Algorithms for Asymmetrically Weighted Pair of Disjoint Paths in Survivable Networks

Péter Laborczi[1] János Tapolcai[1,2,3] Pin-Han Ho[3]
Tibor Cinkler[1] András Recski[2] Hussein T. Mouftah[3]

[1] Department of Telecommunications and Telematics,
[2] Department of Computer Sciences and Information Theory,
Budapest University of Technology and Economics,
Pázmány Péter sétány 1/D, H-1117 Budapest,Hungary
e-mail: {laborczi,cinkler}@ttt-atm.ttt.bme.hu
{tapolcai,recski}@cs.bme.hu
[3] Department of Electrical and Computer Engineering,
Queen's University, Kingston Ontario, Canada K7L 3N6
e-mail: hoph@ee.queensu.ca, mouftah@ece.queensu.ca

Abstract

To find node-disjoint path-pairs is a critical issue of survivable networks. One of the paths have frequently higher priority in which case the problem is called *asymmetrically weighted pair of disjoint paths*. In this novel approach a special, weighted objective function should be minimized. We prove the problem to be **NP**-complete and several heuristics are proposed based on k-shortest paths searching, Suurballe's algorithm, Integer Linear Programming (ILP), Linear Relaxation (LR), and Minimum Cost Network Flow (MCNF) algorithm to achieve best performance in polynomial time. The simulation results show that the heuristic based on MCNF yields the best performance in terms of cost and running time. The optimal solution is found in about 99.9% of all cases while a near-optimal solution in the remaining few cases.

Keywords: Protection and Restoration Algorithms, Spare Capacity Allocation, Transport Networks

1 Introduction

Modern telecommunication networks carry tremendous amount of traffic that risks a serious data loss when a failure occurs (e.g., a fiber cut or a node failure). The most common way to protect a path is to find a protection route over which the protected data flow can be switched when a failure occurs in the network element the path passes through. The working and protection path has to be node-disjoint. The protection path can be computed after its corresponding working path is determined, however this may lead to a non-optimal solution. On the contrary, a working path and a protection path can be found simultaneously to yield the globally optimal node-disjoint path-pair.

This problem is also known as node-disjoint diverse routing problem and usually solved intuitively by the Two-Step-Approach algorithm [1]. This first finds the shortest path, then finds the shortest path in the same graph with the edges and nodes of the first path being temporarily erased. Although this method is straightforward and simple, it may fail to find any disjoint path-pair since erasing the first path can isolate the source node from the destination. This can happen even if the network is highly connected[1]. This approach may perform well if the task is to reduce the cost of working path without any attention paid to the protection path. We will call this approach 2D (two-step-Dijkstra).

To find two disjoint paths with minimal total cost, Suurballe's algorithm (SUURB) [2] [3] can be used. This is a modification of Dijkstra's algorithm and guarantees to find the disjoint path-pair if it exists.

However, in many cases a network operator's aim is not only to reduce the total cost but to optimize the cost sum with weighting one of the paths (e.g., the working path). In this paper, we focus on the optimization of the sum $\alpha cost(P_1) + cost(P_2)$, where $cost(P_1)$ and $cost(P_2)$ are the cost of the working path and of the protection path, respectively.

Parameter α is the weight of the working path, which is determined by the protection strategy. In other words the consumption of network resources by the working path is α times more important than that of the protection path. For example, if a shared protection scheme (e.g., $1:N$ or $M:N$) is adopted, α could be 10..100 depending on how many paths share a link and on the amount of best-effort traffic flowing in the network. On the other hand, α could be small (e.g., $1 \sim 5$) for dedicated protection (e.g., $1+1$) since there is no difference in the consumption of network resources between a working path and a protection path.

In this paper we will present several solution alternatives. After formulating the problem we will prove that the problem is **NP**-hard. In Section 4 we analyze an enhancement of the widely used routing algorithms of dynamic routing problems. In Section 5 the problem is formulated as an Integer Linear Program (ILP). Then the proposed methods are discussed that are based on Linear Relaxation and Minimal Cost Flow. In Section

[1]One can easily construct a graph in that the shortest path between s and d takes all nodes.

7 methods are given to solve the split-flow problem that arise in relaxations. Finally, the numerical results are discussed.

2 Problem Formulation

The input of the problem is a directed or undirected graph $G(V, E)$ representing the network topology where the set of nodes denoted by V and the set of arcs denoted by E with a positive cost for each link (denoted as c_e of edge $e \in E$).

The demand has a source (s), a destination (t), and a factor (or weight) α for weighting the working path. The task is to find edge- or vertex-disjoint working (P_1) and protection (P_2) paths from a given source to destination such that $f(s,t,\alpha) = \alpha\text{cost}(P_1) + \text{cost}(P_2)$ should be minimal. In this paper we are focusing on vertex-disjoint paths and we mean this by the term *disjoint path.* Please note that here the routing of **one** demand is solved. The described methods should be called as a subroutine if a demand is allocated.

As mentioned if $\alpha = 1$ then the problem can be solved by Suurballe's algorithm [2] with complexity of $O(n^2 \log n)$.

3 Complexity Study

In this paper we are going to present a series of approximation algorithms to solve the asymmetrically weighted disjoint path problem. The optimization is proved to be **NP**-complete, which will be verified in the following paragraph.

The result given by Suurballe's algorithm is optimal only in the case of $\alpha = 1$. On the other hand, it can approximate the derivation of $f(s,t,\alpha) = \alpha\text{cost}(P_1) + \text{cost}(P_2)$ no worse than the optimal cost factored by $\frac{\alpha+1}{2}$ [5], where P_1 and P_2 are two disjoint paths between a given source s and destination t and α is a given constant, i.e.,

$$f_{app}(s,t,\alpha) \leq \frac{\alpha+1}{2} \cdot f_{opt}(s,t,\alpha)$$

where $f_{app}(s,t,\alpha)$ is the result given by the algorithm that approximates (which is Suurballe in this case), $f_{opt}(s,t,\alpha)$ is the optimal solution. The above relationship is defined as an approximation with a factor of $\frac{\alpha+1}{2}$.

Theorem 1. Suurballe's algorithm [3],[2] approximates the cost of the optimal asymmetrically weighted path-pair with a factor of $\frac{\alpha+1}{2}$.

Proof. We should prove that

$$\alpha x_1' + x_2' \leq \frac{\alpha+1}{2} \cdot (\alpha x_1 + x_2)$$

where x_1 and x_2 are the optimal solution of our problem, and x_1' and x_2' are the result of Suurballe's algorithm $(x_1' < x_2')$.

$$\begin{aligned} &\tfrac{\alpha+1}{2} \cdot (\alpha x_1 + x_2) && \text{as } \alpha > 1 \\ &> \tfrac{\alpha+1}{2} \cdot (x_1 + x_2) && \text{as } x_1' + x_2' \text{ optimal} \\ &\geq \tfrac{\alpha+1}{2} \cdot (x_1' + x_2') && \text{as } x_1' + \alpha x_2' \geq \alpha x_1' + x_2' \\ &\geq \tfrac{\alpha x_1' + \alpha x_2' + x_1' + x_2'}{2} \geq \alpha x_1' + x_2' \end{aligned}$$

□

The following theorem is to claim that there is no any polynomial-time algorithm that can approximate the derivation of an asymmetrically weighted path-pair within a factor of $\frac{\alpha+1}{2}$, unless **P**=**NP**.

Theorem 2. [4],[5],[6] On directed graphs it is **NP**-hard to approximate the cost of the optimal asymmetrically weighted path-pair within a factor of $\frac{\alpha+1}{2}$, i.e, if an algorithm achieves an approximated cost $f_{app}(s,t,\alpha)$, in which

$$f_{app}(s,t,\alpha) \leq C \cdot f_{opt}(s,t,\alpha)$$

$$1 \leq C < \frac{\alpha+1}{2}$$

then the algorithm is not polynomial, unless **P**=**NP**.

Proof. We can reduce our problem to the following one: Given a directed graph $G_0 = (V, E)$ and four distinct vertices s_1, t_1, s_2, t_2; decide whether there are two node disjoint directed paths in G_0 so that one of them leads from s_1 to t_1 and the other from s_2 to t_2. This was proved to be **NP**-complete by Hopcroft at al., see [7].

Add two vertices s and t to $G_0 = (V, E)$ and define the network $G_0' = (V', E')$ by

$$\begin{aligned} V' &= V \cup \{s, t\} \\ E' &= E \cup \{s \to s_1, s \to s_2, t_1 \to t, t_2 \to t\} \end{aligned}$$

where all arcs have cost 1 except arcs $s \to s_1$ and $t_1 \to t$ which have huge cost denoted by K,where

$$K = \frac{C \cdot (n\alpha - \alpha + 1) + 2(\alpha + 1)}{\alpha + 1 - 2C} + 1 \qquad (1)$$

where n is the number of nodes in G_0. Note that K is well defined only if $C < \frac{\alpha+1}{2}$. It is easy to verify that $K > n$ as $C \geq 1$ and $n \geq 1$ and $\alpha > 1$.

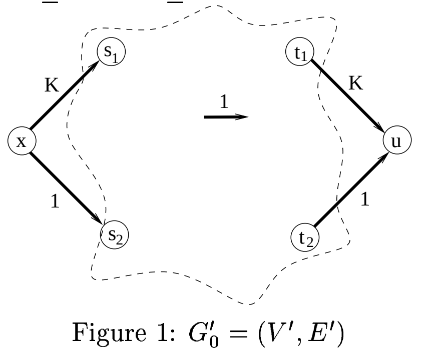

Figure 1: $G_0' = (V', E')$

G_0' can be constructed in polynomial time, and all edge-lengths are bounded by a polynomial in α and n. Now we show that there exist two directed disjoint paths between vertices s_1, t_1, s_2, t_2 if and only if the approximated cost $f_{app}(x, \alpha) \leq C \cdot (n\alpha - \alpha + 2K + 1)$.

$\Leftarrow$

If there exist two directed disjoint paths between vertices s_1, t_1, s_2, t_2, then as $K > n$ the cost of $s \to s_1 \rightsquigarrow t_2 \to t$ and $s \to s_2 \rightsquigarrow t_1 \to t$ is

$f_{app} \leq C \cdot \alpha cost(P_w) + cost(P_p)\}$

$\leq C \cdot \big(\alpha(2+n-3)+(2K+1)\big) \leq C \cdot (n\alpha - \alpha + 2K + 1)\Box$

$\Rightarrow$

The proof is indirect. Assume that the cost is $f_{app} \leq C \cdot \min\{\alpha c(P_w) + c(P_p)\} \leq C \cdot (n\alpha + \alpha + 2K + 1)$ but there is no pair of directed disjoint paths between vertices s_1, t_1, s_2, t_2. Then the approximated cost is

$C \cdot (n\alpha + \alpha + 2K + 1) \geq f_{app} \geq f_{opt}$

$= \min\{\alpha cost(P_w) + cost(P_p)\} \geq C \cdot \big(\alpha(K+2) + (K+2)\big)$

Therefore

$$C \cdot (n\alpha + \alpha + 1) + 2(\alpha + 1) \geq K \cdot (\alpha + 1 - 2C)$$

hence

$$K \leq \frac{C \cdot (n\alpha + \alpha + 1) + 2(\alpha + 1)}{\alpha + 1 - 2C}$$

a contradiction to (1). $\Box$

Nevertheless, in the the proof of **NP**-completeness Hopcroft at al. used a very special graph that can show the "NP" complexity of the problem. In practical implementation, the network topology can be much different, and often undirected graphs are used as the model of the network. Note that the problem in the undirected case is an open question (with a fixed α). Therefore we may hope to be able to derive a very good approximation in polynomial time even if the approach can not be guaranteed to outperform Suurballe's algorithm. The simulation results showed that the approximation algorithms could achieve a high availability and quality in finding disjoint path-pairs in the practical network topologies we engaged (see Section 8 for simulation results).

4 An Enhancement to the Two-Step-Approach

We first propose a heuristic that is based on the approach of finding the k-shortest path. With this heuristic, not only the shortest path is examined but also the k-shortest paths, where $k = 1, 2, ...$ We implement the Sub-Optimal Path-Solver (SOPS) algorithm, for finding the k-shortest paths between an S-D pair in the network.

The SOPS algorithm is based on an iterative replacement, in which a segment of path is erased from the network, and replaced by a path segment generated by applying the shortest path first algorithm between the two cutting nodes. The original path is a seed path that yields a new path that is called a generated path. We will refer to this subroutine as *"Replacement"*.

The algorithm is briefly stated as follows.

STEP 1: Find a complete set of paths $Min\_Path(S, D, 0)$, where $Min\_Path(S, D, k)$ is the set of k-shortest paths and is in the k^{th} stage.

STEP 2: Using each path in $Min\_Path(S, D, 0)$ as a seed path, running *Replacement*.

STEP 3: If a generated segment is shorter than the replaced segment, erase the generated segment from the network and run *Replacement* again.

STEP 4: The generated paths are included into $Min\_Path(S, D, 1)$, $Min\_Path(S, D, 2)$,...,or $Min\_path(S, D, K)$, where N is the offset

STEP 5: Iteratively using $Min\_Path(S, D, 0)$, $min\_Path(S, D, 1)$.....$Min\_Path(S, D, K-1)$ to get $Min\_Path(S, D, K)$, as shown in Figure 2

$$\begin{array}{llllll}
0\to1 & & & & & \\
0\to2 & 1\to2 & & & & \\
0\to3 & 1\to3 & 2\to3 & & & \\
0\to4 & 1\to4 & 2\to4 & 3\to4 & & \\
\vdots & \vdots & \vdots & \vdots & & \\
0\to N & 1\to N & 2\to N & 3\to N & \cdots\cdots & (N\text{-}1)\to N
\end{array}$$

Figure 2: *The derivation of the $Min\_Path(S, D, N)$*

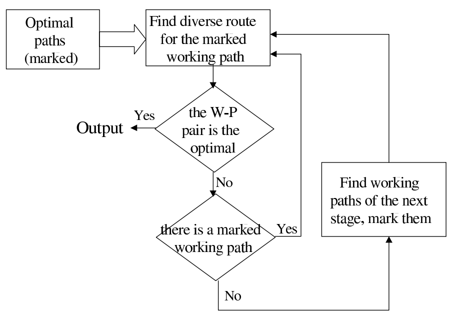
Optimal paths (marked)
Find diverse route for the marked working path
the W-P pair is the optimal
Yes
Output
No
there is a marked working path
Yes
No
Find working paths of the next stage, mark them

Figure 3: *Flow chart for determining node-disjoint path pair*

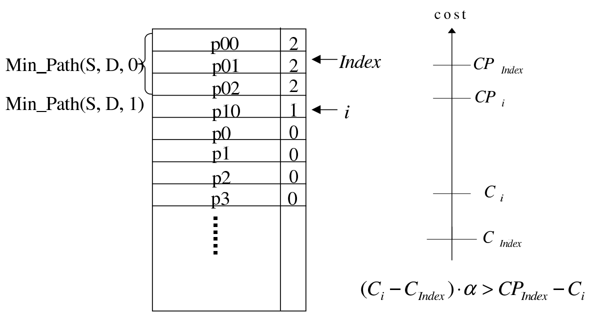

Figure 4: *Checking the optimality of the iteration*

Figure 4 shows the flow chart for finding a node-disjoint working-protection path-pair. Figure 2 illustrates how to utilize the SOPS algorithm and to determine if a path-pair is the optimal one. A path is marked with "2" if it has worked as a seed path to yield more expensive path. A path is marked with "1" if it has been "in order". The order of the paths marked with "0" are still unknown. For example, the path p00, p01, p02 and p10 are "in order", and it is impossible to find a path with a cost in between any two of the paths. However, there might be a path with a cost larger than the p10 and less than p0 if the p10 is used as a seed path.

In the table of Figure 4, the "*Index*" keeps the optimal working path P_{Index} with cost C_{Index}, and another index "i" keeps the working path P_i that is being tried to derive a disjoint path, which has a cost of C_i. The optimality of $\alpha \cdot C_{Index} + CP_{Index}$ can be examined by the inequality $(C_i - C_{Index}) \cdot \alpha > CP_{Index} - C_i$, where CP_{Index} and CP_i are the costs of the protection paths of P_{Index} and P_i. It is easy to see that if the inequality holds, any C'_i that is larger than C_i can at most yield a sum of $(\alpha + 1) \cdot C'_i$, which is larger than $CP_{Index} + \alpha \cdot C_{Index}$; i.e., $\alpha \cdot C_{Index} + CP_{Index}$ is always larger than $\alpha \cdot C_i + CP_i$ in this situation.

The heuristic proposed above can solve the asymmetrically weighted optimal disjoint path-pair problem without regarding the value of α, however, it has to be based on an efficient approach to finding k-shortest path, which is usually notoriously with the exponential computation complexity. Therefore, more efficient algorithms have to be developed to cope with this problem.

5 Integer Linear Programming (ILP)

The problem can be solved optimally by an Integer Linear Program (ILP).

Objective:

$$\text{minimize} \sum_{e \in E} (\alpha x_e + y_e) c_e \qquad (2)$$

Subject to constraints:

$$\sum_{j=1}^{N} x_{ij} - \sum_{k=1}^{N} x_{ki} = \begin{cases} 0 \text{ if } i \neq \text{source} \wedge i \neq \text{sink} \\ 1 \text{ if } i = \text{source} \\ -1 \text{ if } i = \text{sink} \end{cases}$$
$$\sum_{j=1}^{N} y_{ij} - \sum_{k=1}^{N} y_{ki} = \begin{cases} 0 \text{ if } i \neq \text{source} \wedge i \neq \text{sink} \\ 1 \text{ if } i = \text{source} \\ -1 \text{ if } i = \text{sink} \end{cases}$$
$$\text{for all nodes } i \qquad (3)$$

$$x_e \in \{0,1\}, y_e \in \{0,1\}, \text{ for all edges } e \in E \qquad (4)$$

$$x_e + y_e \leq 1 \text{ for all edges } e \in E \qquad (5)$$

The binary flow indicators x_e (y_e) take value 1 if the working (protection) path uses edge e or 0 if not. α is the weight factor for the primary path and c_e expresses the cost of using edge e.

According to constraint (5) which ensures edge-disjoint paths, the working and backup paths may not use the same edge. For making the working and backup paths node-disjoint the following constraint should be used instead of (5): $\sum_{j=1}^{N}(x_{ij} + y_{ij}) \leq 1$ for all nodes i except the source and the sink.

6 Relaxations

As shown in 3 even a "good" relaxation of the problem is **NP**-hard, therefore, derivation of the optimal solution for large networks (e.g., with 100 or more nodes) needs an unacceptable amount of computation time. It is crucial especially in the case of dynamic route computation. To solve a linear program (LP) or a flow problem is much less complex task than to solve an integer program. With the following relaxation process we got 1.5 to 100 times faster running time, depending on the size and density of the network. Both LP and ILP can be solved by many software packages. We did experiments with CPLEX and LP-SOLVE. Besides, we adopted two types of relaxation: *Linear Programming Relaxation* (LPR) and *Single Flow Relaxation* (SFR) in this work.

Linear Programming Relaxation (LPR). LPR is a relaxation of the above ILP formulation by relaxing the constraint (4), i.e., the condition that the variables should be integers. That is, instead of (4) the following constraints are to be applied: $0 \leq x_e \leq 1$ and $0 \leq y_e \leq 1$, for all edges $e \in E$, which however allows fractional flows and may yield useless result (as shown in Figures 7 and 8). Although relaxation is of polynomial complexity in contrast to the integer problem but the split-flow problem that may occur needs to be further manipulated. We will discuss this in details in Section 7 and show that in most of the situations the optimal solution still can be found.

Single Flow Relaxation (SFR). The idea of SFR is based on Minimal Cost Flows (MCF) with the following conditions: all capacity constraints are set to α and a flow of size $1 + \alpha$ is to be found. SFR renders split-flow even more often as shown in Figures 5 and 6.

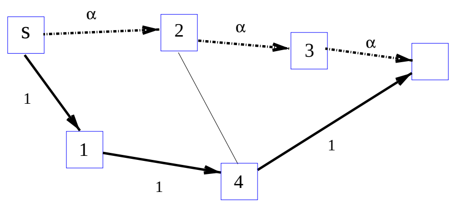

Figure 5: *Example when SFR gives optimal solution. (Dashed arrows represent the working path, with flow size of alpha and solid lines the protection path, with flow size of* 1*).*

There exists a large number of fast Minimal Cost Network Flow (MCNF) implementations, e.g., that of

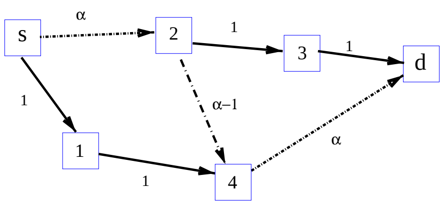

Figure 6: *Example when SFR yields a split flow. (Numbers on the arcs represent the flow values.)*

Goldberg [8]. In addition the Network optimizer of CPLEX can solve the MCNF problem efficiently (in $O(n^2 m \log n)$ time). It can also be solved by any other available LP solver, such as LP-SOLVE.

7 Methods for Solving the Split-Flow Problem

The result of our experiments shows that by the assistance of LR and SFR an optimal solution can be derived in around 80-90% of all cases (in which the relaxation yields a two-path solution). In the other cases (10-20%) the relaxation does not yield a two-path solution but a split flow. However, if the split-flow problem occurs, the result of the relaxation may be useless. In the following paragraphs we propose two approaches, Tiny Additional Costs (TAC) and α-Shifting, for solving the split-flow puzzle, and deriving an optimal or a near-optimal solution for the diverse-routing problem. Before presenting our approaches, two types of split-flow problems are defined and described as below.

Reparable split. Because of the essential characteristics of the LPR and MCF solvers, the flows in the network may be split if there are two or more optimal paths between source and destination. As an example shown in Figure 7, every link cost is one, and the total amount of flow from s to d is one as well. The numbers specified on the edges represent the percentage of flow. In this case, both the working (solid arrows) and the protection (dashed arrows) paths are split although they could take single paths with the same minimum cost as well. Although this situation occurs in a practical network with relatively small probability [2], it may ruin the whole computation process once it exists. To solve this type of split-flow problem, Tiny Additional Costs (TAC) is proposed and discussed in Section 7.1).

Unreparable split. Figure 8 shows another type of split-flow problem (links s-1 and 12-d have high cost, all other links have low cost). In this case the split-flow would have minimum cost, which cannot be repaired in polynomial time[3]. In other words, if a polynomial computation time has to be guaranteed, it may not be able to find an optimal solution. To cope with this

[2]having more than one path of equal cost between source and destination

[3]unless **P**=**NP** the problem is **NP** complete (see Theorem 2) while LP can be solved in polynomial time [9]

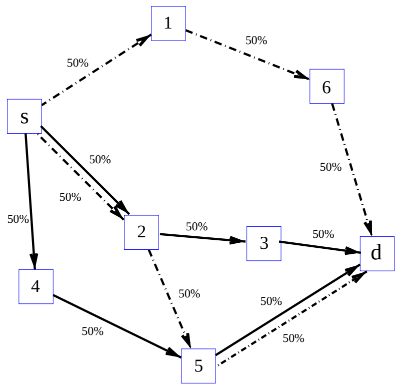

Figure 7: *Example for reparable split of LPR. The numbers next to the arc represent the size of the flow.*

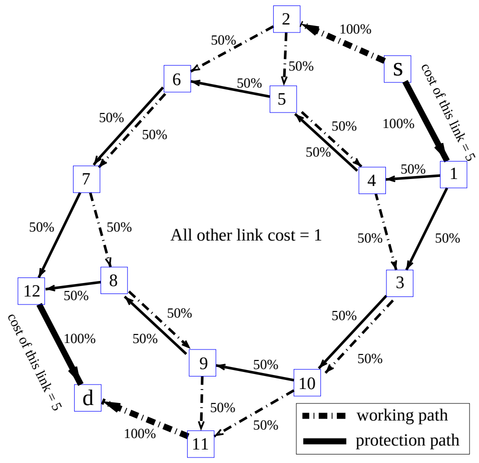

Figure 8: *Example for unreparable split. If* $\alpha = 5$ *the total cost is* $7\alpha + 15 = 50$*, while the disjoint path's cost would be* $9\alpha + 9 = 54$

situation, we propose a heuristic, α-shifting, to find a possibly good repair in polynomial time.

7.1 Tiny Additional Costs (TAC)

TAC is proposed to solve the reparable split problem, in which a random number ϵ_e is added to each edge e, where $0 \leq \epsilon_e \leq 1/\alpha n$, n is the number of nodes in the network. Based on the fact that the original cost of links are all integers, therefore the total cost of the path-pair is integer as well. On the other hand, the number of links used by the path-pair is smaller than n, that is why the deviation of the total cost (e.g. $\alpha\text{cost}(P_1)+\text{cost}(P_2)$) is less than one, since $\sum_{e \in E_1} \epsilon_e \alpha n < 1$, where E_1 is the set of edges used by P_1 or P_2. Therefore, the result of the relaxation is not influenced by applying TAC, while the probability that more than one optimal path exist between source and destination can be significantly reduced.

Two methods based on TAC are tested to solve the reparable split problem. First, LPT is when TAC is performed before the relaxation. If the flow problem still occurs in the relaxation, it fails. Second, LPT+ is an enhancement of LPT when TAC is performed repeatedly on the original network until either the split-flow problem is solved (or an optimal solution is derived) or the time of iterations reaches max_{LPT+}, what is user defined.

7.2 α-Shifting

Recall that ILP can solve the problem and guarantee the optimal solution. However, due to **NP**-completeness of the ILP, large networks (e.g. number of nodes > 200) cannot be handled in this way. To reduce the complexity, we propose a heuristic, α-shifting, to yield a near-optimal solution within a polynomial computation time. The basic idea of α-shifting is to search α' that is closest to α, with which the relaxation process works with unsplit flows or at least reparable flows, i.e., a sub-optimal path pair is yielded.

In order to understand the method, we will analyze the cost of the path-pair plotted against α, denoted by $t_{(s,d)}(\alpha)$. It can be proved that $t_{(s,d)}(\alpha)$ is continuos, monotone increasing, piecewise linear and concave. The piecewise linearity of the function comes from the fact that each path-pair has a cost of $\alpha\text{cost}(P_1)+\text{cost}(P_2)$ what is linear against α. Also possible to prove that the solution of any LP relaxation plotted against α has the same properties (denoted with $t^{relax}_{(s,d)}(\alpha)$). Obviously, $t^{relax}_{(s,d)}(\alpha) \geq t_{(s,d)}(\alpha)$ and equal if the LPR or SFR gives integer solutions (see Figure 9). It can be proved that $t^{relax}_{(s,d)}(1) = t_{(s,d)}(1)$. Figure 9 shows an example where the relaxation process suffers from the unreparable split-flow problem when $\alpha = 4$. With the proposed α-shifting scheme we are going to find α' closest to α such that a sub-optimal path pair can be derived. A binary search is suggested here to find α', in which $\alpha' = 6$ will be tried first, followed by $\alpha' = 5$, and then $\alpha' = 2$ and $\alpha' = 3$. The heuristic works as follows:

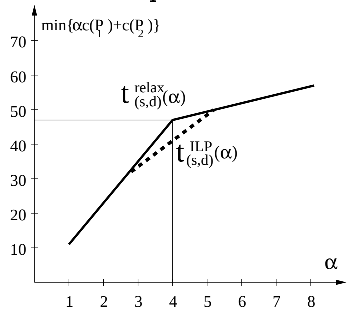

Figure 9: *An example of the cost function plotted against α*

STEP 1: Run relaxation SFR or LPR with 2α.

STEP 2: If successful then find α' closest to α with binary search ($\alpha < \alpha' < 2\alpha$)

STEP 3: If not successful find α' closest to α with binary search ($1 < \alpha' < \alpha$)

With this method in the worst case α' goes to 1 where the problem is degraded and gives the same result as Suurballe's algorithm.

7.3 Methods Summary

The process of solving asymmetrically weighted disjoint path-pair is summarized by the flow chart shown in Figure 10. The approaches (0) $\rightarrow$ (1) $\rightarrow$ (2) $\rightarrow$ (3) $\rightarrow$ (4) can guarantee to derive an optimal solution (if there is any) at an expense of being non-polynomial during (4). On the other hand, the polynomial computation time can be achieved without a guarantee of optimality via the approach (0) $\rightarrow$ (1) $\rightarrow$ (2) $\rightarrow$ (3).

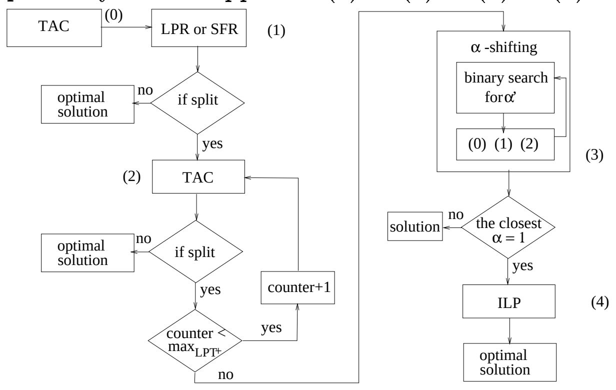

Figure 10: *Flowchart for solving the asymmetrically weighted optimal disjoint path-pair problem*

8 Numerical results

Figure 11 shows a typical function of cost against α in the case when there are different optimal path pairs at different values of α. On the vertical coordinate the value 1 represents the optimum cost (the cost was normalized). With the growth of α 2D gives better and better results compared to the optimum, i.e., it is a decreasing function. Note that for 2D the existence of the solution is not guaranteed (see also Table 5, row *2D success*). On the other hand Suurballe's algorithm (SUURB) gives optimal solution at $\alpha = 1$ and moves away from it if α increases. In this example in the interval $4 < \alpha < 12$ both 2D and SUURB yields more than 1.5 times worse solution compared to the optimal one. It can be seen that our method is always very close to the optimum. However, one has to mention that in a number of cases not only our method but also 2D and SUURB give the optimal solution. That is the reason to get smaller differences between the average costs of all node-pairs as for the "problematic" node pairs in Tables 1, 2, 3.

Four methods are compared according to 3 criteria: running time, success, and cost. Simmulations have been carried out between each pair of nodes and average calculated for running time and cost. Success means the percentage of node-pairs for that the method gives any solution. The methods are: (1) ILP that yields the optimal solution, (2) Suurballe (SUURB) that minimizes the total cost of the two paths, (3) 2D that minimizes the cost of the first paths, (4) LPR, and (5) SFR: our proposed methods. The tests were carried

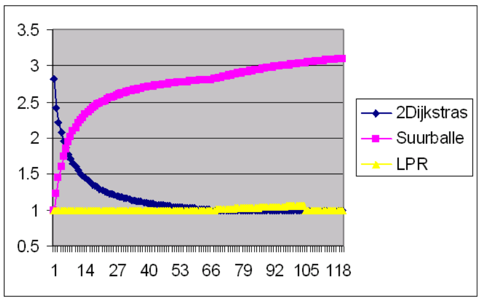

Figure 11: *A typical function of cost against alpha normalized with the value of the optimal solution, when the optimal solution differs from the Suurballe's one.*

out on an 500 MHz Intel PentiumIII based computer with 64 Mbytes of memory. ILOG's CPLEX 6.5 and LP-SOLVE has been used for ILP tasks that are widely used for solving linear programs.

Tables 1 and 2 show numerical results for a dense 30-node and a sparse 35-node network. The running times show the average computational time for one demand in milliseconds. The success value is always 100% except in case of the 2D method since it does not always yield solution. Average normalized cost of all demands are shown for $\alpha = 1, 5, 100$, and average normalized cost for the "problematic" path-pairs. "Problematic" path-pair means that ILP and SUURB does not yield the same quality of result for $\alpha > 1$. Table 3 shows numerical results for 10 demands of a 400-node network.

In smaller networks (n30, n35) SUURB and 2D have shorter running time but worse quality than the three other methods. In case of n35 2D did not find solution in 7% of the cases. LPR did not split any flows. LPR gives very similar running time compared to ILP. In case of n30 and n35 SFR gives always the optimal solution. The reason for that is that in networks of such size SFR runs ILP if the flow splits. The running time of SFR is a fraction of ILP while it gives result very near to optimum.

The running time of the algorithm for a network of 1000 nodes and 1900 edges is as follows: 19.2s for LPR, and 7.1 for SFR. The quality of the results are similar to the other networks.

Table 4 shows the percentage of optimally routed demands in eight networks: four of them are sparser (N5, N15, N25, N35) and seven are denser (N30A, N30B, N30C, N50, N60, N100). The network names refer to the number of nodes. Simulations have been carried out between all pairs of nodes and the percentage of optimal result is shown in case of SFR, LPR, LPR+SFR, LPT, LPT+ In case of LPR+SFR after LPR SFR is applied if and only if LPR did not yield optimal solution. LPR+SFR yields optimal solution in all sparse networks, and about 99% of all demands in dense networks, and nearly optimal in the remaining 1%. LPT+ yields optimal solution for almost all of the demands in all networks.

In Table 5 five methods are compared according to the increase of runtime on different networks.

9 Conclusion

We have proposed methods for finding asymmetrically weighted minimum cost disjoint paths. It finds solution in polynomial time that is in 99.9% of all cases the optimal one. In the remaining cases (1) either a near-optimal solution is delivered, (2) or the optimal solution can be found in longer time. The obtained results can be used for configuring survivable DWDM, SDH, ATM, MPLS, and other networks. The proposed methods deliver results even in large networks (1000 nodes or more).

10 Acknowledgement

This work has been supported by HSNLab (http://hsnlab.ttt.bme.hu) and ETIK (http://www.etik.hu) and by grants OTKA 29772 and 30122 of the Hungarian National Science Fund, and by the NATO Collaborative Linkage Grant PST.CLG.976383. It is also partially supported by an NSERC of Canada Research Grant 1056 and the Computer Networks lab at Queen's University, Kingston, Canada.

References

[1] Ramesh Bhandari. *Survivable networks : algorithms for diverse routing.* The Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston, 1999.

[2] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[3] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[4] Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. Finding disjoint paths with different path-costs: complexity and algorithms. *Networks*, 22(7):653–667, 1992.

[5] Z. Király. Private communication, 2001.

[6] D. Marx. Private communication, 2001.

[7] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoret. Comput. Sci.*, 10(2):111–121, 1980.

[8] Andrew V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, 22(1):1–29, 1997.

[9] L.G. Khachian. A polynomial time algorithm for linear programming. *Doklady Akad. Nauk SSSR 244*, 4:1093–1096, 1979.

n30	Running time [ms]		Success	Cost of all path-pairs [%]			Cost "problematic" [%]		
methods	LP-SOLVE	CPLEX	[%]	$\alpha = 1$	$\alpha = 5$	$\alpha = 100$	$\alpha = 1$	$\alpha = 5$	$\alpha = 100$
Suurb	4.02		100	100	102.60	105.28	100	111.5	123.0
2D	4.00		100	102.80	100.74	100.04	109.8	102.5	100.1
ILP	46.25	35.08	100	100	100	100	100	100	100
LPR	47.82	32.07	100	100	100	100	100	100	100
SFR	13.79	10.85	100	100	100	100	100	100	100

Table 1: *Nummerical results of a (denser) network with 30 nodes and 63 links*

n35	Running time [ms]		Success	Cost of all path-pairs [%]			Cost "problematic" [%]		
methods	LP-SOLVE	CPLEX	[%]	$\alpha = 1$	$\alpha = 5$	$\alpha = 100$	$\alpha = 1$	$\alpha = 5$	$\alpha = 100$
Suurb	3.33		100	100	101.42	103.41	100	108.1	121.0
2D	3.11		93	-	-	-	-	-	-
ILP	25.48	33.75	100	100	100	100	100	100	100
LPR	25.51	28.62	100	100	100	100	100	100	100
SFR	9.24	10.28	100	100	100	100	100	100	100

Table 2: *Numerical results of a (sparser) network with 35 nodes and 51 links*

n400	Running time [ms]		Success	Cost [%]		
methods	LP-SOLVE	CPLEX	[%]	$\alpha = 1$	$\alpha = 5$	$\alpha = 100$
Suurb	721		100	100	105.41	106.14
2D	720		100	107.94	100.91	100.01
ILP	2873	1030	100	100	100	100
LPR	2855	1053	100	100	100	100
SFR	787	167	100	100	100	100

Table 3: *Numerical results of 10 relevant demands in a network with 400 nodes and 1378 links*

Relax	N5	N15	N25	N35	N30A	N30B	N30C	N50	N60	N100
SFR	100	100	88	81.7	95.5	91.4	75.9	87.0	93.9	92.5
LPR	100	100	100	100	98.6	99.7	99.5	97.2	98.7	98.6
LPR+SFR	100	100	100	100	99.3	99.7	99.5	98.8	99.1	99.0
LPT	100	100	100	100	100	99.7	99.5	99.9	99.9	99.9
LPT+	100	100	100	100	100	99.7	99.5	100	100	100

Table 4: *Percentage of optimal routed demands in the 8 examined networks*

Network	No. Node	16	22	30	79	100	400
	No. of edge	27	43	63	108	191	1382
	Avg. length of working path	2.43	2.86	3.27	6.52	7.13	11.16
	Avg. nodal degree	3.37	3.91	4.20	2.73	3.82	6.91
	2D success	93.03	96.20	100	90.36	91.41	100
SOPS	Normalized time per path-pair	1	1.11	2.44	13.49	82.4	-
	Cost [%]	100	100	100	100	100	-
ILP	Normalized time per path-pair	1	2.26	5.32	60.36	46.7	3465
	Cost [%]	100	100	100	100	100	100
LPR	Normalized time per path-pair	1	2.27	5.52	64.80	47.67	3384
	Cost [%]	100	100	100	100	100	100
SFR	Normalized time per path-pair	1	1.10	1.34	5.6	5.25	480
	Cost [%]	100	100	100	100	100	100

Table 5: *The increase of runtime on different networks ($\alpha = 5$)*