Determining the Non-Existence of Compatible OSPF Weights

Peter Broström pebro@mai.liu.se

Kaj Holmberg kahol@mai.liu.se

Department of Mathematics, Linköping university, SE-58183 LINKÖPING, Sweden

Abstract

Many telecommunication networks use the OSPF protocol (Open Shortest Path First) for deciding the routing of traffic. In such networks, each router sends traffic on all shortest paths to the destination. The links in the network are assigned weights to be used by the routers when calculating the shortest paths.

An interesting question is whether or not a set of desired routing patterns can be used in an OSPF network. We investigate this problem, and find new necessary conditions for the existence of weights making the desired patterns shortest. A polynomial algorithm that for most cases verifies the non-existence of compatible weights is presented. The algorithm also indicates which parts of the traffic patterns that are in conflict. Some computational tests of the algorithm are reported.

Key words: *Telecommunication networks, Internet Protocol, OSPF, routing, compatible weights.*

1 Introduction

In telecommunication networks using IP (Internet Protocol) and the routing protocol OSPF (Open Shortest Path First), traffic is routed on the shortest paths from each router to each destination. The routers calculate themselves the shortest paths to all possible destinations. The shortest path calculations are based on link weights set by the network operator. If the shortest path is not unique, traffic leaving a router is split equally on the leaving arcs that belong to a shortest path to the destination. This is called ECMP (the equal-cost multipath principle), and means that *all* shortest paths are used.

We will here study the problem of finding weights that give certain specified traffic patterns in a directed graph. We will also study the more important question of whether or not there exists a set of weights giving the desired traffic patterns. Traffic patterns are represented by the paths to be used. (If the paths are known, it is a simple matter to calculate the actual traffic.)

Similar problems have previously been treated as optimization problems in [4], [9], [2], and [5], and in a larger model for network design in [8]. Most of this work is done for undirected graphs and for single shortest paths. Restricting the traffic patterns to contain only single paths yields a simplification of the more general case we are treating. The usage of directed graphs (i.e. allowing different weights in different directions) is also an important generalization.

Apart from that, our main contribution is the way in which we verify and characterize the non-existence of weights. This is done by identifying combinations of traffic patterns that prohibit the existence of weights. This yields necessary conditions for the existence of weights, that are stronger than the conditions previously known. In addition, we present a polynomial method that explains why the specified patterns can not be used in an IP/OSPF network. This could be very useful in a planning process, since it identifies the parts of the patterns that need to be modified.

The paper is organized as follows. Section 2 is used for presenting the problem in detail, and for presenting a linear model which is used for finding appropriate weights. The LP-dual is formulated in section 3, and one type of unbounded solution to the LP-dual is classified in section 4. The solution method is presented in section 5, while possible modifications of SP-graphs are discussed in section 6. The method is exemplified in section 7, while computational results are presented in section 8. The last section concludes the paper and identifies parts that will be studied further.

2 Problem formulation

We consider a directed graph $G = (N, A)$ with a set of nodes N and a set of arcs A. A number of subsets of the arcs, $A_l \subseteq A$ for $l = 1, \ldots, m$, called SP-graphs (shortest path graphs), are given. We assume that each set A_l contains a spanning tree (ignoring the direction of the arcs) and that no set A_l contains a directed cycle. (These assumptions are motivated below.)

An SP-graph contains a number of paths, and these paths are the desired shortest paths. We wish to find weights w_{ij} for all arcs $(i, j) \in A$, so that the paths in A_l have minimal sum of the weights (i.e. are shortest). Thus if A_l contains a path from node s to node t, this path should have a minimal sum of weights. All paths from s to t not completely in A_l should have larger sums of weights. If A_l contains more than one path from node s to node t, all these paths should have (the same) minimal sum of weights.

Let us by $W(p)$ denote the sum of weights of all arcs in path p, i.e. $W(p) = \sum_{(i,j)\in p} w_{ij}$. If $p(s,t)$ and $q(s,t)$ are two paths from node s to node t, and $p(s,t) \subseteq A_l$ while $q(s,t) \not\subseteq A_l$, we require that $W(p(s,t)) < W(q(s,t))$. If both $p(s,t)$ and $q(s,t)$ lie in A_l, then we should have $W(p(s,t)) = W(q(s,t))$.

The case when there is at most one path in A_l between a pair of nodes is called the *simple path case*. More work has been done on the simple path case (see e.g. [4] and [5]) than on the more general case. However, in order to enable the use of *load balancing*, which is important in practice, an SP-graph must be allowed to contain several paths between a pair of nodes.

An SP-graph is meant to be the result of a router's shortest path calculations to all possible destinations, so usually an SP-graph is an out-graph with a single origin, spanning all nodes. Different SP-graphs then have different origins.

The weights w_{ij} must be integers greater or equal to 1. In principle there is an upper bound, namely the largest integer that can be represented by the router, for example 2^{16}, but this upper bound is considered to be redundant. In this context, we might mention that in [6] only weights up to 20 are considered when demonstrating the advantages of optimizing over the weights.

Definition 1 *The weights w are said to be compatible with A_l if $W(p(s,t)) = W(r(s,t))$ for any two paths $p(s,t) \subseteq A_l$ and $r(s,t) \subseteq A_l$, and $W(p(s,t)) < W(q(s,t))$ for any two paths $p(s,t) \subseteq A_l$ and $q(s,t) \not\subseteq A_l$.*

We will use the term **compatible weights** for a set of weights w that are compatible with each A_l for $l = 1, \ldots, m$. This means that compatible weights simultaneously give all the desired shortest paths for all SP-graphs. Our first objective is to *find a set of compatible weights.*

Note that the weights do not depend on l, so the existence of compatible weights imply some sort of similarities between the sets A_l. Given the sets A_l for $l = 1, \ldots, m$, there are two possibilities, either compatible weights exist or they don't. If compatible weights exist, the difference between two different sets of compatible weights is often unimportant. Thus we will mainly address the following question: *Does a set of compatible weights exist?*

We are not only interested in the yes/no answer to this question. If the answer is yes, we wish to find compatible weights. More importantly, if the answer is no, we wish to identify the parts of the SP-graphs that prohibit the existence of compatible weights. This will open possibilities of modifying SP-graphs so that compatible weights can be found.

If the weights w are given, the routers determine the shortest paths to each destination by solving shortest path problems. The SP-graphs have different origins/destinations, so we need to solve one shortest path problem for each SP-graph l.

Let $P^l(w)$ denote the shortest path problem obtained for the weights w and the origins/destinations given by SP-graph l. If we let b^l denote the right-hand-side of the constraints of $P^l(w)$, then the differences between SP-graphs are restricted to b^l. The LP-dual of the shortest path problem $P^l(w)$ is given below.

$$\max \sum_i b_i^l y_i \text{ s.t. } -y_i + y_j \leq w_{ij} \; \forall (i,j)$$

Here the objective function depends on l, while the feasible set does not. The dual constraints state $w_{ij} + y_i - y_j \geq 0 \; \forall (i,j)$, while the complementary slackness conditions tell us that only arcs with $w_{ij} + y_i - y_j = 0$ can be used by the shortest paths.

Since a shortest path problem is an LP-problem, an optimal solution is a basic solution, and thus forms a spanning tree. This means that there is a spanning tree of arcs with $w_{ij} + y_i - y_j = 0$. If the shortest paths are not unique, there are additional arcs with $w_{ij} + y_i - y_j = 0$. The arcs with $w_{ij} + y_i - y_j = 0$ will however never form a directed cycle, since the weights are positive.

A first necessary condition for the existence of compatible weights is that no SP-graph contains a conflict in itself. In other words, we assume that there exists a set of weights compatible with each SP-graph A_l. Disagreement only occurs as conflicts between two or more SP-graphs. We assume that each SP-graph has been obtained by solving a shortest path problem (for some given weights). This motivates the assumtions that each SP-graph spans all nodes and that no SP-graph contains a directed cycle.

Let us now construct a mathematical model for the problem of finding compatible weights. The problem is really only a feasibility problem, but let us add the goal of minimizing the sum of the weights. This is no important goal, but there is no point in letting the weights become unnecessarily large.

$$\begin{array}{lrcll} \min & \displaystyle\sum_{(i,j)\in A} w_{ij} & & & & \\ \text{s.t.} & w_{ij} + \pi_i^l - \pi_j^l & = & 0 \;\; \forall (i,j) \in A_l, l = 1, \ldots, m & (1.1) & \text{(P1)} \\ & w_{ij} + \pi_i^l - \pi_j^l & \geq & 1 \;\; \forall (i,j) \notin A_l, l = 1, \ldots, m & (1.2) & \\ & w_{ij} & \geq & 1 \;\; \forall (i,j) \in A & (1.3) & \end{array}$$

Constraints 1.1 ensure that the arcs in A_l are in minimal weight paths, while constraints 1.2 ensure that arcs outside of A_l are not in minimal weight paths. The coefficients in the

objective function are unimportant, and could be replaced by any nonnegative coefficients.

The solution of P1 will be rational, and by multiplying w and π by a constant of appropriate size, it can be made integral. It is easy to show that this retains feasibility in P1. (For details, see [3].)

Lemma 1 *If there exists a feasible solution to P1, there exists a feasible integer solution to P1.*

For a certain l each arc appears once in constraint set 1.1 or once in constraint set 1.2, since an arc is either in the SP-graph or not in it. Therefore any feasible solution to P1 will satisfy $w_{ij} + \pi_i^l - \pi_j^l \geq 0 \ \forall (i,j), \forall l$. Summing these constraints over any path $p(s,t)$ yields

$$W(p(s,t)) = \sum_{(i,j)\in p(s,t)} w_{ij} \geq \sum_{(i,j)\in p(s,t)} (\pi_j^l - \pi_i^l) = \pi_t^l - \pi_s^l,$$

so we have

$$W(p(s,t)) \geq \pi_t^l - \pi_s^l \qquad \text{for any path } p(s,t) \text{ and any } l. \tag{1.4}$$

The number of variables in P1 is $|A| + m|N|$, and the number of constraints is equal to $(m+1)|A|$, so the size of P1 is quite reasonable.

Theorem 1 *P1 has a feasible solution if and only if there exists a set of compatible weights.*

Proof: Consider a certain SP-graph A_l and two nodes s and t such that there exists a path $p(s,t)$ in A_l from node s to node t. Now assume that P1 has a feasible solution, w and π. The sum of weights for a path $p(s,t) \subseteq A_l$ is

$$W(p(s,t)) = \sum_{(i,j)\in p(s,t)} w_{ij} = \sum_{(i,j)\in p(s,t)} (\pi_j^l - \pi_i^l) = \pi_t^l - \pi_s^l,$$

due to constraints 1.1. Let d_{st} be the minimal sum of weights w on any path from node s to node t. Since $p(s,t)$ is one possible path from s to t, we have $W(p(s,t)) \geq d_{st}$, so $d_{st} \leq \pi_t^l - \pi_s^l$.

There must exist a path $q(s,t)$ from node s to node t with minimal sum of weights. For such a path we get $d_{st} = W(q(s,t)) \geq \pi_t^l - \pi_s^l$ due to 1.4, i.e. $d_{st} \geq \pi_t^l - \pi_s^l$.

Combining these results yields $d_{st} = \pi_t^l - \pi_s^l$. Above we noted that if $p(s,t) \subseteq A_l$ then $W(p(s,t)) = \pi_t^l - \pi_s^l$, so then $W(p(s,t)) = d_{st}$, which means that $p(s,t)$ is a minimal weight path. We have thus proved that any path in A_l is a minimal weight path.

Now consider another path $r(s,t) \not\subseteq A_l$ between the same two nodes. Constraints 1.1 and 1.2 yields the following.

$$W(r(s,t)) = \sum_{(i,j)\in r(s,t)} w_{ij} = \sum_{(i,j)\in r(s,t)\cap A_l} w_{ij} + \sum_{(i,j)\in r(s,t)\setminus A_l} w_{ij} \geq \sum_{(i,j)\in r(s,t)\cap A_l} (\pi_j^l - \pi_i^l) +$$
$$\sum_{(i,j)\in r(s,t)\setminus A_l} (\pi_j^l - \pi_i^l + 1) = \sum_{(i,j)\in r(s,t)} (\pi_j^l - \pi_i^l) + |r(s,t) \setminus A_l| = \pi_t^l - \pi_s^l + |r(s,t) \setminus A_l| =$$
$$d_{st} + |r(s,t) \setminus A_l| > d_{st}.$$

This shows that any path with $|r(s,t) \setminus A_l| > 0$ (i.e. at least one arc outside of A_l) has $W(r(s,t)) > d_{st}$, i.e. is not a minimal weight path. So for any l, we have shown that the paths in A_l are minimal weight paths, and paths not completely in A_l are not minimal weight paths. This verifies that the weights w are compatible with all SP-graphs, so there exists a compatible set of weights if P1 has a feasible solution.

Let us now assume that there exists a set of compatible weights, $w \geq 1$. Then we can solve the shortest path problems $P^l(w)$ for each l, and get the dual solutions y^l. Since the weights are compatible, all arcs in the SP-graphs will be included in minimal weight paths, while arcs outside of the SP-graphs will not.

This means that $w_{ij} + y^l_i - y^l_j = 0$ for each arc in A_l, while $w_{ij} + y^l_i - y^l_j > 0$ for each arc outside of A_l. If $w_{ij} + y^l_i - y^l_j < 1$ for some $(i,j) \notin A_l$, then w and π can be multiplied with a positive constant of appropriate size, in order to make the solution satisfy constraints 1.2. (See also lemma 1.) This verifies that there exists a feasible solution to P1 if there exists a compatible set of weights. □

Comments: It may be noted that that in the first part of the proof, no additional assumptions were made on the structure of the SP-graphs. In the second part, however, we note that shortest path problems yield spanning shortest path trees, so if the SP-graphs were not spanning, not connected or contained directed cycles, constraints 1.2 might not be satisfied.

In conclusion, if P1 has a feasible solution, no further assumptions on the SP-graphs are necessary. Verifying that P1 has a feasible solution if comptaible weights exist, however, requires that each SP-graph can be obtained by solving a shortest path problem.

It should be pointed out that nothing in the proof prohibits SP-graphs from containing several paths between a pair of nodes.

3 Using LP-duality

As mentioned above, our main interest lies in whether or not there exists compatible weights. This question can now be reformulated to whether or not P1 has a feasible solution. We start by formulating the LP-dual to P1.

Let γ^l_{ij} be the dual variables to constraint sets 1.1 and 1.2 (note that for any l, each arc appears once, either in constraint set 1.1 or in constraint set 1.2), and let δ_{ij} be the dual variables for constraint set 1.3. The LP-dual can be formulated as follows.

$$
\begin{aligned}
\max \quad & \sum_{l=1}^{m} \sum_{(i,j)\notin A_l} \gamma^l_{ij} + \sum_{(i,j)\in A} \delta_{ij} & & & & \\
\text{s.t.} \quad & \sum_{l=1}^{m} \gamma^l_{ij} + \delta_{ij} & = & 1 \quad \forall (i,j) \in A & (2.1) \\
& \sum_{j:(i,j)\in A} \gamma^l_{ij} - \sum_{j:(j,i)\in A} \gamma^l_{ji} & = & 0 \quad \forall i \in N, l = 1,\ldots,m & (2.2) \\
& \delta_{ij} & \geq & 0 \quad \forall (i,j) \in A & (2.3) \\
& \gamma^l_{ij} & \geq & 0 \quad \forall (i,j) \notin A_l, l = 1,\ldots,m & (2.4)
\end{aligned}
\qquad \text{(P2)}
$$

Note that γ^l_{ij} is free (not sign-restricted) for $(i,j) \in A_l, l = 1,\ldots,m$, and that these variables do not appear in the objective function.

Let us first eliminate δ. Constraint set 2.1 immediately gives $\delta_{ij} = 1 - \sum_{l=1}^{m} \gamma^l_{ij}$, and constraint set 2.3 yields $\sum_{l=1}^{m} \gamma^l_{ij} \leq 1$. Doing the substitution in the objective function yields

$$\sum_{l=1}^{m} \sum_{(i,j)\notin A_l} \gamma^l_{ij} + \sum_{(i,j)\in A} (1 - \sum_{l=1}^{m} \gamma^l_{ij}) = |A| - \sum_{l=1}^{m} \sum_{(i,j)\in A_l} \gamma^l_{ij}.$$

We now ignore the constant $|A|$ and change from maximization to minimization. (The actual objective function value is unimportant.) We have now simplified P2 to the following.

$$
\begin{aligned}
\min \quad & \sum_{l=1}^{m} \sum_{(i,j)\in A_l} \gamma_{ij}^l & & & & \\
\text{s.t.} \quad & \sum_{l=1}^{m} \gamma_{ij}^l & \leq & 1 & \forall (i,j) \in A & (3.1) \\
& \sum_{j:(i,j)\in A} \gamma_{ij}^l - \sum_{j:(j,i)\in A} \gamma_{ji}^l & = & 0 & \forall i \in N, l = 1, \ldots, m & (3.2) \\
& \gamma_{ij}^l & \geq & 0 & \forall (i,j) \notin A_l, l = 1, \ldots, m & (3.3)
\end{aligned}
\qquad \text{(P3)}
$$

Let v be the optimal objective function value of P3. The only difference between P3 and a standard *multicommodity network flow problem*, see for example [1], is that some variables are free (not nonnegative). Our goal at the moment is not to solve this problem computationally, but rather to investigate its properties.

Constraint set 3.2 states that the inflow to each node must equal the outflow, for each commodity. This means that we are looking for a circulating flow, as there are no sources or sinks. Constraint set 3.1 corresponds to capacity constraints, and all capacities are equal to one. (Note that these ones are the objective function coefficients for w in P1, and could, as mentioned, be other non-negative constants.)

We note that a feasible solution is obtained by setting $\gamma_{ij}^l = 0 \ \forall (i,j) \in A, \ \forall l$. This means that $\delta_{ij} = 1 \ \forall (i,j) \in A$, and a quick look at the complementary slackness conditions reveals that this corresponds to setting $w_{ij} = 1 \ \forall (i,j) \in A$. This is unlikely to be a feasible solution in P1, but since it is a feasible solution to P3, an upper bound to the optimal objective function value of P3 is zero. A better solution can be found if some of the (free) variables in the objective function can be decreased. Lemma 1 and LP-duality gives the following result (see [3] for details).

Lemma 2 *P1 has a feasible integer solution if and only of P3 has a bounded optimal solution.*

Thus there is no feasible integer solution to P1 if P3 has an unbounded solution, so we can study P3, in order to draw conclusions about the existence of compatible weights.

4 Unbounded multicommodity flow solutions

Let us now study unbounded solutions to P3. We start at a feasible solution, for example $\bar{\gamma} = 0$, and change it such that some variables go toward infinity.

Constraints 3.2 only allows circulating flow, so we must change the flow in cycles. Consider a *cycle* $C \subseteq A$, $C = F \cup B$, where F are the arcs used *forwards* (in their directions) and B are the arcs used *backwards* (against their directions). We change the flow of commodity l' in the cycle C by increasing γ_{ij}^l with the amount θ on forward arcs, and by decreasing γ_{ij}^l with the amount θ on backward arcs. To get an unbounded solution, we need to increase θ infinitely.

Constraint set 3.1 says that $\sum_{l=1}^{m} \gamma_{ij}^l \leq 1$, so if one variable in the left-hand-side is to be increased infinitely, another variable must be decreased infinitely. Specifically, if the flow of a commodity l' in arc (i,j) is increased by θ, then the flow of another commodity, l'', in that arc must be decreased by the same amount. This can be easily accommodated by using the same cycle C for commodity l'' as for commodity l', but doing the change in reversed direction. Thus we get the following change.

$$\gamma_{ij}^{l'} = \bar{\gamma}_{ij}^{l'} + \theta \quad \forall (i,j) \in F, \quad \gamma_{ij}^{l'} = \bar{\gamma}_{ij}^{l'} - \theta \quad \forall (i,j) \in B$$
$$\gamma_{ij}^{l''} = \bar{\gamma}_{ij}^{l''} - \theta \quad \forall (i,j) \in F, \quad \gamma_{ij}^{l''} = \bar{\gamma}_{ij}^{l''} + \theta \quad \forall (i,j) \in B$$

According to constraint set 3.3, $\gamma_{ij}^{l} \geq 0 \ \forall (i,j) \notin A_l \ \forall l$, some variables can not be decreased infinitely. The flows that are decreased are commodity l' in arcs B and commodity l'' in arcs F, so these variables must not appear in any non-negativity constraint. In other words, all $(i,j) \in B$ must also be included in $A_{l'}$ and all $(i,j) \in F$ must also be included in $A_{l''}$. This means that it is necessary that $B \subseteq A_{l'}$ and $F \subseteq A_{l''}$. This can also be written as $|B \cap A_{l'}| = |B|$ and $|F \cap A_{l''}| = |F|$. Furthermore, since $|B \cap A_{l'}| \leq |B|$ and $|F \cap A_{l''}| \leq |F|$, an equivalent statement is that $|B \cap A_{l'}| + |F \cap A_{l''}| = |B| + |F|$.

Lemma 3 *The flow in a cycle $C = F \cup B$ can only be increased infinitely if $B \subseteq A_{l'}$ and $F \subseteq A_{l''}$.*

We call such a cycle a *feasible* cycle, while if $|B \cap A_{l'}| + |F \cap A_{l''}| < |B| + |F|$, the cycle is called *infeasible*.

In order for P3 to have an unbounded solution, the objective function value must be decreased infinitely. Inserting the parameterized solution into the objective function yields

$$\begin{aligned} v &= \sum_{l=1}^{m} \sum_{(i,j) \in A_l} \gamma_{ij}^{l} = \sum_{l=1}^{m} \sum_{(i,j) \in A_l} \bar{\gamma}_{ij}^{l} + \sum_{(i,j) \in F \cap A_{l'}} \theta - \sum_{(i,j) \in B \cap A_{l'}} \theta - \sum_{(i,j) \in F \cap A_{l''}} \theta + \sum_{(i,j) \in B \cap A_{l''}} \theta \\ &= \Gamma + (|F \cap A_{l'}| - |B \cap A_{l'}| - |F \cap A_{l''}| + |B \cap A_{l''}|)\theta = \Gamma + \hat{r}^C \theta, \end{aligned}$$

where $\hat{r}^C = |F \cap A_{l'}| - |B \cap A_{l'}| - |F \cap A_{l''}| + |B \cap A_{l''}|$ is called the *reduced cost* for cycle C, and $\Gamma = \sum_{l=1}^{m} \sum_{(i,j) \in A_l} \bar{\gamma}_{ij}^{l}$, which is a constant.

In order for $v \to -\infty$ as $\theta \to \infty$, the reduced cost $\hat{r}^C$ must be negative. Now we remember that this unbounded solution is feasible only if $|B \cap A_{l'}| = |B|$ and $|F \cap A_{l''}| = |F|$, so the reduced cost becomes $\hat{r}^C = |F \cap A_{l'}| + |B \cap A_{l''}| - |F| - |B|$ and a negative reduced cost is obtained if $|F \cap A_{l'}| + |B \cap A_{l''}| < |F| + |B|$. Since $|F \cap A_{l'}|$ can never be larger than $|F|$ and $|B \cap A_{l''}|$ can never be larger than $|B|$, the only possibility for this inequality *not* to hold (i.e. $\hat{r}^C = 0$) is that $|F \cap A_{l'}| = |F|$ and $|B \cap A_{l''}| = |B|$. In other words, in order for the reduced cost to be negative, it is enough if there is one element in F not in $A_{l'}$ or one element in B not in $A_{l''}$.

We call arc (i,j) *eligible* if either $(i,j) \in F$ and $(i,j) \notin A_{l'}$ or $(i,j) \in B$ and $(i,j) \notin A_{l''}$. Moreover, a cycle with $|F \cap A_{l'}| + |B \cap A_{l''}| < |F| + |B|$ is called an *improving* cycle, while if $|F \cap A_{l'}| + |B \cap A_{l''}| = |F| + |B|$, the cycle is called *non-improving*.

Lemma 4 *A feasible cycle $C = F \cup B$ indicates an unbounded solution of P3 only if $|F \cap A_{l'}| + |B \cap A_{l''}| < |F| + |B|$, i.e. if there is at least one eligible arc (an arc in F not in $A_{l'}$ or in B not in $A_{l''}$).*

Summing up these conclusions, we wish to find a cycle $C = F \cup B$, and two commodities l' and l'' such that $B \subseteq A_{l'}$ and $F \subseteq A_{l''}$ while $|F \cap A_{l'}| + |B \cap A_{l''}| < |F| + |B|$, which means that the cycle is both feasible and improving.

Definition 2 *A cycle $C = F \cup B$ is called <u>valid</u> if there exist two indices l' and l'' such that $|B \cap A_{l'}| = |B|$, $|F \cap A_{l''}| = |F|$ and $|F \cap A_{l'}| + |B \cap A_{l''}| < |F| + |B|$. Equivalently, for a valid cycle, $B \subseteq A_{l'}$ and $F \subseteq A_{l''}$, while $B \not\subseteq A_{l''}$ and/or $F \not\subseteq A_{l'}$.*

Theorem 2 *If there exists a valid cycle, then there exists no compatible set of weights.*

Theorem 2 can also be proved by direct arguments in P1, not using LP-duality. By walking around the cycle, one can sum up all constraints encountered, and from this draw the conclusion. The details of this are given in [3].

A previously known necessary condition for the existence of compatible weights is that if two desirered paths use the same two nodes in the same order, then the paths between the two nodes must be identical. Paths that satisfy this are called *sub-optimal*. It is in [3] shown that if two SP-graphs contain paths that are not sub-optimal, then a valid cycle exists. Furthermore, in section 7 we give an example with sub-optimal paths where a valid cycle exists. Thus, the absence of valid cycles is a stronger necessary condition for the existence of compatible weights than sub-optimality.

One can easily show the following (for details, see [3]).

Lemma 5 *A valid cycle must contain at least three nodes and three arcs.*

If all SP-graphs are trees, there does not exist any cycle within an SP-graph, even if we disregard the arc directions. Since a feasible cycle has $B \subseteq A_{l'}$, we can draw the conclusion that $F \not\subseteq A_{l'}$ for any feasible cycle. Also, since $F \subseteq A_{l''}$, we know that $B \not\subseteq A_{l''}$. This immediately tells us that any feasible cycle is improving, i.e. valid.

Theorem 3 *If the SP-graphs $A_{l'}$ and $A_{l''}$ are trees, then any feasible cycle (i.e. with $B \subseteq A_{l'}$ and $F \subseteq A_{l''}$) is also valid.*

5 A method for finding valid cycles

Let us now consider practical ways of finding valid cycles. We wish to find a valid cycle $C = F \cup B$ and two indices l' and l'' verifying its validity, i.e. such that $|B \cap A_{l'}| = |B|$, $|F \cap A_{l''}| = |F|$ and $|F \cap A_{l'}| + |B \cap A_{l''}| < |F| + |B|$.

We enumerate all pairs of commodities, i.e. try to find a valid cycle for each $l' = 1, \ldots, m$ and $l'' = 1, \ldots, m$. There are $(m-1)^2$ possibilities, but as soon as we find a valid cycle, we stop. So assume now that l' and l'' are given.

First we note that $A_{l'} \cup A_{l''}$ must cover the cycle, so arcs not belonging to any of these sets are discarded, i.e. removed from the graph. Then, since we require $B \subseteq A_{l'}$ and $F \subseteq A_{l''}$, we label each arc in $A_{l'}$ with B, and each arc in $A_{l''}$ with F. This is called the *labeling phase*.

After this, all remaining arcs are labeled at least once. Arcs labeled only with B must belong to the set B, if they are included in the cycle, and arcs labeled only with F must belong to the set F, if they are included in the cycle. Arcs labeled with B or F are eligible, while arcs labeled with both B and F are not eligible.

The next step is to remove parts of the remaining graph that can not be a part of a valid cycle. This is called the *reduction phase*. In this stage we must obey the arc labels, so that an arc labeled F can only be used forwards, and an arc labeled B can only be used backwards, and an arc labeled both with B and F can be used in both directions. For example, an arc "entering" a node i can be an arc ending in node i with label F, an arc starting in node i with label B, or an arc labeled with both F and B (regardless of original direction).

The reduction phase aims at ensuring that all remaining nodes have at least one entering arc and at least one leaving arc (different from the entering arc).

- For any node with only one adjacent arc: Discard the node and the arc.

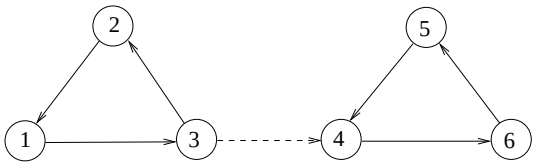

Figure 1: A graph with an arc not included in any cycle.

- If there are no arcs entering (leaving) a node: Discard the node and all adjacent arcs.
- If there is only one arc entering (leaving) a node, and this arc is labeled with both B and F: Keep the label that enables entering (leaving) the node, and remove the other label.

Furthermore we note that there must be at least three nodes and three arcs in a valid cycle, see lemma 5, so any graph smaller than that may be immediately eliminated.

- For any connected graph component with less than three nodes or less than three arcs: Discard all nodes and all arcs in the component.

These graph reductions should be repeated until no more changes occur. We need to investigate each node at least once, checking all its adjacent arcs. As soon as a change is made in the graph, all affected nodes have to be checked again. As the changes are either removing an arc label or removing a node, at most $|N| + |A|$ changes can be made. If we keep a list of the in-degree and out-degree of every node, this list can easily be updated when changes are made, especially if the change only affects one arc. An simple estimation of the complexity of the reduction phase is $O(|N|^3)$.

Lemma 6 *If the graph is completely eliminated by the reduction phase, there exists no valid cycle with the two SP-graphs considered.*

If the graph is not completely eliminated, a graph with at least three nodes remains, and it contains at least one feasible cycle. According to lemma 4, a feasible cycle is valid if it contains an eligible arc, i.e. one arc in F outside of $A_{l'}$, or one arc in B outside of $A_{l''}$. If there are no eligible arcs in the graph, there is no valid cycle in the graph, and we are finished with this commodity pair.

Let us start with an eligible arc. A cycle is found simply by traversing nodes, using a leaving arc that is different from the entering. After at most $|N|$ steps, we will return to a node previously visited, and have thus found a cycle.

It is not certain that our starting arc is included in the cycle we find. Actually, it is not certain that there exists a cycle including our starting arc. See the graph in figure 1, which might be the result of the reduction phase. Note that the graph has two cycles, and that regardless of which arc we start with, one of them will be found. However, if we start with arc (3,4), and hope to find a cycle with it in, we will fail. We will instead find the cycle 4-6-5.

In general there are two possibilities. Either there exists a cycle containing an eligible arc, or there does not exist one. In the first case, we wish to find the cycle. In the second case, we wish to verify that there is no such cycle, and conclude that there exists no valid cycle for this pair of commodities. We try to make the cycle as large as possible, since that might increase the probability that our starting arc will be included in the cycle. If

there is more than one outgoing arc from a node, we avoid if possible arcs going to already visited nodes.

As soon as we find a cycle including our starting arc or another eligible arc, we have succeeded in finding a valid cycle, and can terminate the procedure. If there is no eligible arc in the cycle, we have found a cycle with $F \subseteq A_{l'}$ and $B \subseteq A_{l''}$, i.e. a cycle with reduced cost equal to zero. Such cycles are not valid, and should if possible be removed from the graph.

If the cycle is "isolated", in the sense that the total out-degree from the cycle is equal to zero (see cycle 4-6-5 in figure 1) or the total in-degree into the cycle is equal to zero (see cycle 1-3-2 in figure 1), it can be eliminated. The reason is that the arcs in the cycle can not be a part of another cycle, since either we can not leave the cycle once we are in it, or we can not enter it from nodes outside the cycle. This reasoning can be extended to any subgraph as follows.

Definition 3 *A subgraph containing no eligible arcs and with either total in-degree equal to zero or total out-degree equal to zero, is called an isolated subgraph.*

Lemma 7 *No node in an isolated subgraph can be a part of a valid cycle.*

All nodes in an isolated subgraph (and all adjacent arcs) can thus be discarded. We then return to the reduction phase, since this could enable additional reductions in the graph.

Let us now assume that a graph with at least three nodes and at least one eligible arc remains. We also assume that no valid cycle has been found, no more reduction of the graph is possible and that the heuristic cycle search fails to find a cycle.

Suppose that we choose an eligible arc (i, j), say a forward arc not in $A_{l'}$. A cycle including this arc consists of arc (i, j) and a path from node j to node i. Therefore we search for a path from j to i (or a path from i to j if (i, j) is a backward arc). We set cost zero on all eligible arcs and one on all others. Arcs labeled with both F and B are duplicated, one in each direction.

Then we find the shortest path from node j to i with a standard shortest path method, for example Dijkstra's method, which has the complexity $O(|N|^2)$. The result is either a path from j to i, or a proof (a cut separating j from i) that there exists none. If we have found a path, a cycle is formed by adding arc (i, j), and we have succeeded in finding a valid cycle. If there is no path, we know that there exists no cycle including arc (i, j). Then this arc can be removed from the graph, and we return to the reduction phase, which may yield new results in the absence of arc (i, j).

Actually, using Dijkstra's method, we will label all nodes that are reachable from node j, and the cut will indicate a set of nodes, D, that has no leaving arc. If there is no eligible arc in the subgraph spanned by D, we have found an isolated subgraph which can be eliminated.

This way we will either find a valid cycle or eliminate the whole graph. Concerning the complexity, at least one arc will be removed in each main iteration, after which the reduction phase is redone. A crude estimation of the complexity of the method is $O(m^2|A||N|^3)$, which is $O(|A||N|^5)$ if the number of SP-graphs equals the number of nodes, and it is certainly not more than $O(|N|^7)$. This can probably be decreased, but our conclusion is that the method is polynomial. Furthermore, this complexity has little to do with the practical performance of the method.

Let us summarize the Valid Cycle (VC) method in a more algorithmic fashion.

1. **Choice of SP-graphs:** If all pairs of SP-graphs have been compared, go to 12. Otherwise choose two SP-graphs l' and l'' not previously compared.

2. **Labeling phase:** Label each arc in $A_{l'}$ with B and each arc in $A_{l''}$ with F. Arcs labeled with only B or only F are marked eligible.

3. **Reduction phase:** Repeat until no more changes:
   Remove nodes with only one adjacent arc.
   Remove nodes with no entering (leaving) arcs.
   Remove an arc label if the other label gives the only entering (leaving) arc.
   Remove all isolated components with less then three nodes or arcs.

4. **Elimination check:** If all arcs are eliminated, go to 11 .

5. **Eligible arc:** Search the remaining graph for an eligible arc. If no eligible arc exist, go to 11. Otherwise, let (i, j) be the eligible arc found, and L its label.

6. **Heuristic cycle search:** Find a cycle by heuristic: Start with arc (i, j) (in the proper direction) and traverse nodes, using adjacent arcs. Never use the same arc twice. Stop when a node is visited a second time: A cycle is found.

7. **Evaluation of cycle:** If the found cycle contains an eligible arc, go to 13. If the found cycle has total in-degree or out-degree equal to zero, go to 10.

8. **Shortest path cycle search:** Set arc cost equal to zero for eligible arcs and equal to one for the other arcs. Find shortest path from node j to node i if L is F, or from node i to node j if L is B. If a path exists, add arc (i, j) to form a cycle, and go to 13.

9. **No path:** Remove arc (i, j). If the reachable subgraph contains some eligible arc, go to 3.

10. **Isolated subgraph:** An isolated subgraph is found. Eliminate all nodes in the subgraph and all adjacent arcs. Go to 3.

11. **Graph eliminated:** No valid cycle found. Go to 1.

12. **No valid cycle found:** No valid cycle exists. Terminate the method. (Try to find compatible weights.)

13. **Valid cycle found:** A valid cycle is found. Terminate the method. No compatible weights exist.

Comments: Feasible directions of the arcs are given by the labels. When a node is removed in the reduction phase, all adjacent arcs are also removed. When searching for cycles, the direction is given by the label of the eligible starting arc. A cycle found by the shortest path method always contains an eligible arc, so this does not need to be checked.

For the special case when all SP-graphs are trees, theorem 3 tells us that any feasible cycle is also improving, and thus valid. A feasible cycle is always found in step 6, and we know that it is valid, so we will go directly to step 13. In this case, steps 7, 8, 9 and 10 will never be used, and can be removed from the algorithm.

Theorem 4 *After a finite number of steps, algorithm VC will terminate, either with a valid cycle, or with the whole graph eliminated, in which case there exists no valid cycle.*

The algorithm VC can be used to check if a number of SP-graphs agree. See section 6 for a discussion about the possibilities of changing SP-graphs to make them agree. It can also be used in different iterative procedures for determining which SP-graphs to use, out of a larger number. One might also consider to use it within an advanced *Constraint Programming* method, where algorithm VC is an implemented constraint.

6 Modifications of SP-graphs

If no compatible set of weights exist for a certain set of SP-graphs, these SP-graphs can not be realized in an IP network using OSPF. In this context there is something "wrong" with this set of SP-graphs.

If our task is to determine the values of the weights, and we are forced to use the weights found, we will get traffic patterns that are different from what is desired. However, as P1 is infeasible, solving it will not yield any useful information about how to set the weights.

P1 could be made feasible by changing some of the indata. The simplest possibility is to remove some SP-graph, which means that some paths are no longer considered to be desired. In such a case, our method is directly useful, since it indicates two conflicting SP-graphs, l' and l''. Removing one of these will remove that particular conflict. Obviously this may have to be repeated, since our method stopped when it found the first conflicting pair of SP-graphs.

Another possibility is to modify an SP-graph. Again it is useful that our method indicates which SP-graphs to consider. We even know which set of arcs in the SP-graphs to consider. In such a situation, we get a cycle $C = F \cup B$ and two indices l' and l'', such that $B \subseteq A_{l'}$ and $F \subseteq A_{l''}$ (since it is feasible). We would also like to have $B \subseteq A_{l''}$ and $F \subseteq A_{l'}$, but there is at least one arc not fulfilling this, since the cycle is improving. The arcs making it improving are given by the sets $\hat{F} = \{(i,j) : (i,j) \in F, (i,j) \notin A_{l'}\}$ and $\hat{B} = \{(i,j) : (i,j) \in B, (i,j) \notin A_{l''}\}$.

In order to remove a valid cycle, we can either make it non-improving or make it infeasible (or both). The following actions are possible to take.

1. Add all arcs in $\hat{F}$ to $A_{l'}$ and and all arcs in $\hat{B}$ to $A_{l''}$. This makes the cycle non-improving.

2. Remove one arc in $B \cap A_{l'}$ from $A_{l'}$ or one arc in $F \cap A_{l''}$ from $A_{l''}$ in such way that the SP-graph remains connected. This makes the cycle infeasible.

3. Replace arc(s) in $A_{l'}$ or $A_{l''}$ to make the cycle infeasible and/or non-improving.

Unfortunately, these changes might create new conflicts between the SP-graphs. Adding arcs might make a previously infeasible cycle feasible, and if it is improving, it will become valid. Removing arcs might make a previously non-improving cycle improving, and if it is feasible, it will be valid. The development of a better method for changing SP-graphs is a topic for future research.

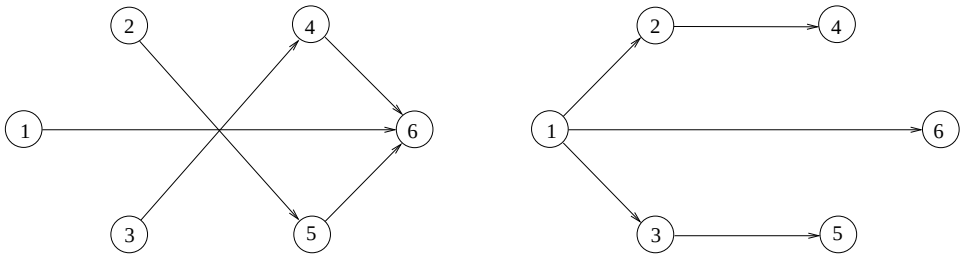

Figure 2: The in-tree A_1 and out-tree A_2.

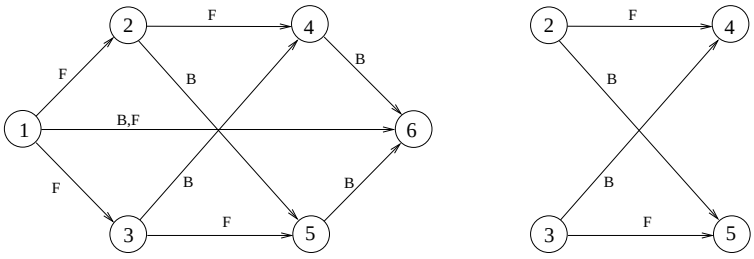

Figure 3: The graph after the labeling phase (left) and the graph after the reduction phase (right).

7 A small example

Let us now study an example with two SP-graphs (based on an example given in [7]). SP-graph A_1 is an in-tree entering node 6, while SP-graph A_2 is an out-tree leaving node 1, as shown in figure 2. This pair of SP-graphs is suboptimal, since the only node pair connected in both A_1 and A_2 is connected by an identical path (node pair (1,6)).

We label all arcs in A_1 with B and all arcs in A_2 with F. The left graph in figure 3 shows the situation after the labeling phase.

In the graph reduction phase, arcs (1,2) and (1,3) are labeled F so if it should be possible to enter node 1, arc (1,6) must be labeled B. Therefore we remove label F from arc (1,6). However, now node 6 can not be entered, so node 6 and all adjacent arcs are discarded. Since arc (1,6) was removed, it is not possible to enter node 1, so it is discarded, together with all adjacent arcs. In the right part of figure 3, we show the graph remaining after the reduction phase.

All arcs are now eligible, since both arcs in F lie outside A_1 and both arcs in B lie outside A_2. The heuristic cycle search will find the cycle 2 - 4 - 3 - 5 - 2 and we conclude that there does not exist any compatible weights for this pair of SP-graphs.

If we wish to modify the SP-graphs so that compatible weights exist, we could add arcs (2,4) and (3,5) to A_1 and (2,5) and (3,4) to A_2. There are now two paths in A_1 between nodes 2 and 6 (and nodes 3 and 6), so this introduces splitting of the traffic. The same happens in A_2 between nodes 1 and 4, and between 1 and 5.

We might instead consider removing an arc from A_1 or A_2. However, as A_1 and A_2 are trees, this is not possible since the corresponding SP-graph will fall apart. It is possible to replace an arc in one SP-graph, for example replace (2,5) by (2,4) in A_1. This change will remove the valid cycle. Furthermore no new conflicts will appear, and there exists compatible weights after the change.

Table 1: Computational results.

$\|N\|$	P	N_V	N_W	N_O	T_T	T_I
10	41	18	22	1	16	0.4
15	42	35	7	0	34	0.8
20	40	33	7	0	97	2.4
30	40	33	7	0	536	13.4

8 Implementation and computational tests

The VC-algorithm has been implemented in Tcl/Tk within the framework of the graphical package VINEOPT (Visual Network Optimization) (www.vineopt.com). Tcl/Tk is a scripting language, and the code has been translated to C with the package MKTCLAPP. Nevertheless this implementation is probably much slower than a proper implementation in C.

A number of test problems have been generated in the following manner. We start with four networks with 10, 15, 20 and 30 nodes and generate weights such that splitting is probable. Shortest path trees to each node are calculated, and SP-graphs are constructed by including all arcs with reduced cost equal to zero. This means that the number of SP-graphs is equal to the number of nodes, and that all SP-graphs are in-graphs spanning all nodes.

The resulting SP-graphs obviously have compatible weights. A goal of the problem generation is that it should not be known in advance if compatible weights exist, so some modifications are made. The first modification is to include one arc with reduced cost equal to one in a randomly chosen SP-graph. The second modification is to add an arc to a randomly chosen SP-graph, such that the depth of the ending node of the arc is greater than the depth of the starting node. It can be shown that neither of these modifications lead to directed cycles in the SP-graphs.

One solution approach is to first try to find compatible weights by solving P1 with an LP-code, and if it fails to find a feasible solution, analyze the situation further with the VC-algorithm. Another approach is to first run the VC-algorithm, which will either prove that no compatible weights exist, or indicate that compatible weights might exist. In the latter case, we try to find the weights by solving P1.

The first approach is probably more efficient, since all pairs of SP-graphs must be considered by the VC-algorithm. However, since the topic of this paper is to analyze a set of SP-graphs, we have chosen to use the second approach. The largest part of the solution time is therefore spent on comparing the SP-graphs, since P1 is only solved when no valid cycle is found. We use the code LPSOLVE for solving P1.

In table 1 the computational results are summarized. P denotes the number of instances in the group, N_V is the number of instances with valid cycles, N_W is the number of instances with compatible weights, N_O is the number of instances without valid cycles and compatible weights (which means that the unbounded solution of P3 is of a more complicated type). T_T is the total time for all of the problems, and T_I is the average time for each instance, both in seconds. The computer used is a 2.4 GHz PC running Linux.

Table 1 shows that the solutions times are reasonable. Another important result is that only for *one* of the 163 different instances solved, compatible weights do not exist even though no valid cycle is found. This indicates that valid cycles seem to capture most cases where compatible weights do not exist. We can therefore draw the conclusion that valid

cycles give a practical and useful characterization of instances for which no compatible weights exist.

9 Conclusions

We have presented a new and useful way of finding instances of shortest paths graphs that together prohibits the existence of compatible weights for IP networks using OSPF. The characterization is based on so called valid cycles. A polynomial method for finding valid cycles is proposed. Computational results confirm that the solution method does not take exceedingly long time, and that it seems to capture most cases where compatible weights do not exist.

Future research will consist of including this method into a method for finding the optimal design of an OSPF network. We will also investigate and develop practical solution methods for the more complicated cases where neither compatible weights nor valid cycles exist.

Acknowledgment: This work has partly been financed by the Swedish Research Council.

Bibliography

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B., *Network Flows. Theory, Algorithms and Applications*, Prentice Hall 1993.

[2] Ben-Ameur, W. and Gourdin, E., "Internet routing and related topology issues", *SIAM Journal on Discrete Mathematics* 17 (2003) 18–49.

[3] Broström, P. and Holmberg, K., "Determining the non-existence of a compatible OSPF metric", Research Report LiTH-MAT-R-2004-06, Department of Mathematics, Linköping Institute of Technology, Sweden 2004.

[4] Farago, A., Szentesi, A., and Szviatovszki, B., "Allocation of administrative weights in PNNI", in: *Proceedings of the Networks'98, Sorrento*, pages 621–625, 1998.

[5] Farago, A., Szentesi, A., and Szviatovszki, B., "Inverse optimization in high-speed networks", *Discrete Applied Mathematics* 129 (2003) 83–98.

[6] Fortz, B. and Thorup, M., "Internet traffic engineering by optimizing OSPF weights", in: *Proceedings of IEEE INFOCOM '00* volume 2, pages 519–528, 2000.

[7] Gourdin, E., "Optimizing internet networks", *ORMS Today* 28/2 (2001) 46–49.

[8] Holmberg, K. and Yuan, D., "Optimization of Internet Protocol network design and routing", *Networks* 43(1) (2004) 39–53.

[9] Pioro, M., Szentesi, A., Harmantos, J., Jüttner, A., Gajowniczek, P., and Kozdrowski, S., "On open shortest path first related network optimization problems", *Performance Evaluation* 48 (2002) 201–223.