Multicommodity network flow with jump constraints

Jean-François Maurras, Yann Vaxes*

Faculté des Sciences de Luminy, Laboratoire d’informatique de Marseille,
163 Avenue de Luminy, 13288 Marseille, France

1. Introduction

Multicommodity network flow is an important problem that occurs in many areas of operations research. Given a set of commodities which are to be carried between certain nodes of a capacitate network, the general problem is to find an optimal distribution of the traffic such that all the needs are satisfied without violating any capacity constraint. This problem can be formulated as a large linear program whose structure can be used to speed up the simplex method [1, 4, 5, /]: price directive and resource directive decomposition, partitioning, specific technique for the GUB structure, etc. The most promising adaptation of the simplex method designed to solve multicommodity flow problem seems to be the primal partitioning technique for the arc-chain formulation introduced by Farvolden et al. [3]. The aim of the present paper is to show how the same approach can be used to solve a little more complicated problem, arising from telecommunication network optimization: the multicommodity flow problem with jump constraints. This problem differs from the previous one by the fact that commodities can only flow through paths with no more than a fixed number of arcs. Furthermore, a new result and shortest proofs dealing with properties of solutions produced by primal partitioning are presented. We also note that it is possible to extend the same partitioning idea to the node-arc formulation of the problem but without taking into account the jump constraints.

2. Problem statement

A network can be represented by a directed graph G = (X, U) with n = |X| nodes and m = |U| arcs. For each arc u = (i, j), the capacity b_u is the maximal amount of commodities that can be sent from node i to node j using the line u and c_u the linear cost associated with this line. Let K be a set of commodity, for each k ∈ K, r^k is the...
value of the demand from the source $s^k$ to the sink $t^k$. Let $\delta$ be the maximum number of arcs in a path used by the commodity. Let $p^k$ be the number of distinct paths with no more than $\delta$ arcs joining $s_k$ to $t_k$ and $\{Q^k_1, Q^k_2, \ldots, Q^k_{p_k}\}$ the set of these paths. Let $x^k_j$ denote the flow of commodity $k$ through the path $Q^k_j$, $k \in K$ and $j = 1, \ldots, p^k$. With these notations, the arc-chain formulation of a *jump constrained* multicommodity network flow problem is as follows:

\[
\begin{align*}
\text{Min} & \quad \sum_{k \in K} \sum_{j=1}^{p^k} g^k_j x^k_j \\
\text{subject to} & \quad \sum_{k \in K} \sum_{j \in Q^k_i} x^k_j + e_u = b_u \quad \text{for all } u \in U, \\
& \quad \sum_{j=1}^{p^k} x^k_j = r^k \quad \text{for all } k \in K, \\
& \quad x^k_j \geq 0 \quad \text{for all } k \in K, \quad j = 1, \ldots, p^k,
\end{align*}
\]

where $e_u$ is the slack variables associated with the capacity $b_u$. In the next section, we will recall the basic ideas of the primal partitioning algorithm [3].

### 3. Primal partitioning method

#### 3.1. Partitioning of the basis

Given a basis $B$, a commodity $k$ will be called *unsplitted* if exactly one path variable $x^k_j$ belongs to $B$, and *split* otherwise. An arc will be called *saturated* if the slack variable associated with its capacity constraint is not present in the basis and *unsaturated* otherwise. Let $\mathcal{U}$ be the set of capacity constraints for unsaturated arcs, $\hat{\mathcal{U}}$ the set of flow conservation constraints for split commodities and $\mathcal{X}$ the set of flow conservation constraints for unsplitted commodities. Let $\mathcal{V}$ be the set of basic slack variables, $\hat{\mathcal{V}}$ the set of basic path variables for split commodities, and $\hat{\mathcal{C}}$ the set of basic path variables for unsplitted commodities. With appropriate line and column permutations, the basis $B$ can be presented as follows:

\[
B = \left( \begin{array}{ccc}
I^\mathcal{V}_\mathcal{U} & P & Q \\
R & S & I^\mathcal{V}_\mathcal{X} \\
I^\mathcal{C}_\mathcal{V} & I^\mathcal{C}_\mathcal{X} & \mathcal{C}
\end{array} \right)
\]

where $I^\mathcal{V}_\mathcal{U}$ and $I^\mathcal{V}_\mathcal{X}$ are identity matrices. Such a partitioning can be used to significantly speed up computations of linear systems in the simplex method using the following
equations:

\[
\begin{align*}
&\mathcal{U} \cup \mathcal{K} \begin{pmatrix} a' \\ a'' \\ a''' \end{pmatrix} = A \quad \text{and} \quad \mathcal{U} \begin{pmatrix} d' \\ d'' \\ d''' \end{pmatrix} = D,

&B_d = a \quad \Rightarrow \quad \begin{cases} 
    d' = a' - Q_a''' - P_d''', \\
    R_d'' = a'' - S_d''', \\
    d''' = a''',
\end{cases}
\]

where the column vectors \(a\) and \(d\) denote, respectively, the entering column and its current representation. By solving the system of equations \(y_B = g_B\), we can calculate the dual variables which are used to select the entering variable.

\[
\begin{align*}
&\mathcal{U} \mathcal{G} \begin{pmatrix} y' \\ y'' \\ y''' \end{pmatrix} = y_B \quad \text{and} \quad \begin{pmatrix} y' \\ y'' \\ y''' \end{pmatrix} = y.

&y_B = g \quad \Rightarrow \quad \begin{cases} 
    y' = 0, \\
    y'' \cdot R = g''_B, \\
    y''' = g'''_B - y'' \cdot S.
\end{cases}
\]

3.2. Properties of basic solutions

In [3], a relationship between the number of paths associated with split commodities, saturated arcs and split commodities is established. The proof of this proposition is based on the systematic study of the pivot types that can occur during the simplex method. The property is proved to be true at the beginning of the optimization process and to be maintained by pivoting. We shall give a direct proof of this proposition as well as a corollary which improves the characterization of the solutions produced by the primal partitioning algorithm.

**Proposition 1.** The number of basic path carrying split commodities \( |\mathcal{G} | \) is equal to the number of split commodities \( |\mathcal{K} | \) plus the number of saturated arcs \( |\mathcal{U} | \).

**Proof.** The basis is square : \(|\mathcal{U}| + |\mathcal{U}|' + |\mathcal{K}| + |\mathcal{K}| = |\mathcal{K}| + |\mathcal{K}| + |\mathcal{K}|\), unsplit commodities are carried by exactly one path, \(|\mathcal{K}| = |\mathcal{K}| \) and one basic slack variable appears for each unsaturated arc \(|\mathcal{U}| = |\mathcal{K}|\), thus \(|\mathcal{U}| = |\mathcal{K}| + |\mathcal{K}|\). \(\square\)

**Corollary 1.** The number of split commodities \( |\mathcal{K} | \) is less or equal to the number of saturated arcs \( |\mathcal{U} | \).

**Proof.** Proposition 1 gives \( |\mathcal{G}| = |\mathcal{U}| + |\mathcal{K}| \). A split commodity is carried by at least two basic path, \(|\mathcal{G}| \geq 2 \times |\mathcal{K}| \) and thus, \(|\mathcal{U}| \geq |\mathcal{K}|\). \(\square\)
This corollary gives a mathematical justification to an intuition well noted in [3]. When the arc capacities are very large compared to the size of an average commodity, few commodities will be split.

**Remark 1.** The basic idea of the partitioning can be extended to the node-arc formulation of the multicommodity network flow. A commodity will be called *split* if more than \(|X| - 1\) among their arc variables belong to the basis, *unsplit* otherwise. The Proposition 1 and Corollary 1 keep correct in that case and their proofs are very similar.

**Remark 2.** The introduction of jump constraint breaks the equivalence between the arc-chain and the node-arc formulation of a multicommodity network flow. In the arc-chain formulation, adapting the column generation permits to take into account the jump constraint, but no such adaptation is known for the node-arc formulation. We can only select from \(G\) a subgraph \(G^k\) (with diameter \(\delta\)?) and make the commodity \(k\) use only the arcs of \(G^k\), nevertheless a path with more than \(\delta\) arcs between \(s^k\) and \(t^k\) can exist in the graph \(G^k\) and can be used by the commodity \(k\).

4. Column generation with jump constraints

We will denote by a \(\delta\)-path, a path in the graph \(G\) with no more than \(\delta\) arcs. Let \(g_u\) denote the reduced cost of the slack variable \(e_u\) and \(\tilde{g}_j^k\) denote the reduced cost of the path variable \(x_j^k\). Let \(y_u^0\) denote the dual variable of the capacity constraint for the arc \(u\), and \(y_k\) denote the dual variable of the flow conservation constraint for the commodity \(k\).

\[
\tilde{g}_j^k = \sum_{u \in G_j^k} (c_u - y_u^0) - y_k.
\]

The pricing of the nonbasic path variables is made by computing the shortest \(\delta\)-paths between each pair of nodes the length of an arc \(u\) being \((c_u - y_u^0)\).

4.1. Matrix multiplication shortest \(\delta\)-path algorithm

We use a matrix multiplication shortest path algorithm similar to the one described in [6]. Suppose \(W = [w_{ij}]\) is the weight matrix of \(G\), we want to compute \(W^\delta\), in which \(w_{ij}^\delta\) is the length of the shortest \(\delta\)-path from \(i \in X\) to \(j \in X\). We define a sequence of matrices \(W^{(l)} = [w_{ij}^{(l)}]\) as follows:

\[
w_{ij}^{(1)} = w_{ij},
\]

\[
w_{ij}^{(l)} = \min \left( w_{ij}^{(l_1)}, \min_{k \in X} \left( w_{ik}^{(l_1)} + w_{kj}^{(l_2)} \right) \right) \quad \text{for } l = l_1 + l_2, l_1 \geq l_2,
\]
assuming that the weight of a nonexistent arc is $\infty$. The value of $w_{ij}^{(l)}$ is the length of the shortest $l$-path between $i$ and $j$.

Let $p$ be the maximal integer such that $2^p \leq \delta$. Computation of the matrices $W^{(1)}, W^{(2)}, \ldots, W^{(2^p)}$ requires $p$ matrix multiplications. With these matrices, the matrix $W^{(s)}$ can be obtained with at most $p$ additional multiplications. The complexity of this algorithm is $O(n^3 \log \delta)$.

This algorithm does not actually produce the shortest $\delta$-paths between each pair of nodes, it only produces their lengths. In the case of a classic shortest path algorithm, the paths themselves can be constructed from only one path matrix $\mathscr{P} = [p_{ij}]$, in which $p_{ij}$ is a node that belong to the shortest path from $i$ to $j$. A recursive procedure allows to construct this path as the concatenation of a shortest between $i$ and $p_{ij}$ and a shortest between $p_{ij}$ and $j$.

For the shortest $\delta$-path, it is not possible to use the same construction because it would lead to paths with more than $\delta$ arcs. Associated with each matrix $W^{(l)}$ involved in the calculation of $W^{(s)}$, we have to calculate a matrix $\mathscr{P}^{(l)} = p_{ij}^{(l)}$, in which $p_{ij}^{(l)}$ is a node that belongs to the shortest $l$-path from $i$ to $j$ otherwise.

5. Numerical experiment

The observed CPU times seem to be equivalent to the Farvolden's ones despite the additional time spent to find paths with no more than $\delta$ arcs. Farvolden's method had been compared on a set of randomly generated problems with some recent optimization methods (i.e. interior point method OB1 and general commercial simplex MINOS) and seems to be better on large problems. Table 1 contains the characteristics of the randomly generated problems which had been used to test our program and Table 2 presents the CPU time obtained with this data set on a SPARC 10 station (64 M RAM).

The difficulty of a problem depends not only on the size of the network, but also on the number of saturated arcs. It can be understood because the size of our working basis depends on the number of saturated arcs. Table 2 shows that the same problem can be

<table>
<thead>
<tr>
<th>Table 1</th>
<th>A set of randomly generated problems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Networks characteristics</strong></td>
<td><strong>Equivalent LP size</strong></td>
</tr>
<tr>
<td>Number of nodes</td>
<td>Number of arcs</td>
</tr>
<tr>
<td>R.20</td>
<td>20</td>
</tr>
<tr>
<td>R.30</td>
<td>32</td>
</tr>
<tr>
<td>R.50</td>
<td>49</td>
</tr>
<tr>
<td>R.60</td>
<td>59</td>
</tr>
<tr>
<td>R.70</td>
<td>68</td>
</tr>
<tr>
<td>R.80</td>
<td>80</td>
</tr>
<tr>
<td>R.100</td>
<td>99</td>
</tr>
</tbody>
</table>
Table 2

<table>
<thead>
<tr>
<th></th>
<th>CPU time (s)</th>
<th>No. of sat. arcs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Easy</td>
<td>Normal</td>
</tr>
<tr>
<td>R.20</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>R.30</td>
<td>3.4</td>
<td>5.1</td>
</tr>
<tr>
<td>R.50</td>
<td>55.3</td>
<td>75.2</td>
</tr>
<tr>
<td>R.60</td>
<td>92.5</td>
<td>155.0</td>
</tr>
<tr>
<td>R.70</td>
<td>292.9</td>
<td>533.9</td>
</tr>
<tr>
<td>R.80</td>
<td>203.1</td>
<td>268.0</td>
</tr>
<tr>
<td>R.100</td>
<td>522.0</td>
<td>695.9</td>
</tr>
</tbody>
</table>

less or more difficult when the capacity of their arcs are increased or decreased. Easy problems are problems for which we have increased the capacity of each arc (+10%) and hard problems are problems for which we have decreased the capacity of each arc (−10%). Most of the CPU time is spent performing matrix operations: updating the working basis inverse, and solving linear systems of equations. Improvements can be done by replacing the update of the working basis inverse by factorization routines for working basis of changing dimension [2] which would take advantage of the sparsity of $R$.

References