MASTER'S THESIS

EVALUATING THE NETWORK SURVIVABILITY ISSUE OF K-BEST PATHS THROUGH GRAPH THEORETIC TECHNIQUES

Marinos Stylianou

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

June 2005

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

EVALUATING THE NETWORK SURVIVABILITY ISSUE OF K-BEST PATHS THROUGH GRAPH THEORETIC TECHNIQUES

Marinos Stylianou

Supervising Professor
Dr. Andreas Pitsillides

This thesis is submitted in partial fulfillment of the requirements for the Degree of Masters in Science at Computer Science Department of the University of Cyprus

June 2005

Acknowledgements

I would like to thank my professor Dr. Andreas Pitsillides for his close cooperation we had during the authorship of my thesis and also for the invaluable help he offered. His advise and direction contributed in the biggest possible way, for the successful completion of this work. Also to my colleagues and especially Josephine Antoniou for her help

I owe one big thank to Katerina for her support tolerance but also understanding that she showed.

Finally I would want to thank the God that allowed me to complete with success this work.

Summary

In this thesis graph theoretic techniques are adopted for addressing the network survivability issue of disjoint paths selection. The evaluation was conducted after the implementation of a solver that produces a solution of the problem after successive application of two algorithms on any given topology, the algorithm of Louca et al [19] and Castanon's [8]. The first algorithm transforms any networks into a trellis graph and the second exploits the special structure of the trellis graph and solves for the k-best paths using the minimum cost network flow (MCNF) algorithm. The transformation and evaluation of the K-best paths solution is illustrated for a number of topologies through the graphical user interface adapted from [37]. It is also contrasted with the k-successive approximation methods, which cannot guarantee the selection of the K-best paths, due to the successive removal of shortest paths at each iteration. Furthermore, the performance of the algorithm and its time complexity are investigated and also compared with Surballe's Disjoint Pair Algorithm [31]. Even though the trellis transformations algorithm can find all possible disjoint paths in the vast majority of cases, pathological situations where the algorithm may fail is also identified in the thesis, analysed, and a solution is provided and evaluated.

Table of Contents

Chapter 1

Introduction

Today's large-scale, highly distributed, networked systems improve the efficiency and effectiveness of organizations by permitting whole new levels of organizational integration. However, such integration is accompanied by elevated risks of intrusion and compromise. Incorporating survivability capabilities into an organization's systems can mitigate these risks.

Survivability builds on related fields of study (e.g., security, fault tolerance, safety, reliability, reuse, performance, verification, and testing) and introduces new concepts and principles. Survivability focuses on preserving essential services, even when systems are penetrated and compromised [43].

1.1 The Definition of Survivability

We define survivability as the capability of a system to fulfil its mission, in a timely manner, in the presence of attacks, failures, or accidents. The term system is used in the broadest possible sense, including networks and large-scale systems of systems.

The term mission refers to a set of very high-level requirements or goals. Missions are not limited to military settings, because any successful organization or project must have a vision of its objectives whether expressed implicitly or as a formal mission statement. Judgments as to whether or not a mission has been successfully fulfilled are typically made in the context of external conditions that may affect achievement of that mission. For example, imagine that a financial system shuts down for 12 hours during a period of widespread power outages caused by a hurricane. If the system

preserves the integrity and confidentiality of its data and resumes its essential services after the period of environmental stress is over, the system can reasonably be judged to have fulfilled its mission. However, if the same system shuts down unexpectedly for 12 hours under normal conditions or minor environmental stress, thereby depriving its users of essential financial services, the system can reasonably be judged to have failed its mission even if data integrity and confidentiality are preserved.

Timeliness is a critical factor that is typically included in (or implied by) the very high-level requirements that define a mission. However, timeliness is such an important factor that we included it explicitly in the definition of survivability.

The terms attack, failure, and accident are meant to include all potentially damaging events; but in using these terms we do not partition these events into mutually exclusive or even distinguishable sets. It is often difficult to determine if a particular detrimental event is the result of a malicious attack, a failure of a component, or an accident. Even if the cause is eventually determined, the critical immediate response cannot depend on such speculative future knowledge.

Attacks are potentially damaging events orchestrated by an intelligent adversary. Attacks include intrusions, probes, and denials of service. Moreover, the threat of an attack may have as severe an impact on a system as an actual occurrence. A system that assumes a defensive position because of the threat of an attack may reduce its functionality and divert additional resources to monitor the environment and protect system assets.

We include failures and accidents in the definition of survivability. Failures are potentially damaging events caused by deficiencies in the system or in an external element on which the system depends. Failures may be due to software design errors, hardware degradation, human errors, or corrupted data. The term accident comprises a broad range of randomly occurring and potentially damaging events such as natural disasters. We tend to think of accidents as externally generated events (i.e., outside the system) and failures as internally generated events.

With respect to system survivability, a distinction between a failure and an accident is less important than the impact of the event. Nor is it often possible to distinguish

between intelligently orchestrated attacks and unintentional or randomly occurring detrimental events. Our approach concentrates on the effect of a potentially damaging event. Typically, for a system to survive, it must react to and recover from a damaging effect (e.g., the integrity of a database is compromised) long before the underlying cause is identified. In fact, the reaction and recovery must be successful whether or not the cause is ever determined.

Finally, it is important to recognize that it is the mission fulfilment that must survive not any particular subsystem or system component. Central to the notion of survivability is the capability of a system to fulfil its mission, even if significant portions of the system are damaged or destroyed. We use the term survivable system as shorthand for a system with the capability to fulfil a specified mission in the face of attacks, failures, or accidents. Again, it is the mission, not a particular portion of the system that must survive.

1.2 Glossary of Survivability Terms

In this section we summarize terms often used when describing survivability.

1. Accidents: A broad range of randomly occurring and potentially damaging events such as natural disasters. Accidents are often externally generated events.
2. Adaptation services: system functions provided to continually improve a system's capability to deliver essential services, typically by improving resistance, recognition and recovery capabilities.

3. Attack: A series of steps taken by an intelligent adversary to achieve an unauthorised result. Attacks include intrusions, probes, and denial of service.

4. Essential services: Services that must be provided to system users in the presence of attacks, failures or accidents.

5. Failure: A potentially damaging event caused by deficiencies in the system or in an external element on which the system depends. Failures may be due to software design errors, hardware degradation, human errors or corrupted data.

6. Detection services: System functions that detect anomalies, possible attacks, and the extent of system damage or compromise.

7. Recovery services: system functions to support the restoration of services after an attack has occurred. Recovery services also help a system maintain essential services during an attack.

8. Survivability: a system's capability to fulfil its mission, in a timely manner, in the presences of attacks, failures or accidents.

1.3 Network Survivability

Network Survivability can be defined as: (1) the ability of a network to maintain or restore an acceptable level of performance during network failure conditions by applying various restoration techniques; and (2) the mitigation or prevention of service outages from potential network failures by applying preventive techniques. Survivability techniques can be classified into three categories [23,24]: (a) prevention, (b) network design, and (c) traffic management and restoration.

Prevention techniques focus primarily on improving component and system reliability. Some examples are the use of fault-tolerant hardware architectures in switch design, provision for backup power supplies, pre-deployment stress testing of software, use of frequency hopped spread spectrum techniques to prevent jamming in military radio networks and so on. Network design techniques try to mitigate the effects of system level failures such as link or node failures by placing sufficient diversity and capacity in the network topology. For example, the use of multi-homing nodes so that a single link failure cannot isolate a network node or an access network. Traffic management and restoration procedures seek to direct the network load such that a failure has minimum impact when it occurs, and that connections affected by a failure are reconnected around the failure.

1.4 Thesis objective

Survivability, or at least graceful degradation, in the event of network failure remains with open research issues.

Knowledge of the K-best paths can be used in the network survivability problem. It is worth noting that the identification of the K-best paths can be useful in solving numerous networking problems, such as flow balancing, MPLS traffic engineering, video streaming, and so forth, as well as other diverse problems, such as molecular biology [10, 14, 15, 16, 18, 20, 29, 34].

Motivation for this thesis is the fact that finding the K-best paths in a network with hop count $h \geq 5$ in an NP-Complete problem as shown by the authors of [42,44]. Castanon in [8] gave an O($n^3 \log n$) algorithm for finding the K-best disjoint paths by using minimum cost network flow (MCNF) algorithm. However the algorithm can only be applied on a trellis graph. A trellis graph is a structured graph offering several advantages in formulating many problems of diverse fields such as radar, sonar, and radio astronomy [3, 5, 18]. In order to overcome this limitation a heuristic approach was taken by [19] in order to transforms any given network into a Trellis graph and then into an equivalent minimum-cost network problem using the Castanon [8] algorithm.

The main thesis objective is to provide extensive evaluation of the K-best paths solution by the combination of [19] and [8] and propose any necessary extensions or modifications. We adopt the transformation provided in [19] as our initial transformation and proceed to compute the K-best paths using Castanon's algorithm [8]. An animation solver is implemented and used to evaluate representative network topologies and known problematic cases, as for example the trap topology [9].

1.5 K-best Path Problem

The K-best path problem is to list the k mutually disjoint (exclusive) paths connecting a source-destination pair in a network. Mutually disjoint means with no link or node dependencies between paths listed.

The selection of the K-best disjoint paths can take into account many factors, such as selection of the shortest paths (hence minimizing delay), minimization of the bandwidth allocation (given the bandwidth demanded by customers), and maximization of network throughput. "Best" (disjoint) paths are those paths which are as diverse as possible (i.e. if the network topology permits, k disjoint paths can be

found), and therefore will maximize our chances of survivability, or ensure at least a graceful degradation, (i.e. display fault tolerance) in the event of a network fault.

1.6 Previous Work

The majority of published work concentrates on the k-shortest link disjoint path problem, [9, 26, 27, 30] rather than the K-best paths. A number of these algorithms are based on the iteration of Dijkstra's shortest path algorithm to find restoration paths for the failed links via surviving spare links on other spans of the network (referred to as the k-successively shortest link disjoint path algorithm in [9]). It is worth pointing out that once a restoration path is found, the spare links which make it up are removed from the network description, and the algorithm is run again until it fails to find any additional paths. Examples include: [9] which addresses span restoration rather than path restoration; [26] and [27] which are based on matrix and recursive matrix calculations respectively to improve computational complexity. Note that these methods are not strictly optimal in terms of finding the maximal number of paths in all possible networks, and worse they may underestimate the number of paths whenever the k-successively shortest link disjoint path algorithm selects a path which blocks other potential paths (well illustrated in [9] using the generalized "trap" topology[1]), or even worse, they may overestimate the number of link disjoint paths (e.g. [27]). On the other hand, [5, 28, 31] concentrate on finding only a pair of disjoint paths between a given pair of nodes, by optimizing the physical length of paths. In [31], the shortest pair of node-disjoint paths is found, but cannot be applied at the span (physical) level (e.g. physical links sharing a common conduit). In [5], a pair of disjoint paths between a given pair of nodes taking into consideration any span sharing by links is found, but the solution for networks with arbitrary connection patterns is not given, and in [28] a heuristic approach for finding in polynomial time a pair of paths which is as diverse as possible, taking into account common spans, is presented.

[1] Trap-topology is such a topology where a heuristic fails to find diverse routes even though they exist. A demonstration of its solution through our proposed transformation is given later.

In Chapter 2 we define the K-best path problem and proposed solution. The two proposed algorithms are outlined along with the Trellis graph theoretical model. In Chapter 3 we describe the implementation of the K-best Path problem solver and in Chapter 4 we present the evaluation of the proposed solutions through a series of transformations of illustrative topologies and well know problematic topologies. Also we present problematic topologies of the first transformation along with some solutions and their evaluation. We conclude in Chapter 5 giving future work.

Chapter 2

K-best Path Problem Proposed Solution

In this chapter we present the transformation provided in [19] and the computation of the K-best paths using Castanon's algorithm [8]. Next we present the first step in the formulation of the K-best path problem in terms of mapping the original network onto a trellis graph. After that we outline the second step, which is the algorithm proposed by Castanon [8] to compute the k-best paths from the trellis graph.

2.1 Trellis Graph Theoretical Model

A *directed graph G = (V, E)* is a structure consisting of a finite set of nodes $V = \{ v_1, v_2, v_3, \ldots, v_n\}$ and a finite set of links

$$E = \{(v_i, v_j) \mid v_i, v_j \in V \quad and \quad v_i \neq v_j\}$$

where each link is an ordered pair. We define a *trellis* as a directed graph *G = (V, E)* with nodes and directed links that satisfy the following conditions:

i. The node set *V* is partitioned into *L* (mutually disjoint) subsets $V_1, V_2, \ldots, V_L$ such that

$$|V_i| = |V_j| = H \quad .$$

ii. Links connect nodes only of consecutive subsets V_l and V_{l+1}, i.e., if $(v_i, v_j) \in E$, then $v_i \in V_l$ and $v_j \in V_{l+1}$, $1 \leq l < L$.

The magnitude T we shall call *depth* of the trellis. A *K-trellis* is a trellis graph with two additional properties:

1. It has two more nodes $s \in V_0$ and $t \in V_{L+1}$, such that the edge $(s,v_i) \in E$ for every $v_i \in V_1$ and the edge $(v_j,t) \in E$ for every $v_j \in L$.
2. The node v_i of the set V_l is connected (where possible) with $K = 2g+1$ nodes $\{v_{i-g},\ldots,v_i,\ldots,v_{i+g}\}$ of the set V_{l+1}, where $1 \le i \le H$, $1 \le l < L$, and $g = 1,2,3, \ldots, (H-1)/2$.

The depth of a K-trellis graph will be equal to L+2. In Figure 2.1 a K-trellis is presented with L = 5, H = 4 and K = 4. Throughout the paper, we shall refer to a K-trellis graph with K =H as trellis graph.

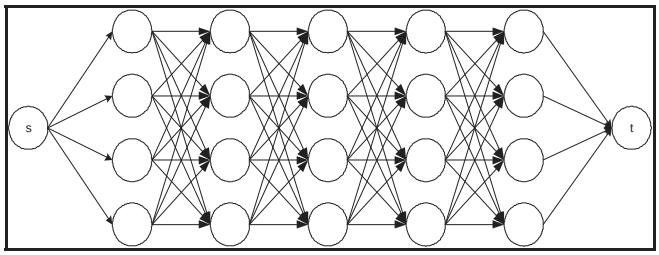

Figure 2.1: K-trellis graph with L = 5, H = 4 and K = 4

A *walk* in a trellis is an alternating sequence of nodes and links, i.e., $W=[v_1,(v_1,v_2),v_2,\ldots,v_{k-1},(v_{k-1},v_k),v_k]$. The *length L(W)* of a walk is the number of links in it. A *path* is a walk in which all nodes are distinct. For simplicity, we shall denote the path P by $P = \{v_1,v_2,\ldots,v_k\}$ and we shall refer to v_1 and v_k as *first* and *last* nodes of P, respectively (i.e. the origin and destination nodes).

Let $G = (V, E)$ be a trellis graph with LH nodes, i.e., $|V| = LH$. Two paths $P_1=\{s,u_1,u_2,\ldots,u_L,t\}$, $P_2= \{s,u'_1,u'_2,\ldots,u'_L,t\}$ are said to be *mutually exclusive* or *unmerged* if $u_l \neq u_l$ for every $l = 1,2,\ldots,L$; otherwise, they are said to be *merged*. Hereafter, we shall refer to unmerged paths as *disjoint* paths (see Figure 2.2).

2.2 Link and Path Cost

In addition to the above trellis definition, in each link $(v_i,v_j) \in E$ of a trellis graph, we associate a third number, which we call *link cost* and denote by $c(v_i,v_j)$ or $c(i,j)$. The cost of a path $P = \{v_1,v_2,\ldots,v_k\}$ is defined as:

$$c(P) = \sum_{(i,j)\in P} c(v_i, v_j)$$

The *shortest path* from the node v_i to node v_j is a path $P=\{v_1,v_2,\ldots,v_j\}$ with minimum cost.

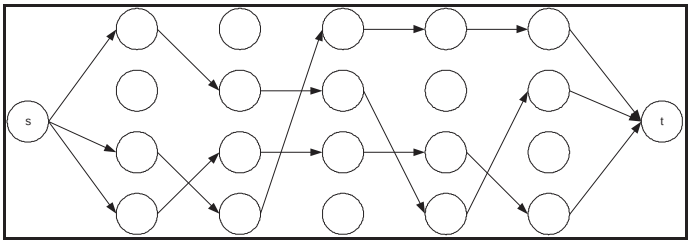

Figure 2.2: Illustration of three mutually disjoint paths on a K-trellis with L = 5, H = 4 and K = 4.

2.3 Trellis Graph Transformation Algorithm

The objective of the transformation algorithm [19] is to map any network topology G, onto a Trellis graph, and subsequently enable the shortest paths in the original graph to be computed. In this algorithm, a trellis graph is used where $K=H$. That is, every node on level V_i is directly connected with all the nodes on level V_{i+1}. Every given topology is being partitioned and labeled (see Section 2.3.1 below) with respect to the source node. The labeled network is being used by the transformation algorithm for mapping it onto a trellis graph.

2.3.1 Network Partitioning and Labelling

Given a graph $G = (V, E)$ and a node $v \in V$, a partition $L(G, v)$ of the node set V is defined (we shall frequently use the term *partition of the graph* G), with respect to the node v as follows:

$$L(G,v) = \{AL(v,\ell) \mid v \in V, 0 \le \ell \le L_v, 1 \le L_v < |V|\}$$

$$\text{where } AL(v,\ell), \quad 0 \le \ell \le L_v ,$$

are the *adjacency-level sets*, or simply the *adjacency-levels*, and L_v is the *length* of the partition $L(G, v)$. The adjacency-level sets of the partition $L(G, v)$, are defined as follows:

$$AL(v,\ell) = \{u \mid d(v,u) = \ell,\ 0 \le \ell \le L_v\}$$

where $d(v, u)$ denotes the *distance* between nodes v and u in G. Notice that $d(v,u) \geq 0$, and $d(v,u)=0$ when $v = u \quad \forall u, v \in V$. Thus, the adjacency-level sets of the graph of Figure 2.3(a), with respect to node s are shown in Figure 2.3(b), i.e.,

$AL(s, O)=\{s\}, AL(s, 1)=\{A, B, C\}, AL(s, 2)=\{D, E, F\}, AL(s, 3)=\{J, K\}$ και $AL(s, 4)=\{t\}$.

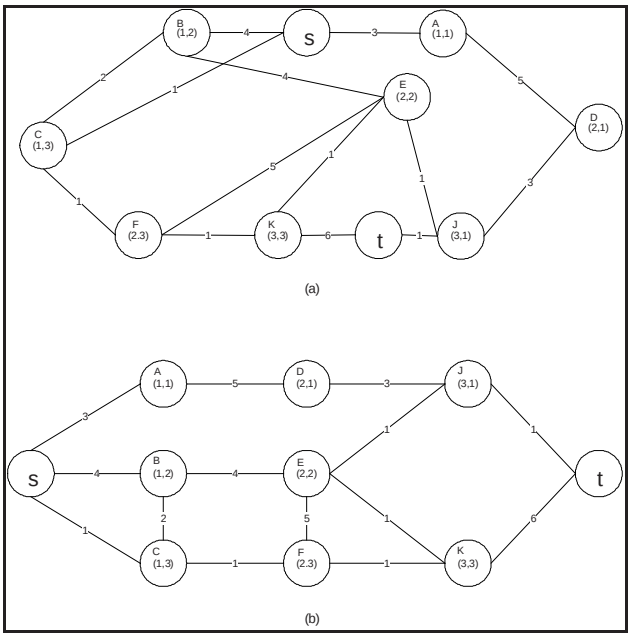

Figure 2.3: (a). A weighted undirected graph G = (V, E) or a network (all weights are positive integer numbers) (b). Partition of the graph G, with respect to vertex s

The transformation algorithm begins by labeling the nodes of the network at "vertical" levels according to their distance, in terms of the number of links, from the source node, s. These labels consist of the ordered pairs $v(d,i)$ where $v \in V$, d is the distance of node v from s (in terms of number of links) and $1 \leq i \leq deg$, where deg is the degree of graph G. The nodes of the trellis graph are eventually labeled in a similar manner. We label the nodes of the trellis graph by the ordered pairs $u(d, j)$ where $u \in U$ (the set of nodes in the Trellis graph), d is the distance from s to u, and $1 \leq j \leq H$.

2.3.2 Transformation Algorithm

In order to determine the size of the Trellis, its preliminary dimensions L´ and H´ are found. L´ is determined by taking the maximum between the number of vertical levels of the partition network and the length of the shortest path based on the number of links (excluding s and t), and not on the weight of each link. L´ minimum value is 2. H´ is defined to be the maximum between the degree of node s and t. L´ and H´ may

change depending on the final form of the Trellis. L´ and H´ are then used to create a preliminary Trellis graph which is used to map the nodes and edges of the original network *G*. The labeling of the Trellis nodes is done as described above. Once the preliminary trellis is constructed, the source node, *s*, and its adjacent nodes are mapped onto their corresponding nodes of the trellis graph. If one of the adjacent nodes of *s* is the destination node, *t*, it is moved to the end of the trellis, and the path from *s* to *t* is determined *using Find-Path(X,Y)* (described later). The second step of the algorithm is to map the end node and all it's adjacent. If an adjacent of the end node is already mapped we compute using Find-Path(X,Y) a path from the node to the end node. If more than one edge is used in determining a valid connection (using *Find-Path(X,Y)),* the extra edges and nodes are assigned value 0 to represent dummy connections at no cost. The edges and nodes which remain unmapped in the trellis graph are assigned to infinity in order to illustrate that they cannot be used in the network. The algorithm is bounded by $O(n^2)$ [19] where n is the number of nodes.

2.3.2.1 The routine FindPath(X,Y)

The routine *FindPath(X,Y)* is used to determine a path from X to Y in the trellis by considering the first available node in the vertical levels, moving in a forward direction. Nodes and edges from the original network that have already been assigned cannot be used (so that the form of the original network does not change). If there is no available node in the next level, the Trellis graph is enlarged by making H=H+1.

The final step of the routine is to assign the nodes and edges along the path to 0, except the first edge encountered which is assigned to the weight of the corresponding edge in the original network. The path established under these conditions is referred to as a *valid_path*.

2.3.2.2 The Algorithm: Convert to Trellis

Before the transformation algorithm begins the partition of the network was computed.

Step 1: Construct Preliminary Trellis

Find the shortest path in the network for the source destination pair. Compute the length of the shortest path. Computer L by taking the maximum between the

number of levels of the partitioned network, the length of the shortest path. If the L<2 then set L=2. Compute H by taking the maximum between the degrees of the start and end nodes. Create the preliminary trellis of size LH.

Step 2: Map Start Node

Map the source node, *s*, of the original graph onto the source node of the trellis. Create two sets, the first one to contain the nodes from the original network that have been assigned to the trellis graph (Nodes_assigned_original ={$v(1,i)$}), and the second one to contain the nodes from the trellis network that have been assigned nodes from the original network (Nodes_assigned_trellis ($u(x,y)$).

For all adjacent nodes of *s*, $v(1,i) \in V$ {

If $v(1,i) \neq t$ { /* *t* is the destination node */

$v(1,i)$ is assigned to their corresponding edge in the trellis, $u(1,i)$. The edge (*s*, $v(1,i)$ is assigned to the edge (*st*, $u(1,i)$).Update the two sets to contain the new nodes that have been assigned.}

If $v(1,i) = t$ {

$v(1,i)$ is assigned to the furthest node along the trellis. Find the path which connects the source node with the destination node *t* through FindPath(*s*,*t*). Update the two sets to contain the new nodes that have been assigned, including the 0 ones.}

}

Step3: Assign Shortest Path

Assign the nodes along the shortest path to their corresponding nodes and edges on the trellis and update the two sets.

Step 4: Assign the rest

For all nodes $v(d,i) \in$Nodes_assigned_original, do

/*Assume $v(d,i)$ was mapped on $u(x,y)$ {

For all nodes $v(j,k)$ being adjacent to $v(d,i)$,{

If ($v(j,k) \notin$ Nodes_assigned_original) {

Assign $v(j,k)$ to the next available node in the next level from the node $u(x,y)$ in the trellis graph. In the case that there is no available node, increase the size of the Trellis by one row or level (if it is at the last level). Assign node $v(j,k)$ to the new node. Assign the weight of edge ($v(d,i)$, $v(j,k)$) to the edge ($u(x,y)$, $u(z,e)$) where $u(z,e)$ is the node corresponding to $v(j,k)$. Update the 2 sets}

If (v(j,k) $\in$Nodes_assigned_original) {

If the connection with $u(x,y)$ and u(z,e) in the trellis is valid, (i.e. no vertical connection), then use FindPath($u(x,y)$,$u(z,e)$) to connect them. If a vertical connection exists, reassign $u(x,y)$ to the first available node. If no node is available, enlarge the Trellis, and move the node to one of the new nodes created. /* The edge will be determined when the node assigned to $u(z,e)$will be examined. */}

} /*end of second loop

} /* end of first loop

This algorithm generates a mapping where the best set of mutually exclusive K paths can directly be found using the method described in the next Section.

2.4 Castanon's Algorithm

Castanon [8] illustrates that the problem of computing the K-best mutually exclusive paths in a trellis graph can be defined as a *Minimum Cost Network Flow* (MCNF) problem. He also showed that the worst-case computation time for this problem is bounded by n^3 *log* n, where n is the number of nodes in the trellis. His $O(n^3$ *log* n) time algorithms are much faster than those proposed in [32]. It is important to note that, for $K \geq 2$ the best set of k paths is not found in general by finding the best path, and then removing it and calculating the next best path that is completely disjoint with

the best path, and so on (as in the k-successively shortest link disjoint paths [9]). The reason for this is that these paths are not independent of one another because of the requirement that the paths be completely disjoint so that it might be better to use one or more of the branches that are part of the best single path for other paths [32]. This is illustrated later during our evaluation.

The solution that was given from Castanon was to define the K-best path problem as: Find K paths $p_1,p_2, \ldots,p_k$ through a Trellis $G(V,E)$ which minimizes the total cost,

$$J = \sum_{k=1}^{K} c(p^k)$$

Subject to constrain that the paths $p_1,p_2, \ldots,p_k$ are mutually disjoint

After that definition an equivalent MCNF problem was defined that did not ensure that the path selected would be mutually exclusive. A transformation of the trellis though met that constraint. Specifically for each subset N_t where $t = 2 \ldots T$-1, a new set of nodes is created and outgoing edges which start at $n_t \in N_t$ are redefined to start from the corresponding new node that we just added. A zero cost edge is defined to connect the node $n_t \in N_t$ to the new corresponding node. Figure 2.4 below demonstrates this transformation.

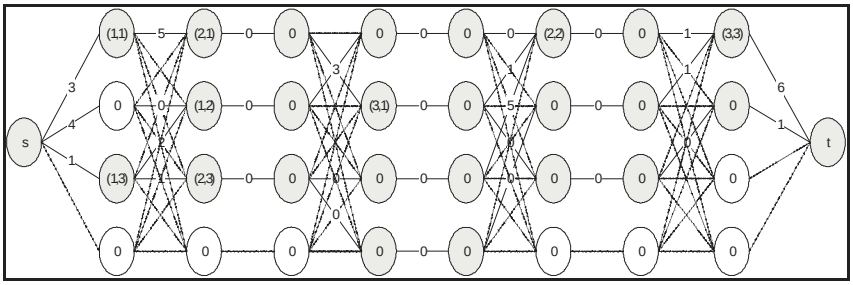

Figure 2.4: Castanon's AlgorithmTransformation

To solve the MCNF problem one may use the solver provided by the Neos Server [39], [40], [41] to solve it through Relax IV algorithm created by Bertsekas in [4]. This is the approach we took in order to solve the problem. The animated solver creates a file which contains the definition of our trellis in DIMACS format. This is then sent to the NEOS server (http://www-neos.mcs.anl.gov/) to be solved using the Liner Network Optimization Solver RelaxIV written by Bertsekas et al [4]. The MCF results are obtained from the server and analysed locally.

When we receive the MCF results from the NEOS server the flow for each arc is either one or zero. The arcs that have flow equal to one belong to the K-paths. We then mark the paths to the trellis and show the results.

Chapter 3

Algorithm Solver Implementation

The two algorithms described in Chapter 2 have been implemented using an object-oriented approach. The implementation was enhanced with a graphical user interface to provide a complete solver of our problem. One can use the GUI to design the topology and then pass it to the algorithm package to solve it and finally present the solution. The animated solver of [37] for finding the shortest path has been adapted in order to provide the Graphical User Interface in our implementation. The language of implementation is Java. The objects of the graph are implemented through classes which describe the characteristics of the objects through variables and their functions through methods. Objects communicate among themselves via messages.

The solver provides the ability to find the K-best paths through the conversion of the Trellis produced, into a Minimum Cost Network Flow problem as described by Castanon in [8]. The MCNF problem formulation of the Trellis Graph is then submitted for solution by the Relax IV algorithm written by Bertsekas [4] and hosted at Neos Server [39,40,41].

The model contains one Java interface called constants which defines some constants used by the various classes for the definition of the values of some of their characteristics. For example, the definition of infinity and dummy nodes are defined through this interface. The classes Node, Network, ShortestPath, TrellisGraph implement it.

3.1 Algorithms Logic Implementation

In this Section, the classes which implement the algorithm logic are presented. The implementation of the algorithm was based in object orientation in order to be able to distinguish between the various actions that a node had to take. This will make it easier if one wants to transfer this algorithm in a real network.

3.1.1 Edge

The class *Edge* defines the connection between two nodes. Its characteristics are the source node, the destination node and the weight on the link of the two nodes. Every object can return its characteristics and change the weight of the link.

3.1.2 Node

The class *Node* defines a node in a graph. Its main characteristics are the name of the node, its type (a source node, a destination node, or any other node within the graph) and the edges initiating from that node. In addition, it defines the position of the node in a labelled graph with respect to the source node, and its previous node in a path.

3.1.3 EdgeList

The class *EdgeList* is a set of edges. It is being used to keep the edge of a network and the edges initiating from a node. It can return the edges with respect to their position in the set, and with respect to the destination node, if such a node exists in the set.

3.1.4 NodeList

The class *NodeList* is a set of nodes. It is being used to keep the nodes of a network as well as for the creation of the two sets being used by the algorithm (nodes-assigned_original, nodes_assigned_trellis).

3.1.5 Network

The class *Network* defines a network. Every network can be represented by a set of nodes and a set of edges. The class has as main characteristics the two sets NodeList and EdgeList which are being implemented as objects. In addition, it can define a partitioned network which has two more characteristics, the source node and the final node.

3.1.6 ShortestPath

The class *ShortestPath* implements Dijkstra's algorithm for finding the shortest path in a network.

3.1.7 TrellisGraph

The class *TrellisGraph* implements a Trellis graph. The Trellis graph is being characterized by the two sets of nodes and edges which are implemented by the classes *NodeList* and *EdgeList* respectively. Given a network topology, this class transforms it into a Trellis using the transformation algorithm.

3.2 Graphical User Interface

In this Section, the classes used to implement the graphical user interface are being described. The classes were first implemented by [37] and were adapted in order to run the algorithm logic.

3.2.1 DocOptions

The class *DocOptions* has all the various user choices for the application and transformation of the original network onto a Trellis graph.

3.2.2 DocText

The class *DocText* enlarges the *TextArea* and contains documentation on the various user choices. This class is being called by *DocOptions*.

3.2.3 Documentation

This class contains the above two classes and is responsible for the presentation and explanation of the various functions, or other explanations during the algorithm execution. Figure 3.1 shows the location of the class on the animator

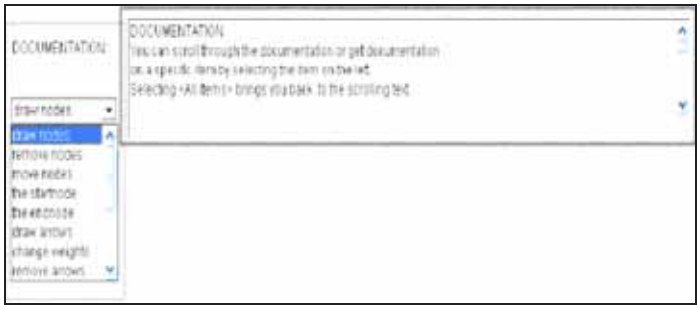

Figure 3.1: The class Documentation which contains the class DocOptions and DocText

3.2.4 GraphCanvas

The class *GraphCanvas* extends the class Canvas and is responsible for the presentation of the graphical network and Trellis. It uses the classes which implement the algorithm in order to pass them the network graphically designed by the user, to use it in the transformation.

3.2.5 Options

The class *Options* contains all the various user choices. Depending on what the user chooses, *GraphCanvas* will execute the analogous task and presents the result to the user.

3.2.6 GraphAlgorithm

This class extends the class Applet and it is the main class of our application. This class contains the objects which implement the graphical representation of the algorithm. Figure 3.2 shows the location of the classes on the animator.

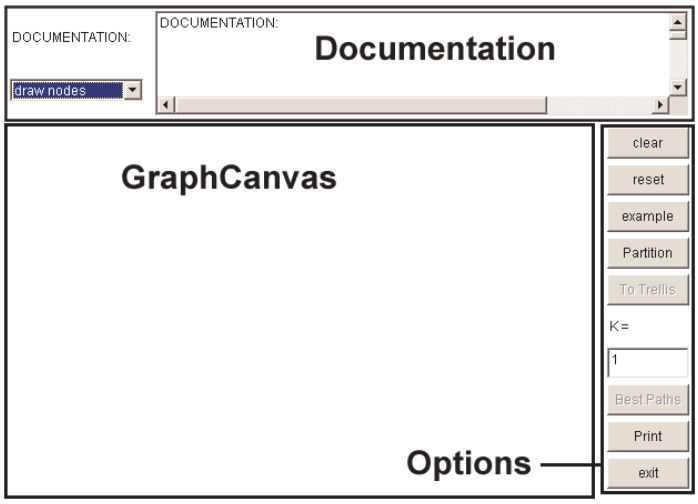

Figure 3.2: The class GraphAlgorithm

3.3 Graphical User Interface Usage

The usage of the graphical user interface is simple. When started, the user can click on the canvas to create a node. The first two nodes created are the start and end node painted as blue and yellow respectively. The rest of the nodes are painted grey. The user can then connect the nodes by clicking on one node and while pressing the mouse button move to another node and releasing the mouse button. This will create a link with initial weight equal to 50. The weight can be then adapted by moving the arrow on the edge created. In order to move a node the user can click on the node and while pressing the SHIFT button on the keyboard move the node to another position on the canvas. In order to change the source or destination the user can click on either the source or destination and while pressing the CTRL button move the blue or yellow circle to another node. To delete a node or link the user has to first click on the node or link and then press the DELETE button on the keyboard.

When the user finishes designing the network he presses the “Partition” button on the options pane and the network is partitioned in respect of the blue node (the start node). After this step is complete the user can then click on the “To Trellis” button to map the partitioned network to a Trellis. While the algorithm runs the user can see the actions taken on the Documentation text area.

The K-best paths can be computed when the transformation is complete by entering K in the text box and clicking on “Best Paths” button. This will convert the Trellis to a Minimum Cost Network Flow problem as described in Castanon’s algorithm and send

it to Neos Server [39,40,41] in order to solve it through Relax IV algorithm created by Bertsekas in [4]. The results (K-best Paths) are then presented to the canvas with yellow arcs. The actual results received by the Neos Server are displayed on the Documentation text area.

Chapter 4

Evaluation of the K-best Paths Algorithm

In this chapter we will evaluate the computational complexity of the proposed algorithms [8], [19], and compare with the well known algorithm of Surballe [30],[31]. Also we will use representative topologies and well known problematic cases to evaluate the behaviour of the proposed algorithms.

4.1 Complexity Analysis

The transformation described, requires $3n^2+(n^2+2L+L)H$. A discussion for the time complexity at each step of the algorithm follows: The initial step of this algorithm is the process of labelling every node in the original network. The labelling can be done with a slight modification to the all-pairs-shortest-path algorithm, used for finding the distance in terms of the number of links between the source, s, and the rest of the nodes. Such algorithms are of the order of n^2. The first step of the algorithm is the construction of the preliminary Trellis which requires at most L^2H^2. Dijkstra's algorithm is used to find the shortest path from source node, s, to destination node, t, requires n^2 steps. In step 2, we map the start node s and the end node t along with their adjacent nodes onto their corresponding nodes of the trellis. If none of the adjacent

nodes of the start node is the destination, t, and then none of the adjacent nodes of the t is also adjacent to s then this step requires at most 2H steps. If one of the adjacent nodes is the destination, we need to determine the path from s to t, which requires in this case L steps. If a node is adjacent to both start and end node then we will also need to determine the path between that node and the end node. This is bounded by LH because at most all nodes are also adjacent to both start and end. Thus, the total time for this step is 2H+LH. In step3, the first loop will be repeated n times and the second one is executed at most H times. The first if statement (Node is not assigned) takes only one step to assign to finish) and the second one (Node is assigned) takes L steps to finish thus giving as at most computation time n2+n2H. Therefore, the total time required for this algorithm is $3n^2+(n^2+2L+L)H$ and we can say that it is bounded by $O(n^2)$.

4.1.1 Comparison with Surballe's disjoint pair algorithm

Surballe's algorithm [31] finds the shortest pair of disjoint paths. This algorithm is famous for its polynomial computational complexity in solving optimal disjoint path-pairs in terms of the cost path of the two paths on a directed graph. It uses Dijkstra's algorithm in order to find the shortest path for every individual node in the network. There are two versions of the algorithm. The edge-disjoint and the vertex-disjoint shortest pairs.

In the edge-disjoint version, the two paths that are going to be selected through out the network should not have any common links. The algorithm considers the cost of each link as a measure to select the pair of paths. The vertex-disjoint shortest pair algorithm considers disjoint vertices for the paths.

Next, the transformation algorithm in combination with Castanon's algorithm for finding K-best paths throughout a trellis graph [8], is compared with Surballe's algorithm [31].

Even though the two algorithms perform the same task, the one (Castanon's) has the ability to give more optional paths (k-paths), whereas Surballe's algorithm gives only a pair of paths. There exist a tradeoff between the number of alternate disjoint paths and their optimality. Even though Castanon's algorithm may produce more paths they

may not be the optimal ones. This is due to the fact that in order to get the K-best paths other “shortest” paths may be ignored.

Furthermore, as stated earlier, the transformation algorithm is bounded by O(n^2). This time is needed just to transform a given network into a Trellis. In order to get the K-best paths we need to use the Castanon’s algorithm for K-best paths. That algorithm requires O($n^3 \log n$) steps; therefore, using both algorithms will result to the time complexity of O($n^6 \log n$). Surballe’s algorithm, on the other hand, requires O($n \log_{(1+n/m)} m$)[31, 36].

Thus, even though Surballe’s algorithm consumes less time and has better computing performance than our algorithm, it only finds a pair of disjoint paths, whereas, the trellis transformations can find K-best paths, with only a modest increase in computational complexity.

4.2 Illustrative Solutions

In this section some illustrative solutions of the algorithm will be presented along with their respective time taken for solution.

4.2.1 Sample topology 1

Figure 4.1 shows the illustrative example selected in [19], which demonstrates a topology for which the k-successive shortest path approach will fail. Figure 4.2 shows how the network is partitioned and labelled and Figure 4.3 the resulting trellis graph. Figure 4.4 and Figure 4.5 show the K-best paths for K=1 and K=2.

It’s worth noting that for K=1 the solution of the algorithm gives the shortest path which if chosen the network has no alternative path. The solution though for K=2 gives two alternative paths leaving the shortest path out.

This topology has 10 nodes and took 110 milliseconds to be mapped to trellis. The resulting trellis has 20 nodes and took 0.071 milliseconds to be solved by the Neos Server.

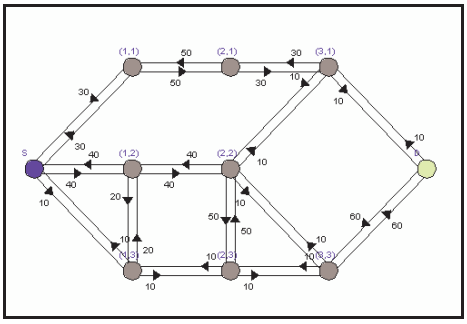

Figure 4.1: Example Network

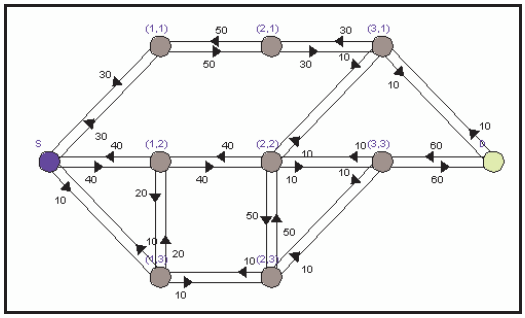

Figure 4.2: Partitioned Network

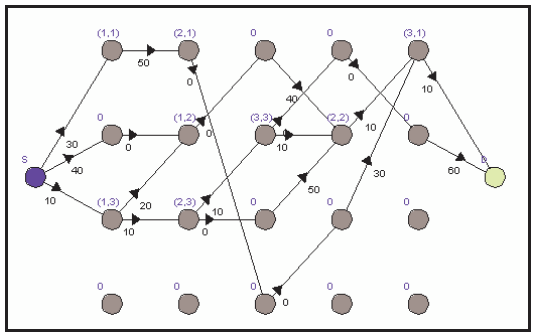

Figure 4.3: Trellis Transformation

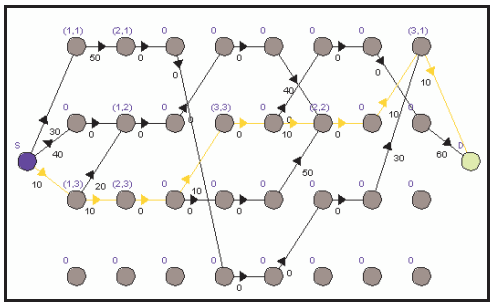

Figure 4.4: K-best Paths with K=1

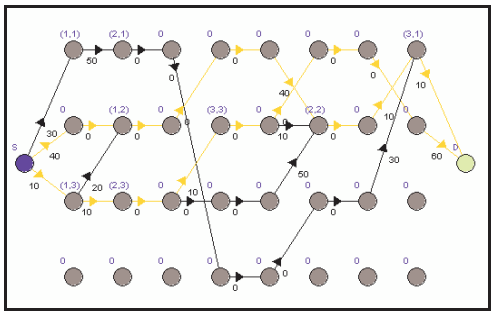

Figure 4.5: K-best Paths with K=2

4.2.2 Sample topology 2

The topology shown in Figure 4.6 has three mutually exclusive paths for which if the shortest path is selected it blocks all three paths.

The resulting solutions from the Castanon's algorithm for K=1,2,3 is shown in Figures Figure 4.8, Figure 4.9, Figure 4.10 and shows that it chooses the best set for all three cases. For K=1 the shortest path, for K=2 it ignores the shortest path and selects the two next (best) paths that are again the shortest but not still blocks the third path and for K=3 it selects the three mutually disjoint paths ignoring again the shortest paths selected for K=2.

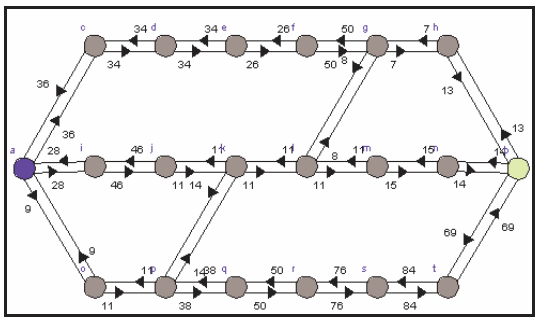

Figure 4.6: Shortest path blocks all three mutually exclusive paths

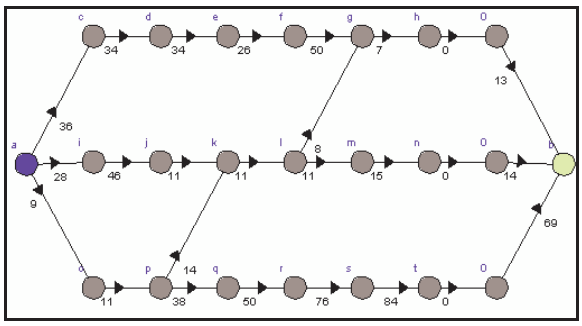

Figure 4.7: The Trellis Transformation

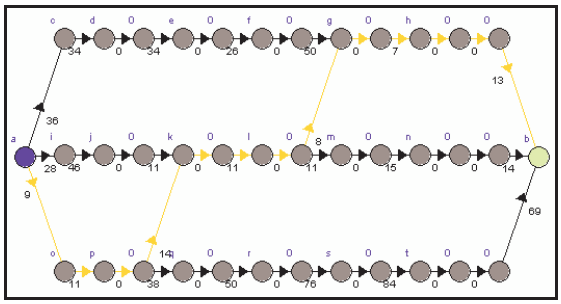

Figure 4.8: Solution for K=1

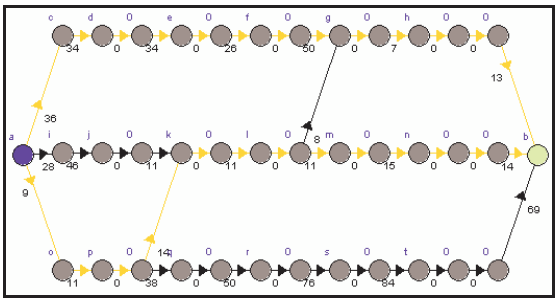

Figure 4.9: Solution for K=2

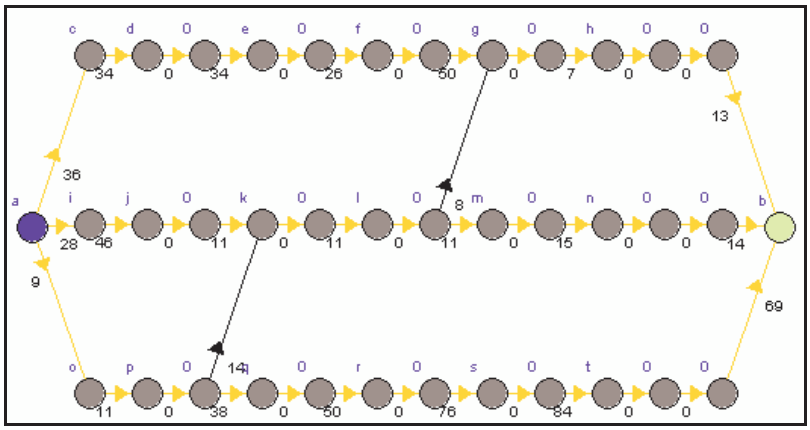

Figure 4.10: Solution for K=3

The time taken for the transformation of the topology to a Trellis graph was 172 milliseconds and the Castanon's solutions was 0.084 milliseconds.

4.3 Algorithms Drawbacks Identified and Possible Solutions

During the algorithm evaluation many scenarios were transformed in order to ensure that the transformation algorithm was performing as expected. These transformations let to the discovery of two problems. The problems identified had to mainly do with the positioning of the nodes in the Trellis. During some transformation we noticed that due to incorrect node assignment stranded nodes were created. A stranded node is

a node in the Trellis that does not have any outgoing links. The problems were identified to be caused due to the inability of the existing algorithm to choose the correct level for inserting a new node or when selecting which node to move during the assignment of a vertical link.

4.3.1 New Node Insertion

The algorithm states a new node is inserted when an edge is found connecting an existing node to it and is assigned at the first available node in the Trellis found at the next level. This leads to the creation of stranded nodes due to the fact the node may be placed too early in the Trellis, thus nodes connecting to it will not be able to, or too far in the Trellis, thus any nodes it should connect to will be at earlier levels.

4.3.1.1 Proposed Solution

Heuristic: Given two nodes n1 already assigned, n2 not assigned:

- Assign n1 at the next level only if there exist a node at a next or same level or same level connecting to n1
- Else assign the new node at the same level as n1 and do not connect them
    - The connection will be done when evaluating n2

This proposed solution showed that it avoids the insertion of nodes too far in the Trellis thus avoiding the creation of stranded nodes.

Such networks are shown in Figure 4.11 and Figure 4.12.

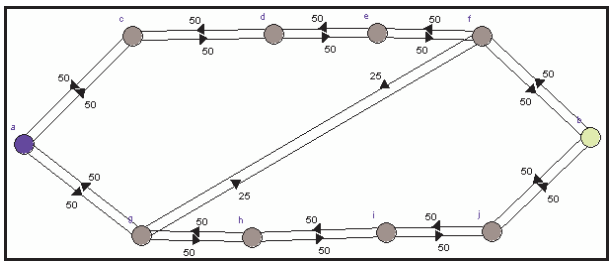

Figure 4.11: Problematic Topology

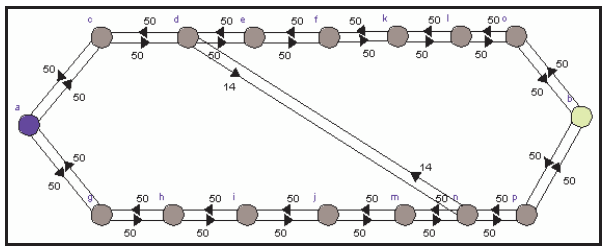

Figure 4.12: Problematic Topology

The partition of topology in Figure 4.11 is shown in Figure 4.13 and the trellis transformation in Figure 4.14. As we can see in the transformation node f is placed

too early in the Trellis Graph thus being mapped into the trellis before node e. This leads to the creation of a stranded node, node e, and loss of path [a, c, d, e, f, b] because the trellis graph does not allow links going backwards.

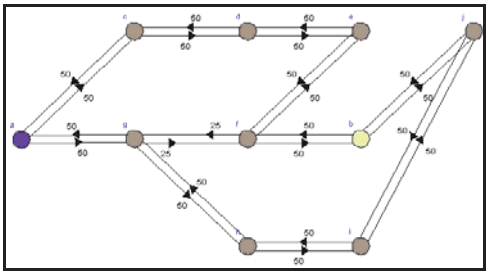

Figure 4.13: The Partitioned Network

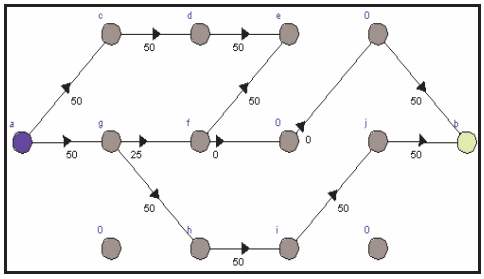

Figure 4.14: The Trellis Transformation

After the heuristic the new Trellis transformation is shown in Figure 4.15 and node e is now before node f eliminating the stranded node.

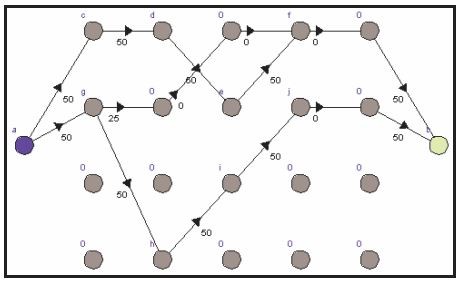

Figure 4.15: Trellis Transformation after the heuristic

4.3.2 Vertical Link assignment

During the assignment of a link between two already assigned nodes that are on the same level one of two nodes needs to be moved to the next level. The algorithm does not state which of the two nodes should be moved. During the implementation of the algorithm the node being assigned was chosen as the node to be moved. This choice let in some topologies a node to be moved too far in the Trellis thus creating a stranded node. After the evaluation of this problem the following heuristic were proposed and applied.

4.3.2.1 Proposed Solution

Heuristic: Given two nodes on the same level n1, n2 check:

- If n2 has no other nodes connecting to it from the previous or same level move all nodes of the same level except n1 to a new level

- Else if there exist a node not yet assigned for n1 or there exist a node already assigned but to a next level for n1 then move n1 to the next level.

- Else move all nodes of the same level except n1 to a new level.

This heuristic was applied and transformation of otherwise problematic topologies showed that by choosing which node to move to the next level we are able to minimize the possibility to allow the creation of stranded nodes thus maximising the ability of the algorithm to find the K-best Paths. This solution however does work in the specific case when there exist two vertical links for the node in question and the node is not connected to any other nodes except these two. This drawback is illustrated in section 4.4.2. It is worth noting that this does not affect the ability of the algorithm to find the correct k-best paths due to the fact the node is redundant. A direct path exists to connect the two nodes and if the node in question is connected to other a third node then the transformation does not give a stranded node.

4.4 Known Problematic Topologies

In this section we demonstrate the ability of the algorithm to find the K-best paths in known problematic topologies found in the literature.

4.4.1 The Trap topology [9]

The topology shown in Figure 4.16 was demonstrated in [9]. There, it was called the generalized trap–topology, due to the blocking of the 2^{nd} shortest path, if one selects the direct route a-b-c-d first. This is a well known limitation in algorithms which successively select the shortest path, and then remove all links using it from the list of available links, as in the k-successive shortest path approach.

Figure 4.16 and 4.17 shows the partitioned and trellis transformation for the trap topology respectively. Figure 4.18 shows the K-best paths for K=1 where it is noticeable that the algorithm returned the “trap” path which is also the “best” path as it is the shortest. In Figure 4.19 we can notice that the algorithm did not use the “trap” path rather that it returned the two other alternative paths.

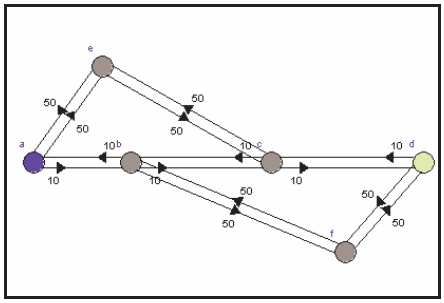

Figure 4.16: The Trap Topology

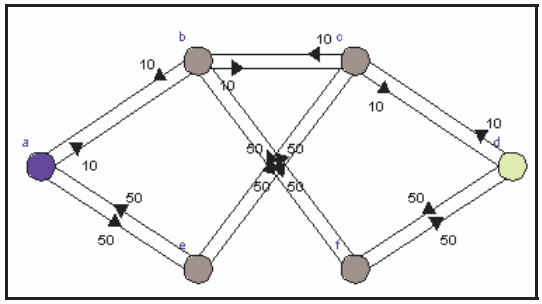

Figure 4.17: Trap Topology Partitioned

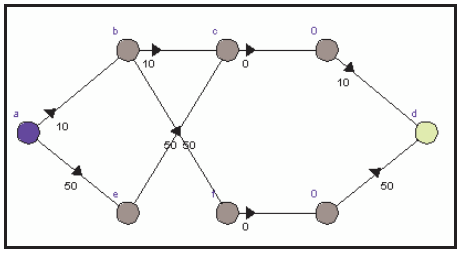

Figure 4.18: Trellis Transformation of Trap Topology

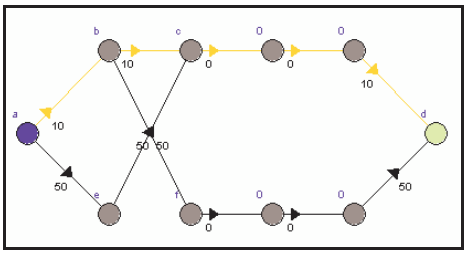

Figure 4.19: The Trap Topology with K=1

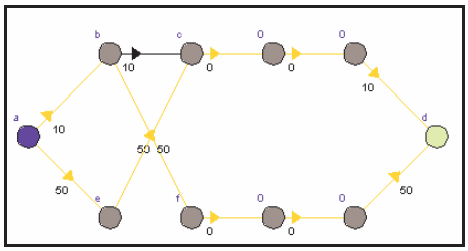

Figure 4.20: The Trap Topology with K=2

4.4.2 A difficult trap topology demonstrated in [45]

The authors of [45] address the spare capacity allocation problem (SCA) considering any single node failure in mesh networks. The spare capacity allocation (SCA) problem is to decide how much spare capacity should be reserved on network links for given traffic flows and their working paths on two-connected mesh networks. It is part of survivable network design and is NP-complete [46]. The SCA node failure problem aims at finding backup routes and providing sufficient spare capacity to protect traffic when any single node fails in communication network. The authors propose a novel matrix formulation of the arc-flow SCA node failure model in order to address the problem.

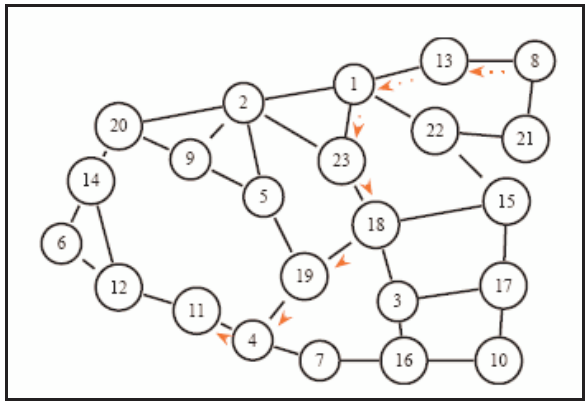

Figure 4.21: A path from node 8 to 11 leaving the source destination pair with no alternative path

In order to provide node-disjoint backup paths, the authors have to guarantee that each working path has at least one node-disjoint backup path on the given 2-node-connected topology. This task is not trivial since working path found by general shortest path algorithms can not guarantee this property and may be infeasible for SCA node failure problem. An example of infeasible working path is shown in Figure 4.21. The working path from node 8 to node 11 is 8-13-1-23-18-19-4-11, where all the numbers between 8 and 11 are intermediate nodes on the path. This path has shortest hop but it does not have a node-disjoint backup path! This paper deals more with finding a working path and a node-disjoint backup path rather that finding the K-best paths. Their algorithm tries to remove the “trap” nodes which make the finding of node-disjoint backup path infeasible in order to solve their problem.

The proposed solution using our algorithm shows that the problem is solved and two alternative paths are found to connect 8 and 11 (h and k respectively in the solver) that are mutually disjoint.

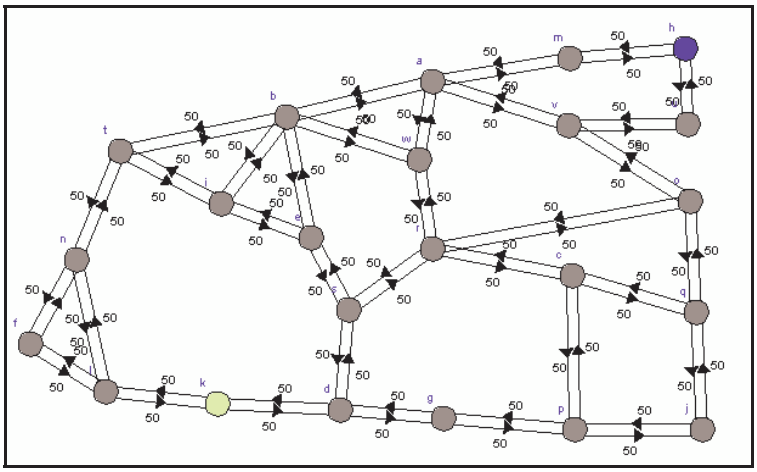

Figure 4.22: The Network described in [45]

The Network is designed in the solver, Figure 4.22. The node numbers are changed to letters and are assigned respectively in the order the letters are found in the alphabet (a=1, …, h=8, …, k=11, …). Figure 4.23 shows the partitioned network in respect with node h (node 8 in Figure 4.21).

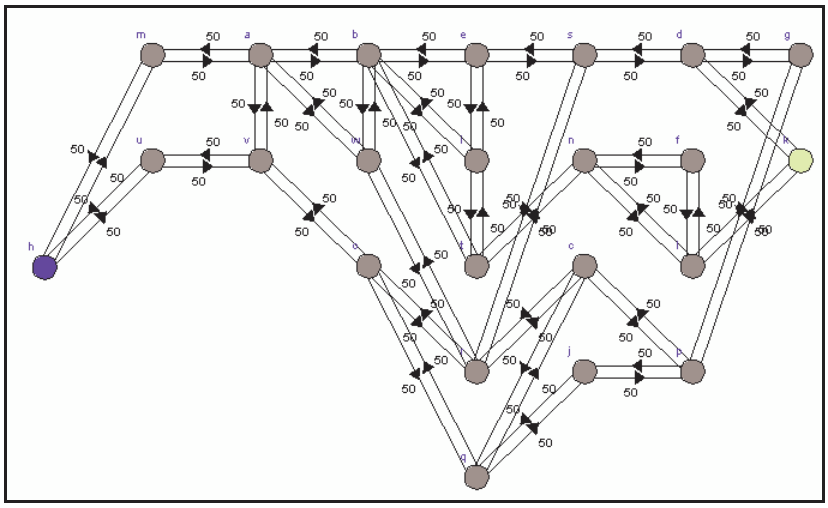

Figure 4.23: The Partitioned Network described in [45]

Figure 4.24 shows the Trellis transformation of the network and Figure 4.25 and Figure 4.26 the solutions for K=1 and K=2 respectively. The solutions found for K=1 and K=2 show that the algorithm again selects the shortest path, the “trap path” for K=1 and ignores it for K=2 selecting two alternative paths that are mutually exclusive.

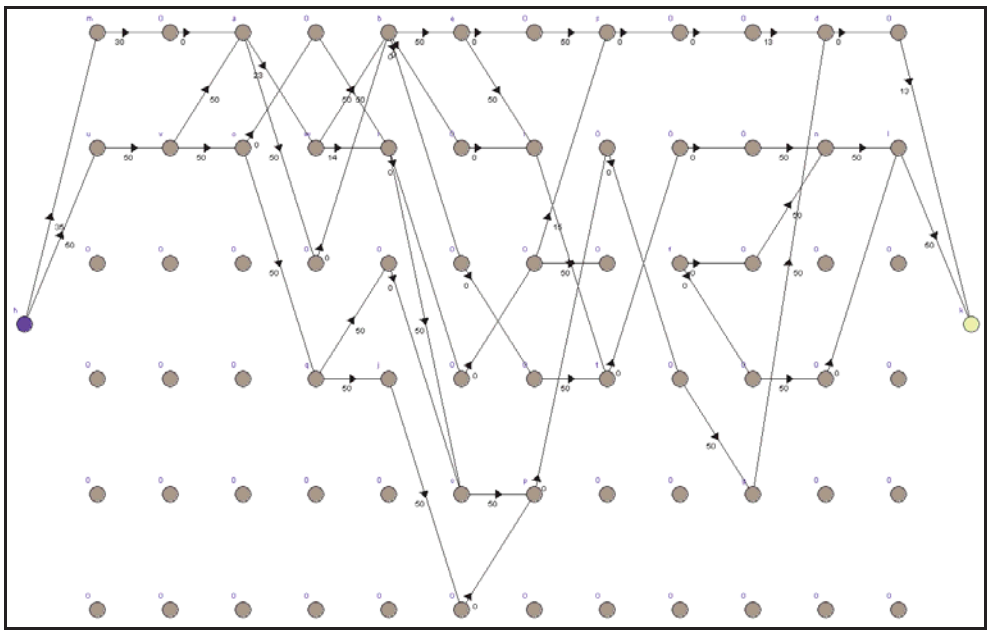

Figure 4.24: Transformation of Network described in [45]

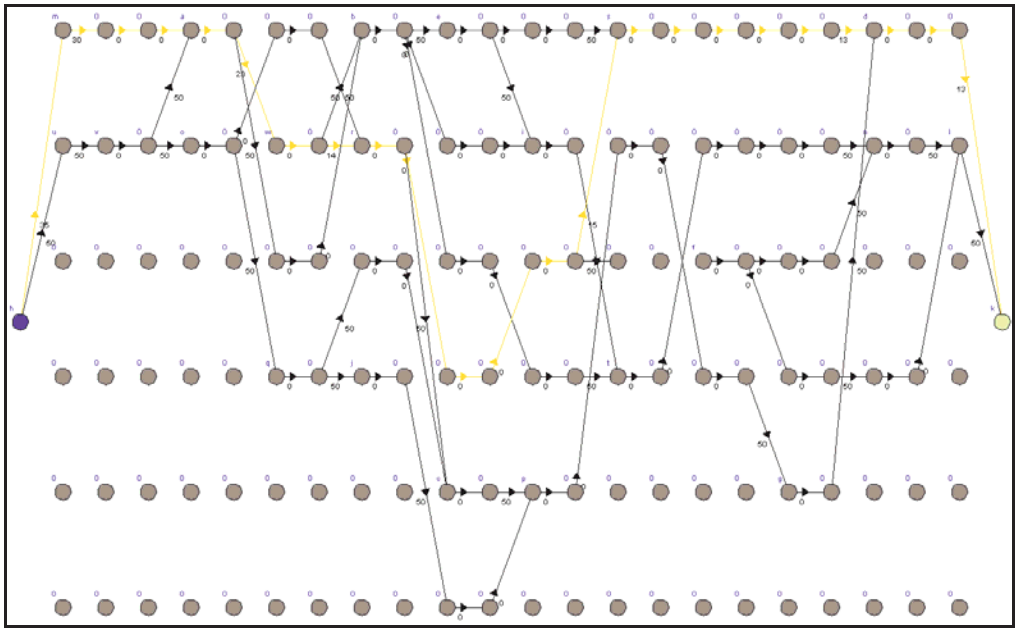

Figure 4.25: Shortest path selected if K=1

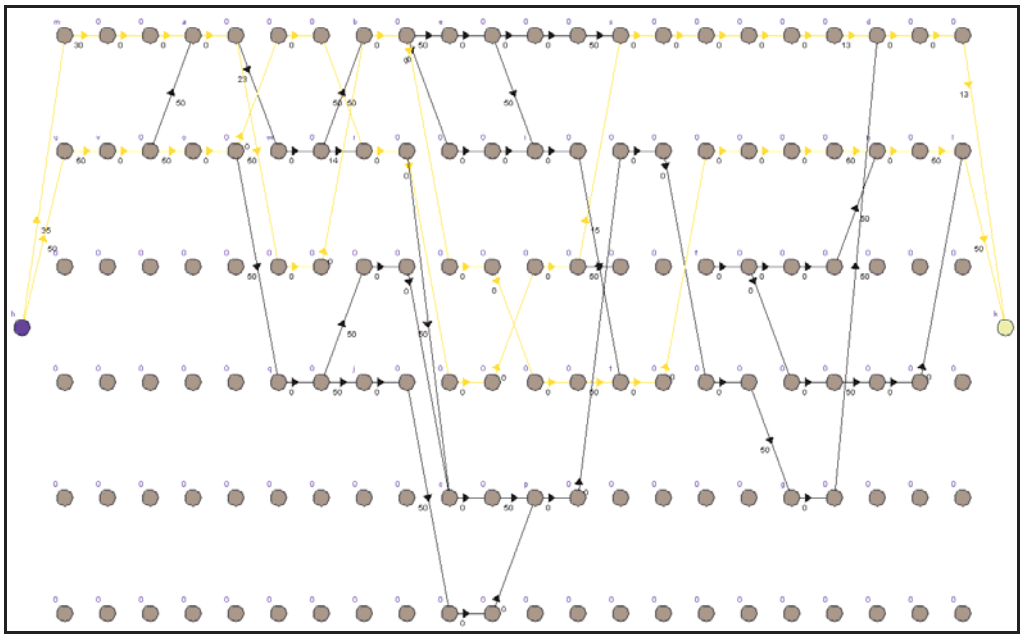

Figure 4.26: Two alternative paths for the topology found in [45]

Although the algorithm solves the problem and gives two mutually disjoint paths the transformation fails to ensure that all possible paths could be evaluated due to the fact Node f has two links to nodes directly connected together therefore creating two vertical links and making the heuristic unusable thus having the stranded node effect expained in section 0. This however does not limit the number of paths due to the fact that node f is not needed. As we can notice in Figure 4.22 node f is only connected to d and l that are directly connected to each other. A very similar topology shown in

Figure 4.27 that connects f with e has the transformation in Figure 4.28 and as we can see node f is not stranded.

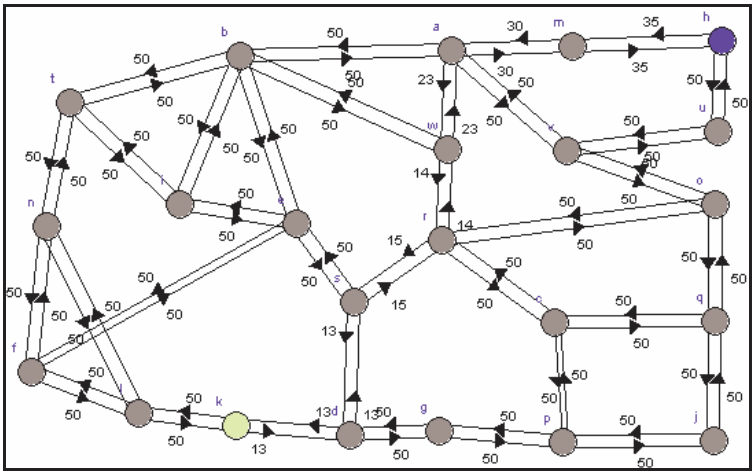

Figure 4.27: Small Difference in Topology from [45]. Node f and e are Connected

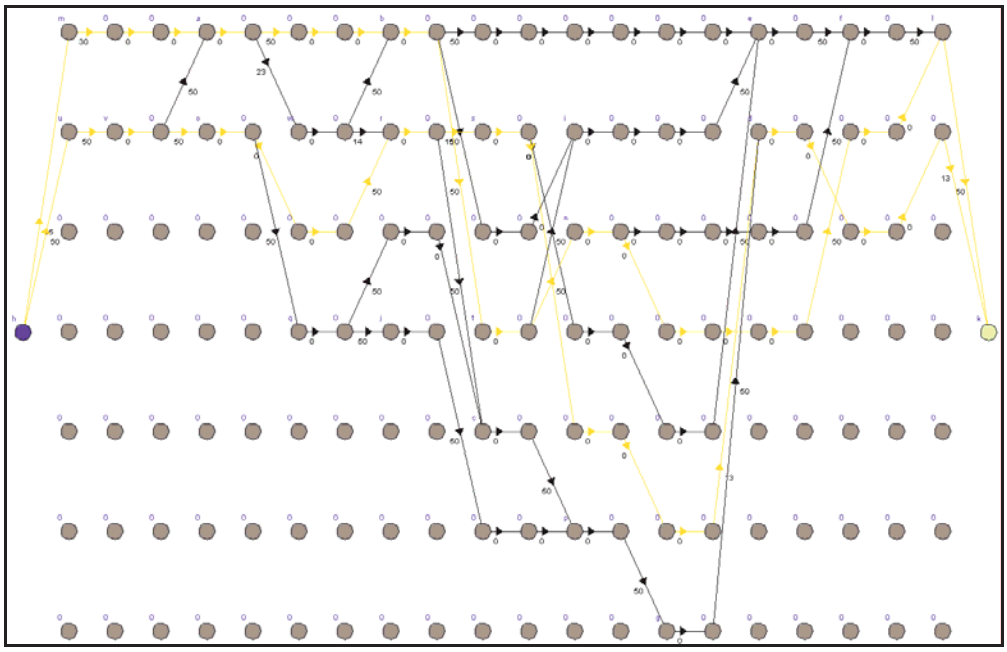

Figure 4.28: Transformation with no stranded nodes after connecting node f with e

4.5 Computation analysis

In order to validate the computational complexity of the algorithm a series of topologies were transformed taking the time it takes for each transformation. These topologies all had the same characteristics and showed the same behaviour during the transformation. Each topology transformation was timed 10 times and after removing the best and worst time an average of the remaining times was taken. The number of nodes for each topology was increased each time from 20 nodes to 156. Specifically

the topologies were 20, 30, 42, 56, 72, 90, 110, 132, and 156. Figure 4.29 shows four of these topologies specifically the topologies with 20, 30, 42, 56 nodes in order to visualize how these topologies were selected. The results collected are shown in Table 4.1. All results were run on a Pentium IV running at 3 GHz with 1 GB of RAM.

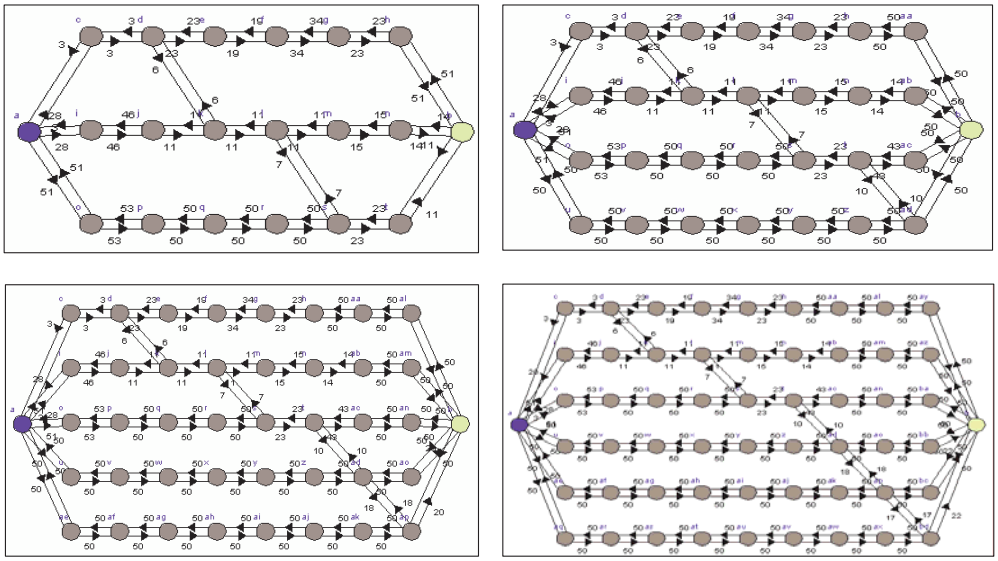

Figure 4.29: Time analysis topologies

Table 4.1: Execution time for each topology

Number of Nodes	Average Executions Time in Milliseconds
20	477.375
30	824
42	1457.5
56	2402.25
72	3785.625
90	5720.625
110	8265.5
132	11855.38
156	16464.75

In order to make a prediction for the behaviour of the transformation algorithm we interpolated the results for the following series, 20:5:160, 20:5:250[2]. The resulting graph is shown in Figure 4.30. The figure shows in the X axis the number of nodes

[2] 20:5:160, 20:5:250 mean values that start from 20 and end to 160 and 250 respectively increasing each time by 5. e.g. [20, 25, 30, 35 …, 155, 160].

and in the Y axis the execution time for the actual results on the blue line, the interpolated results for 20:5:160 on the green line and the interpolated results for 20:5:250 on the red line. As we can see the graph shows a curve similar to an $y = x^2$ as expected since the complexity analysis showed $O(n^2)$ for the transformation algorithm.

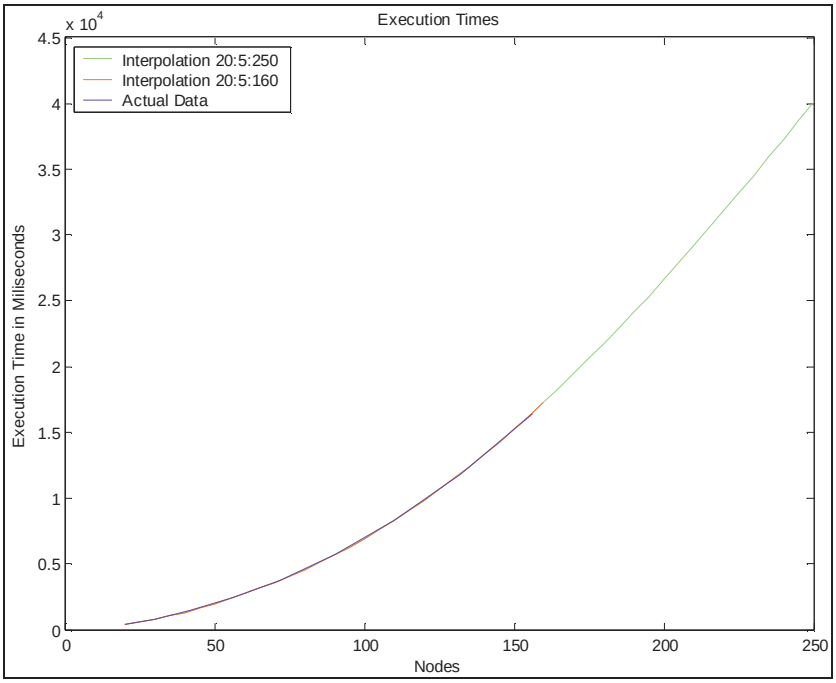

Figure 4.30: Execution times

In order to obtain computation results for the Castanon transformation we had to run the resulting trellises on the NEOS server and get the execution time. Since we had no control on the NEOS server several attempts failed due to unexpected load on the server. For instance we had smaller execution times for solutions for the resulting trellises of the 90 nodes topology than the 42 nodes topology. Due to this problem these computation times are not analyzed in the thesis.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we evaluate a solution that uses graph theoretic techniques to address the problem of network survivability by finding the K-best (disjoint) paths. The algorithms proposed transform the original network topology onto a trellis graph and later transform the problem of finding the K-best paths on a Trellis to an equivalent Minimum Cost Network Flow problem. In order to perform our evaluation a solver for that implements the transformation of any given network topology to a Trellis as described by Louca et al in [19] was implemented and complemented with a Graphical User Interface. The solver also transforms the Trellis into the MCNF problem as proposed by Castanon in [8] and send to NEOS server [41] to be solved by Relax IV written by Bertsekas [4].

The evaluation showed that the proposed solution can find the K-best paths and outperforms proposed algorithms in which k-successive algorithms fail such as the "Trap" topology [9] and a more complicated one illustrated in [45]. The evaluation also revealed two problems of the transformation to trellis algorithm and two proposed solutions were give that overcome these problem.

The time complexity of the transformation algorithm is investigated and in combination with Castanon's algorithm [8] for finding the K-best disjoint paths, is compared with the Surballe's Disjoint Pair Algorithm [31]. It is shown that even though Surballe's algorithm appears to have a better time complexity, the algorithm deals with only a pair of paths, whereas the trellis transformation's algorithm can find all possible disjoint paths. The extra computational burden is within acceptable

bounds. Furthermore a computation analysis of the transformation was performed that validated the complexity analysis performed.

5.2 Future Work

Future work includes optimizing the transformation of trellis graph; that is finding a trellis graph with the minimum number of nodes in order to do the transformation of the original network. Furthermore, the initial size of the Trellis will be studied so that no enlargement will be necessary during the transformation process in order to minimize the execution time of the algorithm. Other future work includes in identifying where the topology has a bottleneck in terms of not allowing multiple mutually disjoint paths and automating the procedure of finding the maximum number of K-best (disjoint) paths in a network.

Bibliography

[1] Y. K. Agarwal, “An Algorithm for Designing Survivable Networks”, AT&T Technical Journal, Vol. 68, No. 3, pp. 64-76, 1989.

[2] J. Anderson, B. T. Doshi, S. Dravida, P.harshavardhana, “Fast Restoration of ATM Networks”, IEEE J. on Sel. Areas in Comm., Vol. 12, pp. 128-138, 1994.

[3] R.E. Bellman and S.E. Dreyfus, Applied Dynamic Programming, Princeton University Press, Princeton (1962).

[4] D.P. Bertsekas and P. Tseng “RELAX-IV: A Faster Version of the RELAX Code for Solving Mimimum Cost Flow Problems”, Report LIDS-P-2276, Lab. for Information and Decision Systems, M.I.T., Nov. 1994.

[5] R. Bhandari, "Optimal diverse routing in telecommunication fiber networks", IEEE INFOCOM'94, Ontario, Canada, pp. 1498-1508, June 1994.

[6] W.G. Bliss and L.L. Scharf, "Algorithms and Architectures for Dynamic Programming on Markov Chains", IEEE Trans. ASSP 37, pp. 900- 912, 1989.

[7] R. H. Cardwell, C. L. Monma, T.H. Wu, “Computer-Aided Design Procedure for Survivable Fiber Optic Networks”, IEEE J. on Sel. Areas in Comm., Vol. 8, pp. 1188-1197, 1989.

[8] D. Castanon, "Efficient Algorithms for Finding the K Best Paths Through a Trellis", IEEE Trans. AES 26, pp. 405-410, 1990.

[9] D.A. Dunn, W.D. Grover, M.H. MacGregor, “Comparison of k-shortest paths and maximum flow routing for network facility restoration”, IEEE Journal of Selected Areas in Communications, vol. 12, no. 1, pp. 88-99, January 1994.

[10] Guo, Y., F. A. Kuipers, P. Van Mieghem, 2003, "Link-Disjoint Paths for Reliable QoS Routing", International Journal of Communication Systems, vol. 16, pp. 779-798.

[11] F. Harary, Graph Theory, Addison-Wesley, Reading (1969).

[12] M. Herzeberg, S. J. Bye, A. Utano, “The Hop- Limit Approach to Spare-Capacity Assignment in Survivable Networks”, IEEE/ACM Trans. On Networking, Vol. 3. Pp. 775-784, 1995.

[13] Horowitz E., Sahni, S., “Fundamentals of Computer Algorithms”, Computer Science Press, Inc., USA, 1978.

[14] V. Jimınez, A. Marzal, "Computing the K Shortest Paths: a New Algorithm and an Experimental Comparison" by. In ``Algorithm Engineering", J. S. Vitter and C. D. Zaroliagis (eds.), Springer-Verlag, Lecture Notes in Computer Science series, vol. 1668, pages 15--29, 1999. © Springer-Verlag. (Proceedings of the 3rd Int. Workshop on Algorithm Engineering, WAE'99, London, July 1999.

[15] V. Jimınez, A. Marzal, J. Monm, "A Comparison of Two Exact Algorithms for Finding the N-Best Sentence Hypotheses in Continuous Speech Recognition", Proc. 4th European Conference on Speech Communication and Technology, EUROSPEECH-95, pp. 1071--1074, Madrid, 1995.

[16] V. Jimınez, A. Marzal ,"An Algorithm for Efficient Computation of K Shortest Paths" DSIC-II/38/94, Universidad Politıcnica de Valencia, Spain, 1994. V. Jimınez, A. Marzal, "Advances in Pattern Recognition", F. J. Ferri, J. M. Iρesta, A. Amin and P. Pudil (eds.), Springer-Verlag, Lecture Notes in Computer Science series, vol. 1876, pages 183-192, 2000. © Springer-Verlag. (Proc. of the Joint IAPR 8th Int. Workshop on Structural and Syntactic Pattern Recognition and 3rd Int. Workshop on Statistical Pattern Recognition, S+SSPR'2000, Alicante, Spain, August/September 2000.)

[17] K. R. Krishnan, R. D. Doverspike, C. D. Pack, “Unified Models of Survivability for Multi- Technology Networks”, Proceedings of 14th Intl. Teletraffic Cong., pp. 655-666, 1994.

[18] S-W. Lee, C-S. Wu, "A K-best Paths Algorithm for Highly Reliable Communication Networks", IEICE Transactions on Communications, p. 586-590, 4/99.

[19] Louca S., Pitsillides A., Samaras G., “On Network Survivability Algorithms Based onTrellis Graph Transformations”, Proceedings of ISCC ’99, Egypt, July 1999.

[20] A. Marzal, V. Jimınez, "A Comparative Study of Classical Algorithms and New Approaches for the Search of the K-best Paths in a Graph" by Andrıs Marzal and Vvctor Jimınez, technical report DSIC-II/28/92, Universidad Politıcnica de Valencia, Spain, 1992.

[21] D. Medhi, “A Unified Approach to Network Survivability for Teletraffic Networks: Models, Algorithms and Analysis”, IEEE Trans. On Communications, Vol. 42, pp. 534-548, 1994.

[22] L. Nederlof, K. Struyve, C. O´Shea, H. Misser, Y. Du, B. Tamayo, “ End-to-End Survivable Broadband Networks”, IEEE Communications Magazine, Vol. 33, No. 9, pp. 63-70, 1995.

[23] S.D. Nikolopoulos, A. Pitsillides, "Towards Network Survivability by Finding the K-best Paths through a Trellis Graph", International Conference on Telecommunications (ICT'96), Turkey, pp. 817-821, April 1996.

[24] S.D. Nikolopoulos, A. Pitsillides, D. Tipper, "Addressing Network Survivability by Finding the K-best Paths through a Trellis Graph", Proceedings of INFOCOM´97, Kobe, Japan, pp., April 1997.

[25] S.D. Nikolopoulos and G. Samaras, "Sub-optimal Approach to Track Detection for Real-time Systems", 21st Euromicro Conference on Design of Hardware and Software Systems, IEEE/CS, Como, Italy, pp. 98-107, Sept. 4-7, 1995.

[26] E. Oki, N. Yamanaka, “A recursive matrixcalculation method for disjoint path search with hop link number constraints”, IEICE Transactions Communications, vol. E78-B, no.5, pp. 769-774, May 1995.

[27] Y. Tanaka, F. Rui-xue, M. Akiyama, “Design method of highly reliable communication network by the use of the matrix calculation”, IEICE Transactions Communications, vol. J70-B, no.5, pp. 551-556, 1987.

[28] S.Z. Shaikh, "Span-disjoint paths for physical diversity in networks", IEEE Symposium on Computers and Communications, Alexandria, Egypt, pp. 127-133, June 27-29, 1995.

[29] Shibuya, T., “New Approaches to Flexible Alignment of Multiple Biological Sequences”, A Master's Thesis, Department of Information Science, UniversityofTokyo, 1997.

[30] J.W. Surballe, "Disjoint paths in a network", Networks 4, pp. 125-145, 1974.

[31] J.W. Surballe, R.E. Tarjan, "A quick method for finding shortest pairs of disjoint paths", Networks 14, pp. 325-336, 1984.

[32] J. K. Wolf, A. M: Viterbi, S.G. Dixon, “Finding the Best Set of K paths through a trellis with Applications to Multitarget Tracking”, IEEE Trans. AES 25, pp. 287-296, 1989.

[33] J. Tapolcai, P. Laborczi, P. -H. Ho, T. Cinkler, A. Recski, and H. T. Mouftah, “Algorithms for Asymmetrically Weighted Pair of Disjointed Paths in Survivable Networks”, Proceedings Third International Workshop on Design of Reliable Communication Networks (DRCN 2001), Budapest, Hungry, Oct. 2001, pp. 239-249.

[34] D. Xu, Y. Xiong and C. Qiao, “A New PROMISE Algorithm in Networks with Shared Risk Link Groups”, IEEE Globecom, San Francisco, CA, Dec. 2003

[35] J. Y. Yen, “Finding the k shortest loopless paths in a network”, Management Science, vol. 17, pp. 712-716, 1971.

[36] http://scholar.lib.vt.edu/theses/available/etd-o42499-225537/unrestricted/chap2.pdf

[37] Dijkstra's Shortest Path Algortithm Animation in Java: http://home.wlu.edu/~vermeerp/Classes/211w99/Demos/Dijkstra/graph.html

[38] Vickie R. Westmark, A Definition for Information System Survivability, Proceedings of the 37th Hawaii International Conference on System Sciences, 2004.

[39] J. Czyzyk, M. Mesnier, and J. Moré, The NEOS Server, which appeared in IEEE Journal on Computational Science and Engineering, 5 (1998), pages 68-75

[40] . Gropp and J. Moré, Optimization Environments and the NEOS Server, that appeared in Approximation Theory and Optimization, M. D. Buhmann and A. Iserles, eds., pages 167-182, Cambridge University Press, 1997

[41] E. Dolan, The NEOS Server 4.0 Administrative Guide, Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, May 2001

[42] A.Itai, Y. Perl, and Y. Shiloach, “The complexity of finding maximum disjoint paths with length constrains”, Networks, Vol 12, p.277-286, 1982

[43] R. H. Anderson, A. C. Hearn, and R. O. Hundley, “RAND Studies of Cyberspace Security Issues and the Concept of a U.S. Minimum Essential Information Infrastructure,” Proceedings of the 1997 Information Survivability Workshop,CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University, 1997.

[44] C. Li, S. T. McCormick, and D. Simchi-Levi, “Finding disjoint paths with different path costs: Complexity and algorithms,” Networks, vol. 22, pp. 653–667, 1992.

[45] Yu Liu, David Tipper, “Successive Survivable Routing for Node Failures” Proceeding of IEEE Global Communications Conference 2001

[46] Y. Liu, Spare capacity allocation method, analysis and algorithms, Ph.D. dissertation, School of Information Sciences, University of Pittsburgh, 2001.