14

Designing WAN Topologies Under Redundancy Constraints

Pablo Sartor Del Giudice and Franco Robledo Amoza
Engineering School - Universidad de la República
Uruguay

1. Introduction

The huge amount of data that can be transported by fiber lines when compared to former existing networks of telephone lines introduced many new challenges when it comes to the design of network topologies. Given the important costs incurred when deploying and then operating such lines and their unprecedented bandwith capacities, "tree-like" topologies are usually sufficient to provide the required information flow while having minimal costs. But such topologies are extremely vulnerable; the loss of one single fiber link (or even worst, the failure of a switching site) might split the entire network into two or more disconnected components. Therefore the problem of designing or expanding an existing fiber Wide Area Network (WAN) involves two antagonistic objectives. A certain level of redundancy is to be achieved to keep certain sites connected in case of eventual failures in components; while at the same time, it is desirable to lower as much as possible the costs associated with fiber deployment and operation, thus leading to the problem of choosing which of subset of the feasible links to deploy. Depending on the particular application, redundancy requirements can consider that switch sites could fail, or assume that these are fault-tolerant and that only the failure of fiber lines is possible. Graph Theory is a field of mathematics useful for designing networks and analyzing their properties. In particular, the problem known as "Generalized Steiner Problem" (GSP) is very suitable for modelling the mentioned antagonistic objectives. It has been shown to be a quite complex NP combinatorial complexity class problem, for which the use of heuristic algorithms is mandatory to solve real general cases with reasonable usage of computer resources. In this chapter it is shown how the GSP can be solved by applying combinatorial optimization metaheuristics both for the node-connected and the edge-connected versions. The underlying context is that a number of existing sites that we will call "fixed sites" are to be connected among themselves (making optional use of existing intermediate switch entities if convenient) through fiber lines whose deployment and operation involve specific costs that are to be minimized; while at the same time the amount of component failures to tolerate is a specific requirement for every pair of fixed sites. Suitable algorithms are proposed for generating low cost designs with reasonable use of computer resources and some of their properties are analyzed. Results of test involving real network topologies are presented showing that this approach generates optimal or near-optimal topologies. Finally limitations, conclusions and current research lines on these topics are presented.

2. Context and problem definition

In general, a typical WAN backbone network has a meshed topology, and its purpose is to allow efficient and reliable communication between the switch sites of the network that act as connection points for the local access networks (eventually incorporating other switch sites for efficiency purposes). The topological design of a WAN basically consists of finding a minimum cost topology which satisfies some additional requirements, generally chosen to improve the survivability of the network (that is, its capacity to resist the failures of some of its components). One way to do this is to specify a connectivity level, and to search for topologies which have at least this number of disjoint paths (either edge disjoint or node disjoint) between pairs of switch sites. In the most general case, the connectivity level can be fixed independently for each pair of switch sites (heterogeneous connectivity requirements). This problem can be modelled as a *Generalized Steiner Problem* (denoted by GSP) and it is an NP-Complete problem (Steiglitz et al., 1969; Winter, 1986; 1987). We present the formal definition of this problem later in this section. Some references in this area are (Agrawal et al., 1995; Baïou, 1996; Balakrishnan et al., 2004; Chopra, 1992; Goemans & Bertsimas, 1993; Grötschel et al., 1995; Ko & Monma, 1989; Robledo & Canale, 2009). Most of these works are either focused on the edge-disjoint flavor of the problem, or on the exploration of particular cases, for example, when it is required to have two disjoint paths between all pairs of distinguished switch sites, which is called the 2-survivability problem (Baïou, 1996). In (Kerivin & Mahjoub, 2005; Stoer, 1992), extensive surveys over high survivability models are introduced. We will denote by GSP-NC and GSP-EC the GSP versions with node-connectivity constraints and edge-connectivity constraints respectively. Topologies verifying edge-disjoint path connectivity constraints assure that the network can survive to failures in the connection lines; whereas node-disjoint path constraints assure that the network can survive to failures both in switch sites as well as in the connection lines.

Winter (Winter, 1985; 1986; 1987) demonstrated that the GSP can be solved in linear time if the network is series-parallel, outerplanar or a Halin graph. Here follows a summary of the survivability problems related to the GSP. Gröstchel, Monma and Stoer (Grötschel & Monma, 1990) consider a particular case of the GSP working on a slightly different context where different types of node exist, representing a hierarchy of fault-tolerance requirements; they called it the NCON problem. In (Stoer, 1992), Stoer gives an extensive survey for the NCON and the ECON (the version with edge-connectivity constraints), and some particular cases. In the NCON (resp. ECON) each node i has an associated nonnegative integer r_i, the **type** of i (the survivability requirement or "importance" of a node is modeled by node types). The GSP model generalizes the NCON(ECON) model since in the GSP there exist general survivability requirements r_{ij} that are specified for each pair i, j of fixed nodes independently. Nevertheless, Grötschel, Monma and Stoer (Grötschel et al., 1991; Grötschel et al., 1992a;b; 1995) introduce the use of node types to define survivability requirements based on the premise that these adequately express the relative importance placed on maintaining connectivity between offices and they classify the different problem types according to the largest occurring node type and according to whether the node types represent node or edge connectivity requirements. Let us note that there exist many specializations of the survivability problems which can be formulated by varying its parameters (the required amount of disjoint paths to connect pairs of sites, general, euclidean, uniform or other hipothesis about costs, etc). There exist polynomially solvable cases of the NCON and ECON problems. They result from relaxing the original problem with restrictions like uniform costs, 0/1 costs, restricted node

types, and special underlying graphs such as outerplanar, series-parallel, and Halin graphs. All these particular cases are referenced and briefly exposed in (Stoer, 1992). On the other hand, lower bounds and heuristics with worst-case guarantees for kECON[1] problems were found for restricted costs, e.g., uniform costs or costs satisfying the triangle inequality, as well as very important results on the structure of optimal survivable networks for this cost structure. Details of these works can be seen in (Bienstock et al., 1990; Cheriyan et al., 2001; Chou & Frank, 1970; Frank & Chou, 1970; Frederickson & Jàjà, 1982; Goemans & Bertsimas, 1993; Goemans & Williamson, 1992; Monma et al., 1990) and in a summarized form in (Stoer, 1992). Unfortunately, there exist few exact algorithms for the NCON and ECON for general costs. Christofides and Whitlock (Christofides & Whitlock, 1981) introduce a cutting plane algorithm together with branch-and-bound for ECON problems where the connection levels are specified for each pair of nodes. Chopra and Gorres (Chopra, 1992) give a cutting plane algorithm mixed with branch-and-bound for solving 2ECON problems.

In the literature there are several works related to approximation algorithms for the GSP and different particular cases. Next, we will introduce a survey of the main existing algorithms based on this approach. In (Ravi & Klein, 1993) the authors show how to obtain approximately optimal solutions to 2-edge-connected versions of the problems addressed in (Goemans & Williamson, 1992). Subsequent papers (Gabow et al., 1993; Goemans et al., 1994; Williamson et al., 1995) extended these methods to give approximation algorithms for the GSP-EC without link duplication. Agrawal, Klein and Ravi (Agrawal et al., 1995) developed an algorithm for the GSP-EC with performance guarantee of $2\lceil \log_2(r_{max}+1)\rceil$, where r_{max} is the highest requirement value. More recently Jain (Jain, 2001) presented a factor 2 approximation algorithm for the GSP-EC. Kortsarz, Krauthgamer and Lee (Kortsarz et al., 2004) introduced the first strong lower bound on the approximability of the GSP when there are no Steiner nodes (i.e. all sites are fixed). An important special case of the GSP occurs when we are searching the minimum-cost k-node-connected subgraph spanning all the nodes. In first place, let us see the general case. In (Cheriyan et al., 2001; 2002; Czumaj & Lingas, 1999; Kortsarz et al., 2004; Kortsarz & Nutov, 2003; Ravi & Williamson, 1997; 2002) the authors propose several approximation algorithms for the problem of finding a minimum-cost k-node-connected spanning subgraph, besides they give their respective approximation ratios. For $k \leq 7$ an approximation ratio of $\lceil (k+1)/2 \rceil$ is known; see (Khuller & Raghavachari, 1996) for $k = 2$, (Auletta et al., 1999) for $k = 2,3$, (Jain, 1999) for $k = 4,5$, and (Kortsarz & Nutov, 2003) for $k = 6,7$. Other approximations for $k = 2$ can be seen in (Böckenhauser et al., 2002; Csaba et al., 2002). Furthermore, in (Czumaj & Lingas, 1999), (Cheriyan & Thurimella, 2000) and (Kortsarz & Nutov, 2003) the authors respectively supply approximation algorithms for the following special cases: the graph has complete Euclidean topology, uniform costs, and metric costs (i.e. when the costs satisfy the triangle inequality).

Finally, let us see works related to the particular case named "Steiner two-node-survivable network problem", (denoted by STNSNP). In (Baïou, 1996) the author mentions different problems related directly to the STNSNP. In particular, the problems known as the Steiner 2-edge-connected subgraph problem (STECSP), the Steiner 2-node-connected subgraph problem (STNCSP) and the Steiner 2-edge-survivable network problem (STESNP). The STNSNP (resp. STESNP) also corresponds to the problem kNCON (resp. kECON) in the case where all nodes have a connectivity level requirement belonging to $\{0,2\}$. Given a graph $N = (X,U)$, a subset $T \subseteq X$ and a matrix C of connection costs associated to U;

[1] ECON problems where there are at least two nodes with connectivity requirement k.

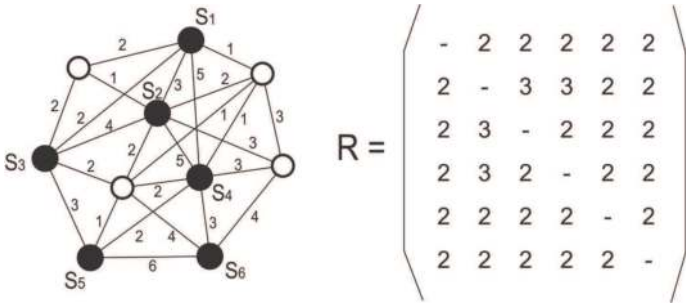

Fig. 1. Example instance for the GSP

the objective in the STNCSP (resp. STECSP) is to find a minimum-cost 2-node-connected (resp. 2-edge-connected) subgraph spanning the set of nodes T. If the matrix C is positive, the sets of optimal solutions associated to the STNSNP and STNCSP are equal. Idem the sets of optimal solutions associated to the STESNP and STECSP. If all the nodes are fixed (there are no Steiner nodes) the problems STESNP and STECSP coincide, and also the STNSNP with the STNCSP. Moreover, it is easy to see that all feasible solution of the STNCSP (resp. STECSP) is also feasible for the STNSNP (resp. STESNP). In (Coullard et al., 1991) the authors developed a linear algorithm to solve the STNCSP in the case of graphs without W_4 (a wheel graph with four nodes) and Halin graphs. The authors of this chapter have previously developed a parallel method (of worst case exponential complexity) for the general case (Cancela et al., 2005). Other works related to particular cases of the STNCSP, e.g. when $T = X$ or uniform costs, already have been mentioned above.

2.1 Problem formalization and definitions

We will formalize our optimal network design problem by using the following notation:

- $G = (V, E, C)$: Simple undirected graph with weighted edges, modelling feasible links;
- V : Nodes of G, representing fixed sites and intermediate optional sites to connect;
- E : Edges of G, representing feasible links between nodes;
- $C : E \to \mathbb{R}^+$: Edge weights, representing the cost of deploying and operating each link;
- $T \subseteq V$: Terminal nodes (representing the set of fixed sites, i.e. the ones that have non-zero connectivity requirements with at least one other node);
- $R : R \in \mathbb{Z}^{|T|\times|T|}$: Symmetrical integer matrix of connectivity requirements; $r_{ij} = r_{ji} \geq 0, \forall i, j \in T; r_{ii} = 0, \forall i \in T$.

We will model our design problem as a Generalized Steiner Problem (GSP) whose definition is as follows.

Definition 2.1. ***GSP****. Given the graph G with edge weights C, the teminals set T and the connectivity requirements matrix R, the objective is to find a minimum cost subgraph $G_T = (V_T, E_T, C_T)$ of G where C_T is the restriction of C to the subset T and every pair of terminals i, j is connected by r_{ij} disjoint paths.*

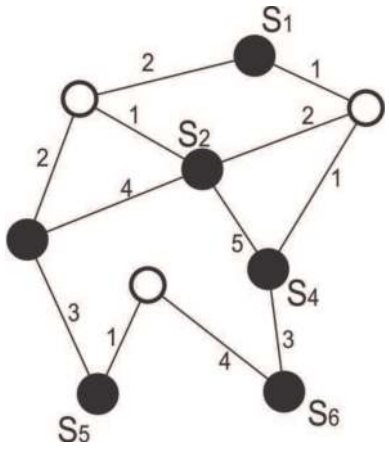

Fig. 2. Solution for the GSP instance

Two different versions of the problem arise depending on the way "disjoint" is interpreted above. If it refers to node-disjoint paths we will denote it as GSP-NC (node-connected); if it refers strictly to edge-disjoint paths (allowing to share nodes) then we will denote the problem as GSP-EC (edge-connected). This versions will allow us to model situations in which only link-failure tolerance is required (GSP-EC) or situations in which site-failure tolerance is required (GSP-NC). An example instance of the GSP is shown in Figure 1. There are six fixed switch sites, colored black and labeled S1, S2, S3, S4, S5 and S6, and four non-fixed switch sites, colored white. The connections that can be potentially deployed are shown in the figure, annotated with their costs. The matrix R shows the connectivity requirements among the fixed sites, ranging in this case from 2 to 3. Figure 2 shows a solution of this instance having cost 29; note that only three of the four non-fixed sites were used. Due to the enormous intrinsic complexity of the GSP, exact algorithms to solve it (i.e. that guarantee that optimal solutions are built) can only be applied under specific circumstances and/or on small instances (a few sites); it is known to be an NP complexity class combinatory problem. Therefore, to deal with real general problems, the use of heuristic algorithms conceived to generate good quality solutions within reasonable time and use of computing power resources turns to be mandatory.

2.2 The GRASP metaheuristic

GRASP (Greedy Randomized Adaptive Search Procedure) is a metaheuristic which proved to perform very well for a variety of combinatorial optimization problems; we will make use of it to solve the GSP. A GRASP is a "multistart local optimization" procedure which performs two consecutive phases in each iteration:

- Construction Phase: it builds a feasible solution that chooses (following some randomized criterion) which elements to add from a list of candidates defined with some greedy approach;

Procedure GRASP$(MetaParams, MaxIter, RndSeed)$

```
1: bestSol ← NIL
2: for k = 1 to MaxIter do
3:    greedySol ← ConstPhase(MetaParams, RndSeed)
4:    localSearchSol ← LocalSearchPhase(greedySol)
5:    if cost(localSearchSol) < cost(bestSol) then
6:       bestSol ← localSearchSol
7:    end if
8: end for
9: return bestSol
```

Fig. 3. GRASP pseudo-code

- Local Search Phase: it explores the neigborhood[2] of the feasible solution delivered by the Construction Phase to reach a local optimum.

Figure 3 presents a generic GRASP pseudo-code. The procedure inputs include metaparameters *MetaParams* which set the size of the list of candidates and other behaviour of the *ConstPhase* procedure; the amount of iterations to run *MaxIter*; and a seed for random number generation. After having run *MaxIter* iterations the procedure returns the best solution found. Details of this metaheuristic can be found in (Resende & Ribeiro, 2003). In the next sections we introduce algorithms for implementing the Construction and Local Search Phases suitable to solve the GSP-EC (edge-connected version) as well as comment any changes necessary for adapting them also to the GSP-NC problem.

2.3 Construction phase algorithm

Our construction phase algorithm proceeds by building a graph which satisfies the requirements of the matrix R; it starts with an edgeless graph and in each iteration one new path is added to the solution under construction. The algorithm is shown in Figure 4. It takes as inputs the graph G of feasible edges, the edge costs C, the set of terminal nodes T and the matrix of requeriments R. In line 1 we initialize the solution graph under construction G_{sol} with the nodes of T and no edges; the matrix $M = (m_{ij})_{i,j \in T}$ which records the amount of connection requirements not yet satisfied in G_{sol} between the terminal nodes i and j; the sets P_{ij} that will be used to record the r_{ij} disjoint paths found for connecting the nodes i, j; and an auxiliary matrix $A = \{A_{ij}\}$ used to record how many times it was impossible to find one more path between two terminal nodes i, j whose requirements r_{ij} were not yet covered. In line 2 we alter the costs of the matrix C in order to make the algorithm satisfy a property that we describe below (together with the altering function used) and introduce random.

Loop 3-15 is repeated until all terminal nodes have their connectivity requirements satisfied, or until for a certain pair of terminals i, j, the algorithm fails to find a path a certain number of times MAX_ATTEMPT. Each iteration works the following way. Line 4 selects two terminal nodes i, j at random for which there are pending connectivity requirements. Line 5 computes the graph obtained by removing from G the edges of all paths already computed to connect i and j; thus, any path computed in G' will be edge-disjoint from the former $(i...j)$ paths in P_{ij}. In the case of the GSP-NC, not only the edges should be supressed but also the

[2] Set of solutions that can be obtained by well-defined replacement of parts of the current solution

Procedure ConstPhase(G, C, T, R)

1: $G_{sol} \leftarrow (T, \emptyset); m_{ij} \leftarrow r_{ij} \forall i, j \in T; P_{ij} \leftarrow \emptyset \forall i, j \in T; A_{ij} \leftarrow 0 \forall i, j \in T$
2: $C \leftarrow$alter-costs(C)
3: **while** $\exists m_{ij} > 0 : A_{ij} <$MAX_ATTEMPT **do**
4: let i, j be any two terminals with $m_{ij} > 0$
5: $G' \leftarrow G \setminus P_{ij}$
6: let $C' = (c'_{uv}) : c'_{uv} \leftarrow$ [0 if $(u, v) \in G_{sol}; c_{uv}$ otherwise]
7: $p \leftarrow$ shortest-path(G', C', i, j)
8: **if** $\nexists p$ **then**
9: $A_{ij} \leftarrow A_{ij} + 1; P_{ij} \leftarrow \emptyset; m_{ij} \leftarrow r_{ij}$
10: **else**
11: $G_{sol} \leftarrow G_{sol} \cup \{p\}$
12: $P_{ij} \leftarrow P_{ij} \cup \{p\}; m_{ij} \leftarrow m_{ij} - 1$
13: $[P, M] \leftarrow$ general-update-matrix(G_{sol}, P, M, p, i, j)
14: **end if**
15: **end while**
16: **return** G_{sol}, P

Fig. 4. ConstPhase pseudo-code

nodes of the former $(i...j)$ paths in order to generate node-disjoint paths. In line 6, the edges already present in the solution under construction are given cost 0; by doing this, they will be taken as costless when considering the cost of any new path, enabling edge-reusing among different pairs of terminals. Line 7 computes the shortest path (regarding costs) connecting i and j, considering as feasible the edges from G' and with costs given by C'. In case this turns to be impossible, this is acknowledged in line 9 by incrementing the counter A_{ij} and resetting the path set P_{ij}, hoping that computing a different sucession of paths for i, j allow to satisfy the r_{ij} requirements. In case a path p was found, it becomes part of the solution under construction (lines 11-12), and the general-update-matrix procedure on line 13 updates the pending connection requirements of the matrix M, by applying the Ford-Fulkerson's algorithm with all capacities equal to 1, to detect if the adoption of the new path turned to satisfy other requirements besides the one for the pair i, j. Finally, the algorithm ends by returning the feasible solution G_{sol} together with the path set P which "certifies" that all requirements R were satisfied.

2.3.1 Altering costs

The algorithm here proposed satisfies the property given below, provided an appropriate function alter-costs is used in line 2 (unlike similar construction phases previously proposed for the GSP-NC in (Robledo & Canale, 2009) as certain trivial instances can attest):

$$\lim_{iterations \to \infty} probability(get\ an\ optimal\ solution) = 1$$

In other words, we can guarantee that any desired level of certainty of getting an optimal solution can be reached provided as many iterations as needed are run.

We proved that this property is verified if the alter-costs function is such that all edges have their costs altered independtly from the others and the altered costs take values in $(0, +\infty)$

with any probability distribution that assigns non-zero probabilities to any open subinterval of $(0, +\infty)$. In our tests we used an exponential distribution with parameter $1/real\_cost$.

Moreover, by altering costs the proposed algorithm proceeds by just computing the shortest path in its main loop, instead of computing a set of "simultaneous disjoint shortest paths" and then randomly choosing one (as in previous algorithms), thus involving less computing.

2.4 Local search phase algorithms

The local search phase starts with a feasible solution obtained from the construction phase and proceeds by consecutively moving to neighbour solutions which reduce the cost of the solution graph until it reaches a local optimum. Any local search algorithm needs a precise definition of the neighbourhood concept; we propose two different ones, which we chain inside our suggested LocalSearchPhase algorithm. They are defined in terms of a certain structural decomposition of graphs that we define below together with some other auxiliary definitions.

Definition 2.2. key-node: *Given a GSP-EC instance and a feasible solution G_{sol}, we define a **key-node** as a non-terminal node with degree at least three in G_{sol}.*

Definition 2.3. key-path: *Given a GSP-EC instance and a feasible solution G_{sol}, we define a **key-path** as a path in G_{sol} such that all intermediate nodes are non-terminal with degree two in G_{sol} and whose endpoints are either terminal nodes or key-nodes.*

Definition 2.4. key-tree: *Given a GSP-NC instance, a feasible solution G_{sol} and a key-node v of G_{sol}, we define as the **key-tree** associated to v the subgraph of G_{sol} obtained through the union of all key-paths with v as an endpoint.*

Definition 2.5. key-star: *Given a GSP-EC instance, a feasible solution G_{sol} and any node v of G_{sol}, we define as the **key-star** associated to v the subgraph of G_{sol} obtained through the union of all key-paths with v as an endpoint.*

2.4.1 Path-based local search neighbourhood

Our first neighbourhood is based on the replacement of any key-path k by another key-path with the same endpoints, built with any edge from the feasible connections graph G (even some of G_{sol}), provided no connectivity levels are lost when reusing edges. Let k be a key-path of a certain solution G_{sol} and P a set of paths which "certificates" its feasibility (as the one returned by ConstPhase). We will denote by $J_k(G_{sol})$ the set of paths $\{p \in G_{sol} : k \subseteq p\}$. These are the paths which contain the key-path k. We will also denote by $\chi_k(G_{sol})$ the edge set

$$\chi_k(G_{sol}) = \bigcup_{q=i\ldots j \in J_k(G_{sol})} E(P_{ij} \setminus q)$$

These are the edges that, if used to replace the key-path k in P (obtaining a path set P'), would turn to be shared by some paths from G_{sol} with the same endpoints, thus invalidating the resulting set P' as a feasibility certificate. We can now define our first neighbourhood.

Definition 2.6. Neighbourhood1*: Given a GSP-EC instance and a feasible solution G_{sol}, it is the set of all graphs obtained by replacing any key-path k of G_{sol} by another path p such that $cost(p) < cost(k)$ and the edges of p are chosen from the set $E \setminus \chi_k(G_{sol})$ and/or k. (Recall that E represents the feasible edges between nodes).*

Procedure LocalSearchPhase1(G, C, T, S)

1: $improve \leftarrow$ TRUE
2: $\kappa \leftarrow$ k-decompose(S)
3: **while** *improve* **do**
4: $improve \leftarrow$ FALSE
5: **for all** kpath $k \in \kappa$ with endpoints u, v **do**
6: $G' \leftarrow$ the subgraph induced from G by $E(k) \cup (E \setminus \chi_k(S))$
7: $C' \leftarrow (c'_{ij})/c'_{ij} = 0$ if $(i,j) \in S \setminus k; c'_{ij} = c_{ij}$ otherwise
8: $k' \leftarrow$ shortest-path(G', C', u, v)
9: **if** cost$(k', C') <$ cost(k, C') **then**
10: $improve \leftarrow$ TRUE
11: update $S : \forall p \in J_k(S)(p \leftarrow (p \setminus k) \cup k')$
12: **if** $\exists z \in V(k'), z \notin \{u, v\}$, degree$(z) \geq 3$ in S **then**
13: remove-cycles$(J_k(S))$
14: $\kappa \leftarrow$ k-decompose(S)
15: **else**
16: $\kappa \leftarrow \kappa \setminus \{k\} \cup \{k'\}$
17: **end if**
18: **end if**
19: **end for**
20: **end while**
21: **return** S

Fig. 5. LocalSearchPhase1 pseudo-code

Based on these definitions we built the path-based local search algorithm LocalSearchPhase1 shown in Figure 5. The algorithm receives as inputs the graph G of feasible connections, the edge cost matrix C, the terminals set T and a path set S which build up a feasible solution. Line 1 initializes the flag *improve* which indicates wheter an improved solution has been found or not. Line 2 computes the decomposition in key-nodes and key-paths of the set S. Loop 3-20 looks for succesive cost improvements until no more can be done. Each iteration proceeds as follows. The loop 5-19 analyzes each key-path k trying to find a suitable replacement with lower cost. Line 6 computes the edge set $E(k) \cup (E \setminus \chi_k(S))$, where edges are to be chosen from to build the replacing key-path. This set is such that, as seen above, ensures no loss of connectivity levels in the new solution obtained, while allowing the reuse of edges already present in the current solution S. Line 7 computes a new cost matrix C', zeroing the cost of all edges of S that are not included in the key-path k, to reflect the fact that using any of those edges to build the replacing key-path adds no extra cost to the modified solution. Line 8 computes the path with lower cost according to the matrix C' over the subgraph computed in line 6. Line 9 verifies if the adoption of the new key-path implies a cost reduction. If so, it is acknowledged by the flag *improve* and k is replaced in all paths of S which included k. Care is taken to remove cycles and recompute the k-decomposition if a certain node happens to have a degree greater than two after the replacement (lines 12-14); if the latter does not happen, line 16 simply updates the k-decomposition by replacing the key-path (thus avoiding computing a new k-decomposition from scratch). After exiting the main loop, line 21 returns a feasible solution whose cost can no more be reduced by moving to neighbour solutions. As we commented in the Construction Phase, for the GSP-NC problem a small change in the

Procedure LocalSearchPhase2(G, C, T, S)

```
improve ← TRUE
κ ← k-decompose(S)
while improve do
    improve ← FALSE
    for all kstar k ∈ κ do
        [k', newCost] ← BestKeyStar(G, C, T, S, k)
        if newCost < cost(k, C) then
            improve ← TRUE
            replace k by k' in all paths from S
            κ ← k-decompose(S)
            abort for all
        end if
    end for
end while
return S
```

Fig. 6. LocalSearchPhase2 pseudo-code

definition of J_k and χ_k must be made, supressing also the nodes involved rather than only the edges.

2.4.2 Key-star-based local search neighbourhood

Our second neighbourhood is based on the replacement of key-stars, which frequently allow to improve feasible solutions that are locally optimal when only considering Neighbourhood1. In the case of the GSP-NC, as no node sharing is allowed among disjoint paths, all key-stars are trees (named *key-trees*); a key-tree replacement neighbourhood for the GSP-NC can be found in (Robledo & Canale, 2009). Due to the possibilty of sharing nodes among edge-disjoint paths, when working with GSP-EC problems, we have to work with key-stars, and unlike (Robledo & Canale, 2009) we will allow the root node to be a terminal node in order to get a broader neighbourhood. In the GSP-NC any key-tree can be replaced by any tree with the same leaves with no loss of connectivity levels. In the GSP-EC, if the replacing structure is also a key-star the same holds true; but it does not for other general structures (non-star trees included). We propose an algorithm that given a key-star k, deterministically seeks for the lowest cost replacing key-star k' able to "repair" the paths from P broken when removing the edges of k.

Let k be a key-star in a certain feasible solution G_{sol} and P a set of paths which "certificates" its feasibility (as the one returned by ConstPhase). For allowing as much reusing of edges as possible, we can extend our previous definition of $J_k(G_{sol})$ and $\chi_k(G_{sol})$ to consider key-stars k instead of key-paths; and thus we can define the key-star based neighbourhood as follows.

Definition 2.7. Neighbourhood2*: Given a GSP-EC instance and a feasible solution G_{sol}, it is the set of all graphs obtained by replacing any key-star k of G_{sol} by the lowest possible cost key-star k' such that k' preserves the same connectivity among the leaves of k and its terminal nodes, and the edges of k' are chosen from the set $E \setminus \chi_k(G_{sol})$ and/or k.*

We present the star-based local search algorithm LocalSearchPhase2 in Figure 6. The algorithm receives as inputs the graph G of feasible connections, the edge cost matrix C, the

Procedure BestKeyStar(G, C, T, S, k)

1: $G' \leftarrow$ the subgraph induced from G by $E(k) \cup (E \setminus \chi_k(S))$
2: $C' \leftarrow (c'_{ij})/c'_{ij} = 0$ if $(i,j) \in S \setminus k; c'_{ij} = c_{ij}$ otherwise
3: add a "virtual node" w to G'
4: $\Omega \leftarrow \psi_k$
5: **if** $\theta_k \in T$ **then**
6: $\quad \Omega \leftarrow \Omega \cup \{\theta_k\}$
7: **end if**
8: **for all** $m \in \Omega$ **do**
9: $\quad$ add $\hat{\delta}_{k,m}$ parallel edges (w, m) to G' with cost 0
10: **end for**
11: $c_{min} \leftarrow 0; k_{min} \leftarrow k$
12: **for all** $z \in V(G)$ **do**
13: $\quad k' \leftarrow$ simult-shortest-paths$(G', \delta_{G,w}, z, w)$
14: $\quad$ **if** k' has $\delta_{G,w}$ paths $\wedge$ $cost(k', C') < c_{min}$ **then**
15: $\quad\quad c_{min} \leftarrow cost(k', C'); k_{min} \leftarrow k'$
16: $\quad$ **end if**
17: **end for**
18: **return** $[k_{min}, c_{min}]$

Fig. 7. BestKeyStar pseudo-code

terminals set T and a path set S which build up a feasible solution. Line 1 initializes the flag *improve* which indicates wheter an improved solution has been found or not. Line 2 computes the decomposition in key-nodes and key-paths of the set S. Loop 3-14 looks for succesive cost improvements until no more can be done. Each iteration proceeds as follows. The loop 5-13 analyzes each key-star k trying to find a suitable replacement with lower cost. Line 6 determines the lowest cost key-star k' that could replace k and its cost (computed assuming that edges from the current solution not in k have no cost to promote edge reusing). To do so it uses the procedure BestKeyStar described later. Line 7 verifies if the replacing key-star has lower cost than k; if it does, lines 8-11 acknowledge the fact, the replacement is done over the set S, the k-decomposition is recomputed and the "for all" loop is aborted to restart looking for improvements. After exiting the main loop, line 15 returns a feasible solution whose cost can no more be reduced by moving to neighbour solutions.

Figure 7 presents the algorithm BestKeyStar. Given a key-star k we denote by θ_k its root node; by ψ_k the set of its leaf nodes; and by $\hat{\delta}_{k,m}$ (being m the root node of k or one of its leaves) the highest amount of key-paths that join m in k with any other node that is root or leaf in k. The algorithm is based on the idea of building key-stars by employing the simult-shortest-paths algorithm (that can be found in (Bhandari, 1997) as *k-shortest-path*). Lines 1-2 compute the subgraph of G obtained by removing the edges that could cause loss of connectivity level if reused; the altered cost matrix C' with cost zero for reused edges; and adds a virtual node w whose purpose is explained below. Lines 4-7 determine the set of leaf nodes that the key-star to build must have. Lines 8-10 connect each of the latter to w with an appropriate number of parallel zero-cost edges totalling $\delta_{G,w}$ (degree of w in G) edges. The loop 12-17 considers nodes of G that could be potencial roots z of the key-star to be found, and then builds the lowest-cost one with root node z through the application of the simult-shortest-path algorithm in line 13; $\delta_{G,w}$ edge-disjoint paths connecting z and w are requested. If found (lines 14-16) and with

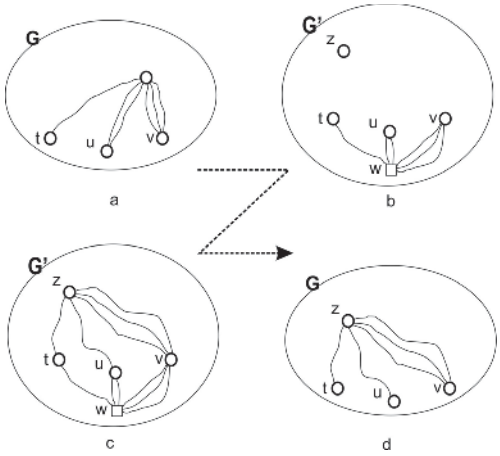

Fig. 8. Computing the best key-star

lower cost than k then the new key-star and its associated cost are recorded as the best ones so far found. After having considered all possible root nodes, line 18 returns both the best key-star and its cost according to C'.

Figure 8 depicts the process of determining which the best key-star to replace a given one is. It illustrates (a) the feasible graph G with a key-star that keeps connected the leaf nodes t, u, v; (b) the graph G' obtained after adding the virtual nodes w linked with no cost to t, u, v by the appropriate amount of edges and choosing a "candidate" root node z; (c) the shortest paths found to connect z and w; and (d) the new key-star obtained after removing the virtual node w.

2.5 GRASP algorithm description

Now we are able to put the pieces together and build a GRASP algorithm for solving the GSP-EC. Figure 9 shows the resulting pseudo-code. Basically the local search phase of this algorithm applies key-path replacement based movements until no further improvements are possible; then it tries to apply the best key-star replacement movement (once); if the latter is done with a cost reduction, then key-path replacements are tried again, and so on, until no further improvements are possible for both kinds of movements.

The algorithm receives as inputs the graph G of feasible connections, the cost matrix C, the terminals T, the redundancy requirements matrix R and a the number of iterations *iters* to perform. In line 1 the minimum cost found c_{min} is initialized to ∞ and an empty path set S_{opt} is initialized. The main loop (2-18) is executed *iters* times and then the best solution found is returned. Line 3 builds a feasible solution employing our ConstPhase greedy randomized adaptive algorithm; being S the path set that certifies feasibility. If the set S has less paths than the so far found best solution S_{opt} (line 4) this iteration is discarded. This could happen if the last call to ConstPhase was not able to satisfy all requirements of R and a previous call was able to do it (or at least to satisfy a greater number); our first objective is to satisfy as many requirements of R as possible. Line 6 applies the key-path based movements by calling LocalSearchPhase1. If this was the first local search or if a cost reduction was achieved then

Procedure GRASP_GSP$(G, C, T, R, iters)$

```
 1: c_min ← ∞; S_opt ← ∅
 2: for i = 1 to iters do
 3:    [G_sol, S] ←ConstPhase(G, C, T, R)
 4:    if |S| ≥ |S_opt| then
 5:       flag ←TRUE
 6:       OptLoop: [G_sol, S'] ←LocalSearchPhase1(G, C, S)
 7:       if flag ∨ cost(S') <cost(S) then
 8:          flag ←FALSE
 9:          [G_sol, S''] ←LocalSearchPhase2(G, C, S')
10:          if cost(S'', C) <cost(S', C) then
11:             S ← S''; go to OptLoop
12:          end if
13:       end if
14:       if cost(S', C') < c_min then
15:          c_min ←cost(S'', C); S_opt ← S
16:       end if
17:    end if
18: end for
19: return S_opt
```

Fig. 9. GRASP_GSP pseudo-code

a best key-star movement is tried in line 9. In case the latter succeeds in reducing the cost (verified in line 10), the execution flow resumes at line 6, for trying a new cycle of chained improvements. When no further local improvements are possible, lines 14-16 update the best known solution in case an improvement was achieved.

3. Performance tests

This section presents the results obtained after testing our algorithms with twenty-one test cases. The algorithms were implemented in C/C++ and tested on a 2 GB RAM, Intel Core 2 Duo, 2.0 GHz machine running Microsoft Windows Vista. All instances were run with the parameter *iters* set to 100.

3.1 Test set description

To our best knowledge, no library containing benchmark instances related to the GSP-NC nor GSP-EC exists; we have built a set of twenty-one test cases that are based in cases found in the following public libraries:

- steinlib (Koch et al., 2000): instances of the Steiner problem; in many cases the optimal solution is known, in others the best solution known is available;
- tsplib (Reinelt, 2004): instances of diverse graph theory related problems, including a "Traveling Salesman Problem" section.

The main characteristics of the twenty-one test cases are shown in Table 1.. For each case we show the amount of nodes (V), feasible edges (E), terminal nodes (T) and Steiner (non terminal) nodes (St). We also show the level of edge-connectivity requirements (one, two,

Case	V	E	T	St	Redund.	Opt
b01-r1	50	63	9	41	1-EC	82
b01-r2	50	63	9	41	2-EC	NA
b03-r1	50	63	25	25	1-EC	138
b03-r2	50	63	25	25	2-EC	NA
b05-r1	50	100	13	37	1-EC	61
b05-r2	50	100	13	37	2-EC	NA
b11-r1	75	150	19	56	1-EC	88
b11-r2	75	150	19	56	2-EC	NA
b17-r1	100	200	25	75	1-EC	131
b17-r2	100	200	25	75	2-EC	NA
cc3-4p-r1	64	288	8	56	1-EC	2338
cc3-4p-r3	64	288	8	56	3-EC	NA
cc6-2p-r1	64	192	12	52	1-EC	3271
cc6-2p-r2	64	192	12	52	2-EC	NA
cc6-2p-r123	64	192	12	52	1,2,3-EC	NA
hc-6p-r1	64	192	32	32	1-EC	4003
hc-6p-r2	64	192	32	32	2-EC	NA
hc-6p-r123	64	192	32	32	1,2,3-EC	NA
bayg29-r2	29	406	11	18	2-EC	NA
bayg29-r3	29	406	11	18	3-EC	NA
att48-r2	48	300	10	38	2-EC	NA

Table 1. Characteristics of the Test Cases

three or mixed) and the optimal costs when available. GSP poblems solved with connectivity level one are Steiner problems and in those cases we got the optimal solution cost from steinlib. Problems b01, b03, b05, b11 and b17 were taken from steinlib's problem instances set "B" and are cases randomically generated with integer uniform costs ranging from 1 to 10. The case cc3-4p belongs to steinlib's instance set "PUC"; eigth terminal nodes are terminal and we solved two instances with uniform connectivity requirements one and three. The cases cc6-2p and hc-6p belong also to steinlib's instance set "PUC"; twelve and thirty-two nodes are terminal and we solved three instances for each one with connectivity requirements one, two, and a mix of one to three. Finally the cases bayg29 and att48 were taken from the library tsplib; both correspond to real cases (twenty-nine cities from Bavaria, Germany; and 48 cities from USA).

3.2 Numerical results

Computational results of the tests can be found in Table 2. Here follows the meaning of each column:

- Reqs.: total amount of requirements satisfied by the best solution found
- t(ms): the average running time (in ms) per iteration
- Cost: the cost of the best solution found

Case	Reqs.	t(ms)	Cost	%LSI
b01-r1	36	77	82	3.0
b01-r2	42	80	98	3.4
b03-r1	300	2611	138	10.6
b03-r2	378	3108	188	4.1
b05-r1	78	298	61	9.2
b05-r2	144	1389	120	5.2
b11-r1	171	1477	88	13.8
b11-r2	324	4901	180	3.4
b17-r1	300	6214	131	10.2
b17-r2	531	15143	244	3.0
cc3-4p-r1	28	388	2338	10.0
cc3-4p-r3	84	2221	5991	4.6
cc6-2p-r1	66	2971	3271	2.4
cc6-2p-r2	132	4801	5962	10.2
cc6-2p-r123	140	6317	8422	9.8
hc-6p-r1	496	25314	4033	6.8
hc-6p-r2	992	28442	6652	3.5
hc-6p-r123	957	26551	7930	5.2
bayg29-r2	110	975	6856.88	4.6
bayg29-r3	165	2413	11722	4.2
att48-r2	90	1313	23214	13.0
Averages	265	6524	-	6.7

Table 2. Numerical Test Results

- LSI: "local search improvement" - the percentage of cost improvement achieved by the local search phase when compared to the cost of the solution delivered by the construction phase, for the best solution found

In all cases with connectivity requirements equal to one (1-EC) for all pairs of terminals (for which the optimal costs are known) every best solution found is optimal, with the exception of the case hc-6p-r1 (found cost 4033 being the optimal cost 4003). Note also that the average cost improvement over the solution delivered by ConstPhase (LSI) amounts to 6.7% (when computed only for the best solutions found). All solutions found are edge-minimal regarding feasibility (no edge can be supressed without losing required connectivity levels); and in all cases the maximum possible number of requirements are satisfied (i.e. for all pairs of terminals i, j, whether their requirement r_{ij} was satisfied or f_{ij} disjoint paths were found being f_{ij} the maximum achievable amount of disjoint paths joining i and j given by the topology of the feasible connections graph G). Figure 10 show the best 3-connected network found to connect the sites of bayg29, as well as the best 2-connected network found to connect the sites of att48.

4. Current research

4.1 Relationship among the GSP-NC and GSP-EC problems

There is a strong relationship among the GSP-NC and the GSP-EC problems. We have demonstrated that any GSP-EC instance can be transformed in polinomial time into a GSP-NC

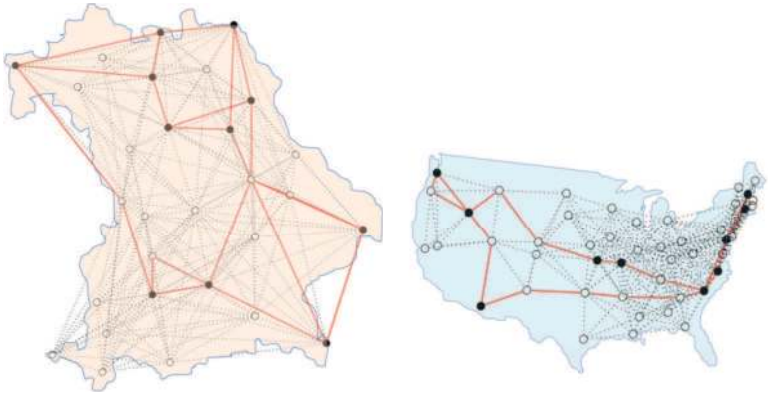

Fig. 10. Best Solutions for cases bayg29 and att48

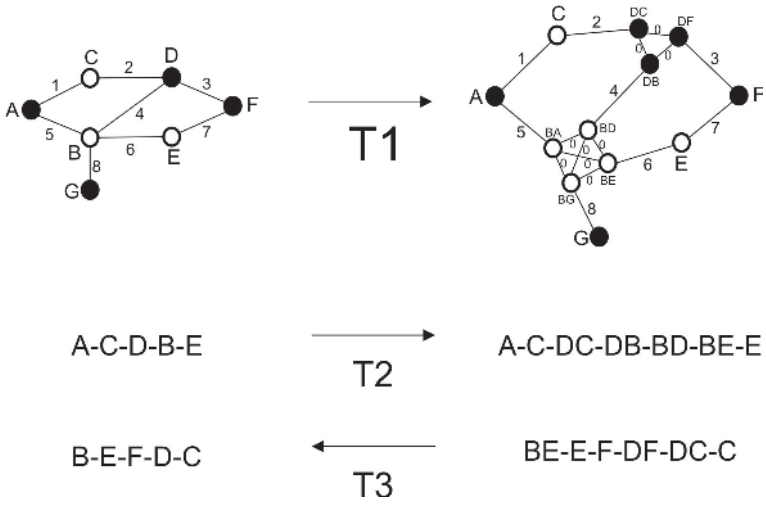

Fig. 11. Transforming GSP-EC into GSP-NC instances

instance, whose optimal solutions can in turn be transformed back (in polinomial time) into optimal solutions of the original GSP-EC instance. To do so a well-known node-splitting technique is applied as shown in Figure 11.. There, a GSP-EC instance involving four terminal nodes and three Steiner nodes is transformed (through $T1$) into a GSP-NC instance with six terminal nodes and six Steiner nodes. Every node with degree $d > 2$ is transformed into a d-clique with all edge costs equal to zero. The transformations $T2$ and $T3$ show how to translate a path from the GSP-EC instance to its corresponding path in the GSP-NC and vice-versa. After solving the GSP-NC instance, finding a solution S with cost c and a set P of certificate paths, these can be translated through $T3$ into a set of paths that happens

to be a valid solution of the GSP-EC instance and with the same cost c. We proved also that if S is optimal for the GSP-NC instance, then $T3(S)$ will be an optimal solution of the GSP-EC instance. This opened the question of wether it is best to focus on solving efficiently GSP-NC instances and using these transformations to solve GSP-EC instances, or focus also on developing GSP-EC specific algorithms which make use of the particulars introduced by the chance of node reusing among paths.

4.2 Altering costs

As we have seen above, altering the costs under certains hipothesis guarantees that the construction phase algorithm can have as high a chance of building an optimal solution as wanted provided the needed amount of iterations are run. In our tests we have used the exponencial distribution to alter the costs; it satisfies those hipothesis. Nonetheless it is pending to investigate which other distributions can be applied, eventually with better results. The exponential distribution just takes into account the expected value of the edge cost as parameter, not considering other factors that characterize the problem like costs dispersion. During testing we found that for those cases with little cost differences among edges the exponential distribution tends to generate too randomized values (i.e. with high dispersion); for many of those cases better results were attained using alternative ways of randomization that took under consideration also the standard deviation of the edge costs.

4.3 A recursive neigbourhood approach

The neigbourhoods that we used to define local search movements are based on the k-decomposition of graphs into key-paths, key-trees and key-stars. When studying the performance of the algorithms we have found several other types of structures that could be optimized yet using more complex movements i.e. not decomposable into sequences of key-paths, key-trees or key-stars replacements. We have also seen that the deterministic computation of the optimal replacing structure rapidly gains complexity when considering more complex structures; in the case of key-stars we have presented an algorithm for computing the best replacing key-star that made processing time soar because of the need of computing k-shortest disjoint paths. Therefore we consider relevant the creation of neighbourhood mechanisms that encompass heuristic criteria with complex structures replacement. We are working on an extension of the GSP that we named EGSP and can be solved by recursively invoking smaller instances of itself, determined by the supression of any desired edge set; this should allow to choose at will the structures to replace no matter their size or complexity. Current questions regarding this topic include: which kind of structures to replace given a certain (sub)instance of the problem for the next recursive calls; and at which point the use of more direct algorithms like the ones in this chapter turns to be more convenient.

5. Conclusion

In this chapter we introduced a framework and algorithms suitable to address the design of minimal cost networks under connectivity constraints, modelled as Generalized Steiner Problems both node-connected (GSP-NC) and edge-connected (GSP-EC). Our algorithm GRASP_GSP was shown to find good quality solutions to the GSP-EC when applied to a series of heterogeneous test cases with up to 100 nodes and up to 406 edges. For all cases with known optimal cost the algorithm was able to find solutions with costs no more than

0,74% higher than the optimal cost. Significant cost reductions were achieved after applying the local search phase over the greedy solutions built by the construction phase. Execution times of the tests run for the GSP-EC were comparable to the ones of previous similar works (Robledo & Canale, 2009) for the node-connected version of the GSP-NC and similar sizes. We also mentioned current research lines and extensions on the ideas treated within this chapter.

6. References

Agrawal, A., Klein, P. & Ravi, R. (1995). When trees collide: an approximation algorithm for the generalized Steiner problem on networks, *SIAM Journal on Computing* 24(3): 440–456.

Auletta, V., Dinitz, Y., Nutov, Z. & Parente, D. (1999). A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph, *Journal of Algorithms* 32(1): 21–30.

Baïou, M. (1996). *Le problème du suos-graphe Steiner 2-arête connexe: approache polyédrale*, PhD thesis, Université de Rennes I.

Balakrishnan, A., Magnanti, T. & Mirchandi, P. (2004). Connectivity-splitting models for survivable network design, *Networks* 43(1): 10–27.

Bhandari, R. (1997). Optimal physical diversity algorithms and survivable networks, *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pp. 433 –441.

Bienstock, D., Brickell, E. & Monma, C. (1990). On the structure of minimum weight k-connected spanning networks, *SIAM Journal on Discrete Mathematics* 3(3): 320–329.

Böckenhauser, H., Bongartz, D., Hromkovič, J., Klasing, R., Proietti, G., Seibert, S. & Unger, W. (2002). On the hardness of constructing minimal 2-connected spanning subgraphs in complete graphs with sharpened triangle inequality, *Proceedings of the 22nd Conference Kanpur on Foundations of Software Technology and Theoretical Computer Science*, Springer-Verlag, pp. 59–70.

Cancela, H., Robledo, F. & Viera, O. (2005). A parallel algorithm for the Steiner 2-edge-survivable network problem, *Journal of ICHIO (Chilean Institute of Operations Research)* 7(1): 15–27.

Cheriyan, J., Jordan, T. & Nutov, Z. (2001). On rooted node-connectivity problems, *Algorithmica* 30(3): 353–375.

Cheriyan, J. & Thurimella, R. (2000). Approximating minimum-size k-connected spanning subgraphs via matching, *SIAM Journal on Computing* 30: 528–560.

Cheriyan, J., Vempala, S. & Vetta, A. (2002). Approximation algorithms for minimum-cost k-vertex-connected subgraphs, *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pp. 306–312.

Chopra, S. (1992). Polyhedra of the equivalent subgraph problem and some edge connectivity problems, *SIAM Journal on Discrete Mathematics* 5(3): 321–337.

Chou, W. & Frank, H. (1970). Survivable communication networks and the terminal capacity matrix, *IEEE Transactions on Circuit Theory* 17: 192–197.

Christofides, N. & Whitlock, C. (1981). An algorithm for the design of optimal invulnerable networks, *Technical Report IC-OR-81-6*, Imperial College, London.

Coullard, R., Rais, A., Rardin, R. & Wagner, D. (1991). Linear-time algorithm for the 2-connected Steiner subgraph problem on special classes of graphs, *Technical Report No. 91-25*, School of Industrial Engineering, Purdue University.

Csaba, B., Karpinski, M. & Krysta, P. (2002). Approximability of dense and sparse instances of minimum 2-connectivity, TSP and path problems, *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 74–83.

Czumaj, A. & Lingas, A. (1999). On approximability of the minimum-cost k-connected spanning subgraph problem, *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 281–290.

Frank, H. & Chou, W. (1970). Connectivity considerations in the design of survivable networks, *IEEE Transactions on Circuit Theory* 17: 486–490.

Frederickson, G. & Jàjà, J. (1982). On the relationship between biconnectivity augmentation and traveling salesman problem, *Theoretical Computer Science* 19: 189–201.

Gabow, H., Goemans, M. & Williamson, D. (1993). An efficient approximation algorithm for the survivable network design problem, *Proceedings of the 3rd MPS Conference on Integer Programming and Combinatorial Optimization*, pp. 57–74.

Goemans, M. & Bertsimas, D. (1993). Survivable networks, linear programming relaxations and the parsimonious property, *Mathematical Programming* 60: 143–166.

Goemans, M., Goldberg, A., Plotkin, S., Tardos, E. & Williamson, D. (1994). Improved approximation algorithms for network design problems, *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pp. 223–232.

Goemans, M. & Williamson, D. (1992). A general approximation technique for constrained forest problems, *SIAM Journal on Computing* 24(2): 296–317.

Grötschel, M. & Monma, C. (1990). Integer polyhedra associated with certain network design problems with connectivity constraints, *SIAM Journal on Discrete Mathematics* 3: 502–523.

Grötschel, M., Monma, C. & Stoer, M. (1991). Polyhedral Approaches to Network Survivability, *in* F. Roberts, F. Hwang & C. Monma (eds), *Reliability of Computer and Communication Networks, Proc. Workshop 1989, New Brunswick, NJ/USA*, Vol. 5 of *Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, pp. 121–141.

Grötschel, M., Monma, C. & Stoer, M. (1992a). Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints, *Operations Research* 40(2): 309–330.

Grötschel, M., Monma, C. & Stoer, M. (1992b). Facets for polyhedra arising in the design of communication networks with low-connectivity constraints, *SIAM Journal on Optimization* 2(3): 474–504.

Grötschel, M., Monma, C. & Stoer, M. (1995). Polyhedral and computational investigations for designing communications networks with high survivability requirements, *Operations Research* 43(6): 1012–1024.

Jain, K. (1999). A 3-approximation algorithm for finding optimum 4,5-vertex-connected spanning subgraphs, *Journal of Algorithms* 32(1): 31–40.

Jain, K. (2001). A factor 2 approximation algorithm for the generalized Steiner network problem, *Combinatorica* 21: 39–60.

Kerivin, H. & Mahjoub, R. (2005). Design of survivable networks: A survey, *Networks* 46: 1–21.

Khuller, S. & Raghavachari, B. (1996). Improved approximation algorithms for uniform connectivity problems, *Journal of Algorithms* 21(2): 434–450.

Ko, C. & Monma, C. (1989). Heuristics methods for designing highly survivable communication networks, *Technical report*, Bellcore.

Koch, T., Martin, A. & Voß, S. (2000). SteinLib: An updated library on Steiner tree problems in graphs, *Technical Report ZIB-Report 00-37*, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin.
URL: *http://elib.zib.de/steinlib*

Kortsarz, G., Krauthgamer, R. & Lee, J. R. (2004). Hardness of approximation for vertex-connectivity network-design problems, *SIAM Journal on Computing* 33(3): 704–720.

Kortsarz, G. & Nutov, Z. (2003). Approximating node connectivity problems via set covers, *Algorithmica* 37(2): 75–92.

Monma, C., Munson, B. & Pulleyblank, W. (1990). Minimum-weight two connected spanning networks, *Mathematical Programming* 46: 153–171.

Ravi, R. & Klein, P. (1993). When cycles collapse: a general approximation technique for constrained 2-connectivity problems, *Proceedings of the 3rd Symposium on Integer Programming and Combinatorial Optimization*, pp. 39–55.

Ravi, R. & Williamson, D. (1997). An approximation for minimum-cost vertex-connectivity problems, *Algorithmica* 18(1): 21–43.

Ravi, R. & Williamson, D. (2002). Erratum: An approximation for minimum-cost vertex-connectivity problems, *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1000–1001.

Reinelt, G. (2004). TSPLIB library, http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html.

Resende, M. & Ribeiro, C. (2003). Greedy randomized adaptive search procedures, *in* F. Glover & G. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, pp. 219–249.

Robledo, F. & Canale, E. (2009). Designing backbone networks using the Generalized Steiner Problem, *Proceedings of the International Workshop of Design of Reliable Communication Networks (DRCN'09)*, Washington DC, pp. 327–334.

Steiglitz, K., Weiner, P. & Kleitman, D. (1969). The design of minimum-cost survivable networks, *IEEE Transactions on Circuit Theory* 16: 455–460.

Stoer, M. (1992). *Design of survivable networks*, Vol. 1531 of *Lecture Notes in Mathematics*, Springer-Verlag.

Williamson, D., Goemans, M., Mihail, M. & Vazirani, V. (1995). A primal-dual approximation algorithm for the generalized Steiner network problem, *Combinatorica* 15: 435–454.

Winter, P. (1985). Generalized Steiner problem in outerplanar graphs, *BIT* 25(3): 485–496.

Winter, P. (1986). Generalized Steiner problem in series-parallel networks, *Journal of Algorithms* 7: 549–566.

Winter, P. (1987). Steiner problem in networks: A survey, *Networks* 17(2): 129–167.

Optical Fiber Communications and Devices

Edited by Dr Moh. Yasin

ISBN 978-953-307-954-7
Hard cover, 380 pages
Publisher InTech
Published online 01, February, 2012
Published in print edition February, 2012

This book is a collection of works dealing with the important technologies and mathematical concepts behind today's optical fiber communications and devices. It features 17 selected topics such as architecture and topologies of optical networks, secure optical communication, PONs, LANs, and WANs and thus provides an overall view of current research trends and technology on these topics. The book compiles worldwide contributions from many prominent universities and research centers, bringing together leading academics and scientists in the field of photonics and optical communications. This compendium is an invaluable reference edited by three scientists with a wide knowledge of the field and the community. Researchers and practitioners working in photonics and optical communications will find this book a valuable resource.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821