_{(ij) \in E} (\widetilde{\pi_{ij}} f_{ij} - \beta_{ij} w_{ij}) - \beta_b w_b \geq \lambda_l \tag{27}$$

where

$$\widetilde{\pi_{ij}} = \begin{cases} 1 - \pi_{ij} & \forall (ij) \in E : i = s_l \\ -\pi_{ij} & \text{otherwise} \end{cases}$$

and $w_{ij} \in \{0,1\}$ denotes whether or not edge $(ij) \in E$ is used by any path in the generated column, and $w_b \in \{0,1\}$ indicates if path b is part of the generated column.

The pricing heuristic handles the new reduced cost through small modifications. Forbidden edges are removed from the graph, while forced edges have their weight set to $\max\{0, \widetilde{\pi_{ij}} - \beta_{ij}\}$ (note that this is an approximation of the edge weight on the left hand side of (27)). Forbidden paths are handled by the pricing heuristic by transforming the graph into a graph with forbidden sub paths. This can be done in polynomial time in the input size using the graph extension algorithm by Villeneuve and Desaulniers [20]. Forced paths are pre-stored by the heuristic, which then tries to find another $k - b$ paths (where b is the number of forced paths for the commodity).

The exact pricing algorithm handles the new reduced cost by including them in the mathematical formulation. Forced paths are by default in the solution and the exact algorithm instead tries to find $k - b$ paths where b is the number of forced paths for the commodity. The resulting formulation is:

$$\max \quad \sum_{(ij)\in E} \left(\widetilde{\pi_{ij}} \sum_{h=1}^{k} f_{ij}^h - \beta_{ij} w_{ij} \right) \tag{28}$$

s. t. Constraints (16)–(20), (24) and bounds (22)–(23)

$$\sum_{(ij)\in E} \left(\widetilde{\pi_{ij}} \sum_{h=1}^{k} f_{ij}^h - \beta_{ij} w_{ij} \right) \geq \lambda_l \tag{29}$$

$$\sum_{(ij)\in b} y_{ij}^h \leq |b| - 1 \qquad \forall b \in B_p, \forall h \in \{1,\dots,k\} \tag{30}$$

$$\sum_{h=1}^{k} y_{ij}^h \leq 0 \qquad \forall (ij) \in B_{ij}^- \tag{31}$$

$$\sum_{h=1}^{k} y_{ij}^h \geq 1 \qquad \forall (ij) \in B_{ij}^+ \tag{32}$$

$$y_{ij}^h - M f_{ij}^h \leq 0 \qquad \forall (ij) \in E, \forall h \in \{1,\dots,k\} \tag{33}$$

$$\sum_{h=1}^{k} y_{ij}^h - k w_{ij} \leq 0 \qquad \forall (ij) \in B_{ij} \tag{34}$$

$$w_{ij} \in \{0,1\} \qquad \forall (ij) \in E, \forall h \in \{1,\dots,k\} \tag{35}$$

The objective function maximizes the reduced cost (27). Note that duals for forced and forbidden paths are left out: forced paths are already included so their dual cost can be considered a constant and thus be left out of the objective function. Forbidden paths are never used, as explained below for constraints (30), hence their dual cost is never paid and can thus be left out of the objective function. Constraints (16)-(20) and bounds (22)-(23) are unchanged from the original pricing problem formulation. Constraints (30) consider forbidden paths: B_p denotes the set of forbidden paths, each path is denoted $b \in B_p$, and the number of edges on the path is denoted by $|b|$. Constraints (31) and (32) forbid or force usage of edges: B_{ij}^- denotes the set of forbidden edges and B_{ij}^+ the set of forced edges. Constraints (33) ensure

Name	$\|V\|$	$\|E\|$	$\|L\|$
Random5-35	5	35	1
Random10-45	10	45	1
Random15-60	15	60	1
Random20-140	20	140	1
tg10-2	12	40	1
tg20-2	22	80	1
tg40-1	42	154	1
tg40-5	42	154	1
tg80-1	82	322	1
Random10-40	10	40	3
Random11-42	11	42	11
Random20-80	20	80	20
Random22-56	22	56	22

Table 1: Sizes of test instances. First column denotes the instance name, then follows the number of vertices, the number of edges, and finally the number of commodities.

that if the binary variable is set, then the corresponding edge on the corresponding path h also carries flow. Here M is some suitably large number to allow edges to carry less than one unit of flow. Constraints (34) set variable w_{ij} to 1, if the edge $(ij) \in E$ is part of any path. The variable w_{ij} is needed in the objective function to calculate the correct reduced cost, and it is bounded in (35).

The extra branching constraints do not add significant complexity to the pricing problem, which is already $\mathcal{NP}$-hard.

4 Computational evaluation

The proposed branch-and-price algorithm is implemented using the framework COIN [15] with `ILOG CPLEX 12.1` as LP-solver. Computations concerning the selection of branching candidates and branching children are handled by COIN. The computational evaluation is performed on a 16-core Intel® Xeon® CPU X5550, 2.67GHz machine with 24 GB of RAM.

The computational evaluation is performed on benchmark instances from the literature [18]: single- and multi-commodity `Random` instances are randomly generated and the single-commodity `tg` instances are generated by the Transit Grid generator[1] using topologies from transportation networks. See Table 1 for details.

Results for testing the new branch-and-price algorithm are seen in tables 2 - 4, where the new algorithm is denoted RBP. The results are compared to the leading algorithms from the literature: the 3-index branch-and-price algorithm (3BP) by Truffot et al. [18], the 2-index branch-and-price algorithm branching on edge sequences (2BP) by Gamst et al. [9] and the 2-index branch-and-price algorithm branching and cutting on edge sequences (2BCP) by Gamst and Petersen [10]. Note that results for the algorithms from the literature are reached using the same computational environment except from having `ILOG CPLEX 10.2` as standard LP-solver.

The first columns of the tables contain the name of the benchmark instance, the value of k and the optimal solution value. Then follows for each algorithm the size and depth of the

[1] `http://www.informatik.uni-trier.de/~naeher/Professur/research/generators/maxflow/tg/index.html`

branch-and-bound tree, and the time spent on solving the instances. Results marked with "-" indicate that the instance could not be solved because of excessive time or memory usage. Best results are marked with **bold** and summed in the bottom row.

As can be seen from the results, the new branch-and-price algorithm does not branch much, which is a significant improvement compared to the results from the literature, where branching constituted a problem. The algorithm is competitive with the leading algorithms from the literature on many instances, but it displays some difficulty solving instances with larger values of k and where k restricts the solution. Solving the pricing problem to optimality using CPLEX is the reason for the large time usage.

			3BP				2BP				2BCP				RBP			
Problem	**k**	**z***	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**
Random5-35	2	128	1	0	14	**0.00**	1	0	7	**0.00**	1	0	7	**0.00**	1	0	6	0.04
	3	182	1	0	27	0.01	1	0	9	**0.00**	1	0	9	**0.00**	1	0	8	0.06
	4	223	13	6	48	0.01	1	0	12	**0.00**	1	0	12	**0.00**	1	0	12	0.27
	5	262	19	9	60	0.03	1	0	12	**0.00**	1	0	12	**0.00**	1	0	14	0.48
	6	297	21	10	78	0.03	1	0	14	0.01	1	0	14	**0.00**	1	0	17	1.22
	7	326	67	12	98	0.10	1	0	14	**0.00**	1	0	14	**0.00**	1	0	21	0.70
	8	326	1	0	104	**0.00**	1	0	11	0.01	1	0	11	**0.00**	1	0	21	0.29
Random10-45	2	142	5	2	15	**0.01**	4	1	9	**0.01**	8	2	9	**0.01**	1	0	3	0.17
	3	209	9	3	33	**0.02**	21	3	15	0.03	20	3	12	**0.02**	1	0	4	0.31
	4	260	45	17	68	0.08	411	12	24	0.56	34	4	20	**0.03**	1	0	5	1.00
	5	306	369	22	102	0.80	23599	18	34	44.96	40	4	20	**0.07**	1	0	7	1.88
	6	345	973	26	137	2.90	>427099	>26	39	-	135	6	26	**0.22**	1	0	9	3.98
	7	381	4281	36	219	16.55	>354551	>22	46	-	313	8	34	**0.64**	1	0	11	53.52
	8	413	22985	43	265	102.51	>431299	>29	46	-	606	9	40	**1.31**	1	0	13	-
	9	429	>110199	>58	380	-	>388228	>26	60	-	2507	11	46	**5.97**	1	0	15	-
	10	451	>104999	>57	448	-	>456699	>41	74	-	2355	12	46	**5.91**	1	0	37	-
Random15-60	2	163	1	0	16	**0.00**	1	0	8	**0.00**	1	0	8	**0.00**	1	0	8	0.03
	3	221	9	3	34	**0.02**	41	6	15	0.06	12	2	12	**0.02**	1	0	12	0.29
	4	248	111	10	70	0.32	>100454	>26	50	-	111	6	20	**0.22**	1	0	15	3.66
	5	268	557	18	101	551.83	>176599	>29	52	-	322	7	29	**0.76**	1	0	18	8.77
	6	287	419	21	135	1.59	>277801	>31	45	-	354	9	30	**0.79**	1	0	24	24.21
	7	295	19097	35	194	72.91	>387565	>23	49	-	836	10	27	**1.74**	1	0	34	95.74
	8	301	>88799	>47	231	-	>413343	>33	55	-	4995	11	30	**11.32**	1	0	38	-
	9	306	>153099	>51	229	-	>547079	>28	48	-	2263	11	19	**4.42**	1	0	34	-
Random20-140	2	158	1	0	14	**0.00**	1	0	7	**0.00**	1	0	7	**0.00**	1	0	7	0.15
	3	228	1	0	30	0.02	1	0	11	**0.00**	1	0	11	**0.00**	1	0	12	1.64
	4	253	9935	31	103	75.25	>41444	>42	68	-	90	18	67	**1.04**	1	0	15	5.50
	5	274	>39999	>41	146	-	>68299	>66	87	-	819	22	51	**12.65**	1	0	18	14.23
	6	294	>30199	>61	184	-	>60299	>86	107	-	>14106	>32	113	-	1	0	23	-
	7	311	>28999	>70	227	-	>75894	>46	91	-	>14299	>32	109	-	1	0	28	-
	8	319	>30599	>80	267	-	>94699	>101	120	-	4028	22	29	**52.95**	1	0	44	-
	9	325	>39599	>93	315	-	>108990	>63	105	-	130	9	25	**0.32**	1	0	55	-
	10	327	2907	109	326	19.15	>272685	>49	68	-	17	3	22	**0.02**	1	0	66	-
	11	327	1325	86	301	8.75	49	3	22	**0.03**	20	5	20	**0.03**	1	0	12	0.05
Best						**7**				**10**				**32**				**0**

Table 2: Results from solving the single-commodity `Random` instances exactly.

			3BP				2BP				2BCP				RBP			
Problem	**k**	**z***	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**
tg10-2	2	557	215	13	26	0.21	355	14	10	0.21	41	5	11	**0.04**	1	0	7	0.18
	3	716	553	19	58	0.70	39505	20	28	32.49	53	5	15	**0.06**	1	0	10	0.34
	4	815	83	17	52	0.10	6	1	8	**0.00**	5	1	8	**0.00**	1	0	9	0.01
	5	815	1	0	40	**0.00**	1	0	8	**0.00**	1	0	8	**0.00**	1	0	9	0.02
tg40-1	2	750	5	2	16	0.07	4	1	10	**0.02**	10	3	12	0.07	1	0	5	-
	3	908	>9999	>40	96	-	>83282	>61	94	-	231	11	21	**3.32**	1	0	7	-
	4	994	>7799	>57	143	-	>82770	>45	64	-	893	18	33	**25.15**	1	0	9	-
	5	1004	15	7	65	0.09	703	27	20	1.41	11	2	18	**0.03**	1	0	11	0.11
	6	1004	1	0	96	0.03	29	3	13	**0.02**	43	6	13	0.07	1	0	11	0.39
tg40-5	2	828	>20599	>46	80	-	>64248	>45	57	-	144	9	23	1.49	1	0	3	**0.74**
	3	1062	>17299	>59	139	-	>77103	>44	65	-	276	8	22	**4.20**	1	0	4	-
	4	1078	181	47	68	0.61	>148934	>22	50	-	1520	21	22	26.53	1	0	5	**0.17**
	5	1078	3	1	90	**0.03**	61	4	16	0.04	76	20	16	1.72	1	0	5	0.22
Best						**2**				**4**				**8**				**2**

Table 3: Results from solving the `tg` instances exactly.

			3BP				2BP				2BCP				RBP			
Problem	**k**	**z***	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**	**size**	**depth**	**vars**	**time**
Random10-40	2	194	1	0	26	**0.00**	34	5	21	0.04	4	1	18	0.01	1	0	11	0.17
	3	258	183	18	80	0.39	213	12	24	0.18	50	6	23	**0.06**	1	0	19	0.86
	4	293	695	36	129	2.23	2956	16	41	4.25	112	7	32	**0.20**	1	0	23	3.35
	5	309	1989	30	176	7.91	>253716	>25	56	**1.06**	561	12	39	**1.06**	9	3	32	4.26
	6	318	15905	36	253	73.84	>610006	>24	59	-	1294	13	60	**2.63**	1	0	39	17.16
	7	321	>153199	>56	286	-	>335959	>26	54	-	26182	18	47	**57.10**	3	1	40	-
	8	323	113	37	299	**0.52**	2008	14	37	1.23	2051	15	36	2.43	1	0	43	-
	9	323	333	49	318	1.38	11	1	32	**0.01**	18	5	32	0.02	1	0	46	-
Random11-42	2	343	1	0	50	**0.01**	7	2	27	**0.01**	6	1	27	**0.01**	1	0	23	0.04
	3	344	1	0	75	**0.00**	1	0	26	**0.00**	1	0	26	**0.00**	1	0	23	0.10
	4	344	1	0	100	**0.00**	1	0	26	**0.00**	1	0	26	**0.00**	1	0	36	0.13
Random20-80	2	553	3	1	100	0.03	4	1	50	0.02	4	1	51	**0.01**	5	2	38	0.59
	3	584	1063	24	188	6.14	57	7	59	**0.16**	1020	16	62	3.45	3	1	46	1.84
	4	601	5599	33	277	40.05	1041	10	60	**2.02**	>81550	>548	601	-	1	0	54	4.31
	5	617	13291	44	340	117.96	4363	14	66	**7.35**	49695	34	67	198.61	23	10	59	22.74
	6	621	>48999	>40	380	-	3998	11	63	**6.42**	32552	29	58	100.08	7	3	73	58.71
	7	626	413	37	412	3.48	17	2	57	**0.02**	116	14	57	0.22	1	0	64	8.36
	8	626	1	0	440	0.03	1	0	57	**0.01**	1	0	57	**0.01**	1	0	69	3.90
Random22-56	2	389	11	4	81	0.02	10	3	42	0.02	9	3	42	**0.01**	17	4	35	0.27
	3	407	1	0	108	**0.01**	1	0	41	**0.01**	1	0	41	**0.01**	3	1	43	0.19
	4	407	1	0	144	0.01	1	0	41	**0.00**	1	0	41	**0.00**	3	1	34	0.25
Best						**6**				**13**				**13**				**0**

Table 4: Results from solving the multi-commodity instances exactly.

Future work on the algorithm should focus on a faster, exact algorithm for the pricing problem. An obvious strategy is to Dantzig-Wolfe decompose the pricing problem (such that the pricing problem constitutes a nested decomposition), but this is not necessarily a good idea. The nested decomposition results in a master problem very similar to that for the path-based decomposition [10]. The nested master problem introduces large branching trees, and one could question if the nested path set- and path-based decompositions would produce better results than having just a path-based decomposition.

Future work on the proposed method could also focus on a different decomposition, where flow is handled in the master problem and the pricing problem simply identifies the paths of a path set. This reduces the complexity of the pricing problem, which would become similar to the polynomial k-shortest path problem. It is, however, not trivial to formulate a master problem, which handles flow on path sets in a good way. The master problem must ensure flow conservation, either through path variables or original edge variables. Either strategy could bring back issues from existing algorithms, i.e. large branch-and-bound trees and symmetry in the solution space.

5 Conclusion

This paper presented a new path set-based decomposition and a corresponding branch-and-price algorithm for the multi-commodity k-splittable maximum flow problem. A path set consists of up to k paths, each carrying a given amount of flow. The pricing problem generates path sets, which corresponds to solving the single-commodity k-splittable flow problem. The master problem merges the path sets into an overall feasible solution with respect to edge capacities. The strength of the path set-based branch-and-price algorithm is smaller search trees and – to a great extend – simpler branching strategies when compared to the leading branch-and-price algorithm from the literature [10]. The proposed branch-and-price algorithm furthermore includes a method for generating the incumbent, two primal heuristics and a method for solving the pricing problem heuristically.

Computational results showed that the proposed branch-and-price algorithm is competitive with the leading algorithms from the literature on smaller instances, but displays some scaling issues. Solving the $\mathcal{NP}$-hard pricing problem to optimality is the main issue. Future work should thus focus on faster ways of solving the pricing problem to optimality. Future work could also focus on a different path set-based decomposition, where flow is handled by the master problem.

References

[1] G. Baier, E. Kohler, and M. Skutella. On the k-splittable flow problem. *Algorithmica*, 42:231–248, 2005.

[2] C. Barnhart, C. A. Hane, and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

[3] M. Caramia and A. Sgalambro. An exact approach for the maximum concurrent k-splittable flow problem. *Optimization Letters*, 2:251–265, 2008.

[4] M. Caramia and A. Sgalambro. A fast heuristic algorithm for the maximum concurrent k-splittable flow problem. *Optimization Letters*, 4:37–55, 2010.

[5] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[6] C. Duhamel and P. Mahey. Multicommodity flow problems with a bounded number of paths: A flow deviation approach. *Networks*, 49(1):80–89, 2007.

[7] J. W. Evans and C. Filsfils. *Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice*. Morgan Kaufmann, 2007.

[8] M. Gamst. A local search heuristic for the multi-commodity k-splittable maximum flow problem. *Optimization Letters*, 2013. Published online.

[9] M. Gamst, P. N. Jensen, D. Pisinger, and C. Plum. Two- and three-index formulations of the minimum cost multicommodity k-splittable flow problem. *European Journal of Operations Research*, 202(1):82–89, 2010.

[10] M. Gamst and B. Petersen. Comparing branch-and-price algorithms for the multicommodity k-splittable maximum flow problem. *European Journal of Operational Research*, 217(2):278–286, 2012.

[11] B. Jaumard, C. Meyer, and B. Thiongane. On column generation formulations for the RWA problem. *Discrete Applied Mathematics*, 157:1291–1308, 2009.

[12] R. Koch, M. Skutella, and I. Spenke. Approximation and complexity of k-splittable flows. In *Approximation and Online Algorithms, Third International Workshop, WAOA*, pages 244–257, 2005.

[13] R. Koch, M. Skutella, and I. Spenke. Maximum k-splittable s, t -flows. *Theory of Computing Systems*, 43(1):55–66, 2008.

[14] S. G. Kolliopoulos. Minimum-cost single-source 2-splittable flow. *Information Processing Letters*, 94(1):15–18, 2005.

[15] R. Lougee-Heimer. The common optimization interface for operations research. *IBM Journal of Research and Development*, 47:57–66, 2003.

[16] M. Martens and M. Skutella. Flows on few paths: algorithms and lower bounds. *Networks*, 48(2):68–76, 2006.

[17] F. Salazar and M. Skutella. Single-source k-splittable min-cost flows. *Operations research letters*, 37:71–74, 2009.

[18] J. Truffot and C. Duhamel. A branch and price algorithm for the k-splittable maximum flow problem. *Discrete Optimization*, 5(3):629–646, 2008.

[19] J. Truffot, C. Duhamel, and P. Mahey. Using branch-and-price to solve multicommodity k-splittable flow problems. In *Proceedings of International Network Optimization Conference (INOC)*, 2005.

[20] D. Villeneuve and G. Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165(1):97–107, 2005.

The Multi-Commodity k-splittable Maximum Flow Problem routes flow through a capacitated graph such that each commodity uses at most k paths and such that the total amount of routed flow is maximized. The problem appears in telecommunications, specifically when considering Multi-Protocol Label Switching. In the literature, the problem is solved to optimality using branch-and-price algorithms built on path-based Dantzig-Wolfe decompositions. This paper proposes a new branch-and-price algorithm built on a path set-based Dantzig-Wolfe decomposition. A path set consists of up to k paths, each carrying a certain amount of flow. The new branch-and-price algorithm is implemented and compared to the leading algorithms in the literature. Results for the proposed method are competitive and the method even has best performance on some instances. However, the results also indicate some scaling issues.

ISBN 978-87-92706-99-7

DTU Management Engineering
Department of Management Engineering
Technical University of Denmark

Produktionstorvet
Building 424
DK-2800 Kongens Lyngby
Denmark
Tel. +45 45 25 48 00
Fax +45 45 93 34 35

www.man.dtu.dk