*This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2008 proceedings.*

# On Survivable Access Network Design: Complexity and Algorithms

Dahai Xu
AT&T Labs - Research
dahaixu@research.att.com

Elliot Anshelevich
CS, Rensselaer Polytechnic Institute
eanshel@cs.rpi.edu

Mung Chiang
EE, Princeton University
chiangm@princeton.edu

***Abstract*— With economic constraints and limited routing capability, the structure of an access network is typically a "fat tree", where the terminal has to relay the traffic from another terminal of the same or higher level. New graph theory problems naturally arise from such features of access network models, different from those targeted towards survivable backbone (mesh) networks. We model the important problem of provisioning survivability to an existing single-level fat tree through two graph theory problem formulations: the Terminal Backup problem and the Simplex Cover problem, which we show to be equivalent. We then develop two polynomial-time approaches, indirect and direct, for the Simplex Cover problem. The indirect approach of solving the matching version of Simplex Cover is convenient in proving polynomial-time solvability though it is prohibitively slow in practice. In contrast, leveraging the special properties of Simplex Cover itself, we demonstrate that the direct approach can solve the Simplex Cover problem very efficiently even for large networks. Extensive numerical results of applying our algorithms are also reported for designing survivable access networks over different types of topologies.**

## I. Introduction

### A. Motivation

In this paper, we address topology design problems to offer fast recovery for access networks after natural failures or malicious attacks. With the ongoing deployment of "triple play" services by cable and telecom service providers (such as U-Verse by AT&T and FiOS by Verizon), the access parts of today's broadband network infrastructure have tree-like topology and aggregate increasing volumes of voice, data, and video traffic from end users. Due to cost-per-customer, carriers are unable to apply some of the redundancy features of backbone networks into the access portion of the network and often design the access network topology as a tree. To maximize protection, it is necessary to design *survivable tree* topologies, through the appropriate addition of a limited number of redundant links to the tree, that can provide the best survivability-cost tradeoff.

Since routing capability is expensive [1], often only the central offices can be equipped with the routing capability, and the other terminals within the access network only have the basic switching capability (such as traffic aggregation using multiplexing and demultiplexing). Accordingly, the structure of the access network is a "fat" tree rooted at the central office, i.e., for an intermediate terminal node within an access network, the capacity/traffic of its upstream link is the aggregation of the capacity/traffic of all its downstream links. A sample fat tree [2] is shown in Fig. 1. Therefore, to recover from a failure of its upstream link, the terminal has to relay the traffic from another terminal of the same or higher level. Such features of access networks make the problem of designing a survivable access network different from that of designing a survivable backbone network.

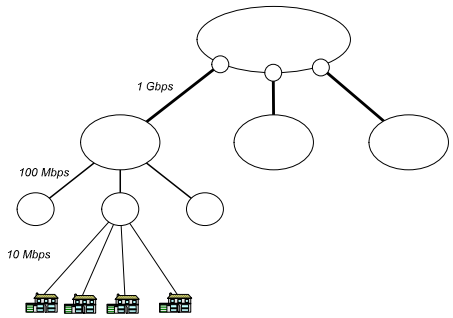


Fig. 1. Logical fat-tree architecture for access network

There are four dimensions along which we develop the following taxonomy of the graph theory problems in this research area [3]:

*1) Fat-Tree Exists or Not:* We can either add necessary survivability on an *existing* fat-tree to cater for user demands, or *jointly* design the tree and the redundant links in the first place.

*2) Single-Level or Multi-Level Tree:* The access network between remote terminals and the central office can be a single-level tree (i.e., star network), or a multi-level tree by putting low cost switches between the levels.

*3) Optimization (Objective-Constraint) Models:* One type of problems is to minimize the total cost to construct and maintain the access network by satisfying the connectivity requirements (e.g., $r_i$ edge-disjoint paths from remote terminal $i$ to the root, so terminal $i$ can survive against up to $r_i - 1$ edge failures). In the second type of problems, the network vendor can achieve a different amount of revenue by provisioning different levels of connectivity for terminal $i$, (i.e., $\{r_i\}$ become optimization variables), the objective is to maximize the total revenue constrained by the limited budget available for network construction.

*4) Link Cost Models:* Link cost is often an affine or convex function of distance, and a concave or constant function of capacity. For example, the concave cost model is mostly used


978-1-4244-2026-1/08/$25.00 © 2008 IEEE

TABLE I
Computational Complexity of Various Problems of Survivable Access Network Design

| *Objective* | | Minimize Total Cost | | Maximize Revenue with Limited Budget |
|---|---|---|---|---|
| *Link Cost Model* | | Uncapacitated Fixed | General Concave | |
| *Fat-Tree Exists* | *Single-Level* | **P** | NPC [1], [4], etc. | NPC |
| | *Multi-Level* | **NPC** | | |
| *No Existing Fat-Tree* | | NPC [5], [6], etc. | | |

in designing fiber networks, where the cost per unit length of deploying an edge is a concave non-decreasing function of the capacity/flow on the edge. The concavity is due to a buy-at-bulk effect. The average cost per unit bandwidth of using a larger capacity cable is usually less than that of using a lower capacity cable, e.g., one OC-12 cable is cheaper than 12 OC-1 cables. An extreme case of concave edge cost models is the uncapacitated fixed cost model, where the cable cost is negligible if the cost of deploying a set of cables with trenching (or hanging along poles) is dominant. With this cost model, the accurate values of traffic demands and flows on edges are not crucial since unlimited capacity is assumed.

There are 16 combinations of problem formulations from the above taxonomy, leading to a rich array of graph-theoretic models for access networking applications. Some of these models have been studied while many remain under-explored.

The concave cost model makes most min-cost design problems NP-Complete [1], [4], [7]. With the uncapacitated fixed cost model, if there is no existing fat-tree, then the survivable access network design problem degenerates into the well investigated *Survivable Network Design* problem, also known to be NP-Complete. In this problem, the goal is to find the minimum cost subgraph that has some pre-specified $r_i$ edge disjoint paths from terminal $i$ to the root [5], [6]. If there exists a multi-level fat-tree, we prove in Appendix B that the problem of provisioning survivability at minimum cost is NP-Complete. We will not further classify the complexities of the budget constrained problems, which are NP-Complete in general, since they can be treated as the decision versions of many NP-Complete optimization problems.

In this paper, motivated by formulations most often encountered in practice, we are interested in *designing the min-cost incremental topology that provides full survivability for all remote terminals within an existing single or multi-level access tree*. This tree already exists, connecting the central office and each remote terminal directly. To provide full survivability, we need to construct an augmented network where each terminal is connected to either one other terminal or the root. This problem can be represented as the *Terminal Backup* problem, where we are given a graph with terminals (required nodes), Steiner nodes (optional nodes), and weighted edges, and the goal is to find the cheapest subgraph so that every terminal is connected to at least one other terminal for backup purposes. We prove it is equivalent to the *Simplex Cover* problem, which involves finding an edge cover of some hypergraphs with certain properties. We will define and study these problems in Sections III-VI, where we also show that, surprisingly, both of these problems are polynomial-time solvable.

Table I summarizes the main results from prior works and this paper on the computational complexity of various problems related to survivable access network design. Problems that are studied in this paper are highlighted in bold.

### B. Overview and Organization

In this paper,

- We summarize the computational complexity of various problems in survivable access network design, based on both known results and new ones in this paper. In particular, we prove that the problem of provisioning survivability to an existing multi-level fat tree is NP-Complete.
- For provisioning survivability to an existing single-level fat tree, we formulate two basic graph theory problems, Terminal Backup and Simplex Cover, and prove they are equivalent.
- We propose an indirect approach with polynomial time complexity for Simplex Cover, by solving its matching version.
- We provide an algorithm to solve Simplex Cover directly, complemented with the elimination of unnecessary edges and an efficient local search policy.
- Extensive numerical experiments show that our direct approach can solve the Simplex Cover problem very quickly, even for very large networks.

The rest of the paper is organized as follows. Some of the related graph-theoretic problems are briefly summarized in Sec. II. In Sec. III, the problems of Terminal Backup and Simplex Cover are formally defined and proven equivalent. Direct and indirect approaches to Simplex Cover are discussed in Sec. IV and V, respectively. Numerical experiments are presented in Sec. VI. Conclusion is presented in Sec. VII. The complexity of provisioning survivability to the existing multi-level fat tree is addressed in Appendix.

## II. Related Work

In this section, we briefly summarize several related graph theory problems that have applications to network design. However, these formulations do not model the problems we study in this paper.

**Matching and Edge Cover**

The weighted (perfect) matching problem is to find the min-cost subset of edges where each node is met exactly by one edge. It has served as the cornerstone for many combinatorial problems since the seminal work of Edmonds [8]. There has been much research on efficient implementations of Edmonds's algorithm, e.g., [9], [10]. In particular, weighted matching is

one of the few integer programming problems that can be solved in polynomial time. The edge cover problem, to find the min-cost subset of edges to cover every node, can be transformed into a weighted matching problem [11].

**Survivable Network Design**

The general survivable network design problem is to find the min-cost subgraph such that there are $r_{i,j}$ edge-disjoint paths between each node pair $(i, j)$ [5], [6]. The best known results are Jain's factor 2 approximation [6] and Agrawal's $2\lceil \log_2(r+1) \rceil$ approximation where $r$ is the largest $r_{i,j}$ [12]. Survivable network design with concave costs has been studied in [13], [14], and [15] investigates algorithms for budget-constrained survivable topology design.

**Access Network Design**

Access Network Design (single root/sink) with general concave costs has been extensively addressed in [1], [7], [16]–[18], etc. For example, Andrews and Zhang introduce the access network design problem and propose a factor $O(K^2)$ approximation where $K$ is the number of trunk types [1]. Meyerson et al. provide an $O(\log n)$ approximation where $n$ is the number of sources [16]. Guha et al. present the first constant approximation (factor 80.566) [7].

**Multiple Tree Construction**

Médard et al. [19] construct two trees from an existing two-connected network such that for any link or node failure, every node remains connected to at least one of the trees.

Our work is different from those above in terms of problem formulations, and its leverage of the fat-tree structure in survivable access network design.

## III. Terminal Backup and Simplex Cover Problems

In this section, we formally define the Terminal Backup problem in a network with Steiner nodes and simplify it by proving it equivalent to the so-called *Simplex Cover* problem in the transformed network without Steiner nodes. Table II summarizes the key terms used in this paper.

Define the *Terminal Backup* problem as follows. We are given an undirected graph with terminals (required) nodes, Steiner (optional) nodes, and edges with nonnegative integer weights. The goal is to find the cheapest subgraph so that every terminal is connected to at least one other terminal.

In an optimal solution of Terminal Backup, OPT, there cannot be an edge such that its removal yields a feasible solution. This means that removing any edge of OPT must cut off exactly one terminal from the rest, and therefore all connected components of OPT must be stars. Moreover, we can decompose each star into stars containing either 2 or 3 terminals, which leads us to the following equivalent formulation of the Terminal Backup problem.

Given an instance $G$ of the Terminal Backup problem, create a hypergraph $H(G)$ containing 2D and 3D edges (i.e., the largest size of an edge is 3) as follows. Create a node for every terminal. For every terminal pair, $x, y$, find the shortest (minimum cost) path $P$ joining these two terminals in $G$, and create an edge $(x, y) \in H(G)$ with weight $w_{x,y} = cost(P)$. For every triple of terminals, $x, y, z$, find the shortest paths $P_1, P_2, P_3$ connecting them to a Steiner point, and create a 3D edge $(x, y, z) \in H(G)$ with $w_{x,y,z} = cost(P_1) + cost(P_2) + cost(P_3)$. Note that $H(G)$ does not contain any Steiner nodes.

TABLE II
Definitions of Key Terms

| | |
|---|---|
| Edge Cover | A subset of edges to cover every node |
| Perfect Matching | A subset of edges where each node is met exactly by one edge |
| Terminal Backup | Min-cost subgraph where every terminal (required node) is connected to at least one other terminal by using Steiner (optional) nodes. |
| Steiner Tree | Min-cost tree containing all the terminals and any subset of the Steiner nodes. |
| Simplex Condition | For every 3D edge $(x, y, z)$, the corresponding 2D edges $(x, y)$, $(y, z)$ and $(x, z)$ also exist, and their weights satisfy $w_{x,y,z} \geq (w_{x,y} + w_{y,z} + w_{x,z})/2$ |
| Simplex Cover | Min-cost edge cover in a hypergraph with simplex condition |
| Simplex Matching | Min-cost perfect matching in a hypergraph with simplex condition |

We say that a graph containing 2D and 3D edges satisfies the *Simplex Condition* if for every 3D edge $(x, y, z)$, the corresponding 2D edges $(x, y), (y, z)$, and $(x, z)$ also exist, and $w_{x,y,z} \geq (w_{x,y} + w_{y,z} + w_{x,z})/2$. Notice that $H(G)$ satisfies this condition, because the shortest paths in $G$ obey the triangle inequality. *Simplex Cover* is defined as the problem of finding the minimum cost edge cover in a hypergraph obeying the simplex condition, i.e, covering every node with 2D or 3D edge.

We have Theorem 1 below that the minimum cost simplex cover in $H(G)$ corresponds exactly to the optimal solution of the Terminal Backup problem in $G$.

***Theorem** 1:* The Terminal Backup and Simplex Cover problems are equivalent.

*Proof:* See Appendix A. ■

Because of Theorem 1, we will focus on Simplex Cover in the rest of this paper. We can assume from now on that the weight of any 3D edge is less than the cost of any two corresponding 2D edges, since otherwise we would never use this 3D edge in a cover, and could eliminate it in advance.

In terms of computational complexity, Simplex Cover is between regular Edge Cover (polynomial time solvable) and Exact Cover by 3-SETS (NP Complete, X3C [20]). The former is for the network only with 2D edges, and the latter can be transformed into the problem of covering nodes only with 3D edges. There is an easy 4/3 approximation to Simplex Cover, obtained by finding an edge cover using only 2D edges, which can be done in polynomial time. To see why this is a 4/3-approximation, let $E^*$ be the optimal simplex cover, and let $E'$ be the 2D edge cover formed by taking every 3D edge $(x, y, z) \in E^*$ and replacing it by the cheapest two of $(x, y)$, $(y, z)$, and $(x, z)$. This cheapest pair of edges costs at most $\frac{2}{3}(w_{x,y}+w_{y,z}+w_{x,z})$, which is at most $\frac{4}{3}w_{x,y,z}$ by the simplex condition. Therefore $E'$ is at most $4/3$ more expensive than the optimal simplex cover, and so the optimal 2D edge cover is a $4/3$-approximation.

While the above argument gives a $4/3$ approximation to Simplex Cover, we can in fact find the optimal simplex cover exactly and also show that Simplex Cover problem is polynomial-time solvable. In the following two sections, we propose and compare two algorithms solving the Simplex

Cover problem: 1) an indirect and conceptually convenient approach by transforming it into the weighted simplex matching problem, and 2) a direct and computationally more efficient approach without the above transformation.

## IV. Indirect Approach for Simplex Cover: Transformed into Weighted Simplex Matching

The reduction from perfect matching to edge cover (see e.g. [11]) suggests that we can use the matching version of Simplex Cover to find the cheapest simplex cover. Because of this, we now look at *Simplex Matching*, which is the problem of finding the minimum cost perfect matching in a hypergraph satisfying the simplex condition.

### A. Theoretical Properties

We can use Simplex Matching to solve Simplex Cover by converting the cover version of the problem to a matching version. To do this, make another copy of $G$, called $G'$, as shown in Fig. 2. For every node $v$ in $G$ and its corresponding node $v'$ in $G'$, form an edge $(v, v')$ with weight $2 \cdot \min_{(v,w)\in E_2} w_{v,w}$, where $E_2$ is the set of 2D edges in $G$. Call this new graph $Q$. According to Theorem 2 below, we can find the minimum-cost

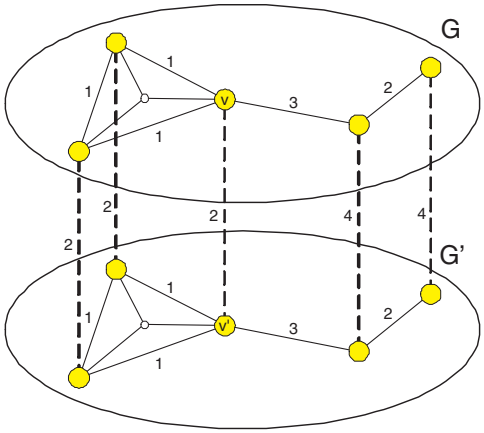


Fig. 2. Transform Simplex Cover to perfect Simplex Matching

simplex cover by solving its simplex matching version.

***Theorem** 2:* The minimum cost simplex cover can be found by solving the instance of weighted Simplex Matching described above.

*Proof:* Let $M$ be the minimum cost perfect simplex matching of hypergraph $Q$ with cost of $\overline{M}$. The costs of the selected edges in $G$ and $G'$ should be equal, otherwise we can replace the more expensive selected edge set with the cheaper one and obtain a new perfect simplex matching with cost of less than $\overline{M}$.

After obtaining $M$, put all the edges in $G$ selected by $M$ into our final solution of Simplex Cover. Now look at all the edges of the form $(v, v')$ that are in $M$, with $v$ in $G$ and $v'$ in $G'$. Take the 2D edge in $G$ that has the weight of $w_{v,v'}/2$, and add the edge to our solution. This creates a valid simplex cover for the original problem which costs $\overline{M}/2$.

We now prove that this simplex cover has minimum cost. Assume that the cost of the cheapest simplex cover $S$ is $\overline{S} < \overline{M}/2$. Note that each component of a min-cost simplex cover will be either a star (one or more 2D edges) or a single 3D edge. This is because if a 3D edge $(x, y, z)$ is in $S$, and $x$ is contained in another edge of $S$, then we could use the 2D edge $(y, z)$ instead of $(x, y, z)$ in our cover without increasing the cost. Form a perfect matching of $Q$ as follows. Select $S$ in $G$, and a copy of $S$ in $G'$. This is not a matching because of the star components in $S$. Consider such a component $C$ in $S$ with two or more edges. We can replace all but one of these edges with edges of the form $(v, v')$ with $v$ in $G$ and $v'$ in $G'$ without increasing the cost. This results in a perfect matching of $Q$ with cost of exactly $2 * \overline{S} < \overline{M}$, which contradicts the proposition that $M$ is the minimum cost perfect simplex matching of $Q$.

In summary, solving the weighted simplex matching version can be used to create the minimum cost simplex cover for the original problem. ■

### B. Algorithm for Weighted Simplex Matching

1: Start with any perfect simplex matching which uses the set $\Phi$ of 3D edges.
2: $\delta \leftarrow MinMatching(\Phi)$
3: **repeat**
4: $\quad\Phi' \leftarrow \Phi$; $\delta_0 \leftarrow \delta$
5: $\quad$**for each** $\Phi'$ resulting from a pair of trigger operations for $\Phi$ **do**
6: $\quad\quad\delta' \leftarrow MinMatching(\Phi')$
7: $\quad\quad$**if** $\delta > \delta'$ **then**
8: $\quad\quad\quad\delta \leftarrow \delta'$; $\Phi \leftarrow \Phi'$
9: $\quad\quad\quad$**continue** for breadth-first search or **break** for depth-first search
10: $\quad\quad$**end if**
11: $\quad$**end for**
12: **until** $\delta = \delta_0$ /*no further improvement*/
13: return $\delta, \Phi$

**Algorithm 1:** Weighted_Simplex_Matching

Algorithm 1 shows the procedure of searching for the minimum cost perfect simplex matching. $\Phi$ is the set of selected 3D edges, $\delta$ keeps the minimum matching cost found so far, and the function $MinMatching(\Phi)$ returns the cost of the 3D edges in $\Phi$ plus the minimum cost regular perfect matching in the residual graph after removing all the nodes adjacent to the 3D edges in $\Phi$. The trigger operation is either of the following two operations: *removing a 3D edge from* $\Phi$ *or adding an unused 3D edge into* $\Phi$. This algorithm is inspired from the proof of polynomial-time solvability in [21].

***Theorem** 3:* Algorithm 1 solves weighted simplex matching in polynomial time.

The detailed analysis and the correctness proof of Algorithm 1 can be found in [21]. Basically, Algorithm 1 needs to run (regular) weighted matching $(m^2 n \log(OPT))$ times, while a regular perfect matching takes $O(n^3)$ time [9]. Here $n$ is the number of nodes in $H$ and $m$ is the number of 3D

edges. This gives us the complexity of $O(\log(OPT) \cdot n^4 m^2)$ for Algorithm 1.

## V. Direct and Efficient Algorithm for Simplex Cover

Though polynomial time in theory, the indirect approach for solving Simplex Cover as discussed in Sec. IV would be prohibitively slow even for small networks. This is partially because the number of 3D edges $m$ could be as high as $n^3$, which would make the total complexity be $O(\log(OPT) \cdot n^{10})$. However, making use of the special properties of simplex cover itself, we can design a much faster approach to find a simplex cover directly, which is practical even for large networks deployed by access network service providers today.

### A. Remove Unnecessary 2D and 3D Edges from Simplex Cover

Some 2D and 3D edges will never be included in an optimal simplex cover. For terminal $a$, denote $\gamma(a)$ as the closest neighbor node and $\hat{w}_a$ as the closest neighbor distance where $\hat{w}_a = w_{a,\gamma(a)} = \min_{(a,i)\in E_2} w_{a,i}$ and $E_2$ is the set of 2D edges. For the sample graph in Fig. 3(a), $c$ and $d$ are the closest neighbor nodes for $a$ and $b$ respectively. If $w_{a,b} \geq \hat{w}_a + \hat{w}_b = w_{a,c} + w_{b,d}$, then 2D edge $(a,b)$ can be removed from the graph, since covering node $a$ and $b$ with one 2D edge $(a,b)$ is always worse than covering them with two 2D edges $(a,c)$ and $(b,d)$. Note that such removal is not always legitimate for the perfect matching problem. The only instance of perfect matching for Fig. 3(a) is $\{(a,b),(c,e),(d,f)\}$.

A 3D edge also can be removed from an instance of simplex cover if its weight is larger than the sum of the weights of a 2D edge between its two nodes and the closest neighbor distance of the third node. For example, in the graph of Fig. 3(b), if $w_{a,b,c} \geq w_{a,b} + \hat{w}_c = w_{a,b} + w_{c,e}$, then covering nodes $a, b$ and $c$ with 3D edge $(a,b,c)$ is not as good as using two 2D edges $(a,b)$ and $(c,e)$. However, such 3D edges might be required for a perfect matching, and the instance of simplex perfect matching for Fig. 3(b) is $\{(a,b,c),(d,f),(e,g)\}$.

As we will show in Sec. VI, such an elimination of unnecessary 2D and 3D edges makes the effective (residual) network very sparse.

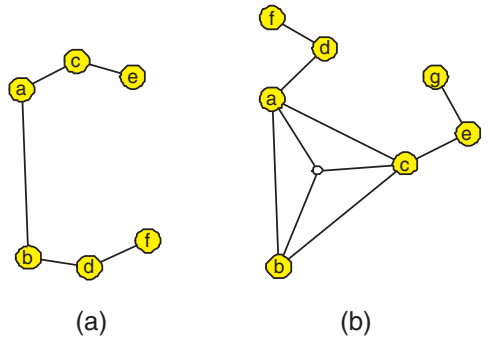


Fig. 3. Remove unnecessary 2D and 3D edges from Simplex Cover

```
1: Φ ← ∅
2: δ ← MinCover(Φ) /* Start with the min-cost regular edge cover (without using any 3D edges)*/
3: repeat
4:    Φ′ ← Φ; δ_0 ← δ
5:    for each Φ′ resulting from legitimate trigger operations for Φ do
6:       δ′ ← MinCover(Φ′)
7:       if δ > δ′ then
8:          δ ← δ′; Φ ← Φ′
9:          continue for breadth-first search or break for depth-first search
10:      end if
11:   end for
12: until δ = δ_0 /*no further improvement*/
13: return δ, Φ
```

**Algorithm 2:** Weighted_Simplex_Cover

### B. Direct Algorithm for Simplex Cover

Algorithm 2 shows the procedure of finding a simplex cover directly. Function $MinCover(\Phi)$ returns the sum of the weights of the 3D edges in $\Phi$ plus the minimum cost regular edge cover in the residual graph after removing all the nodes adjacent to 3D edges in $\Phi$. The differences from Algorithm 1 are underlined. Denote '+' as adding an unused 3D edge into $\Phi$ and '-' as removing a 3D edge from $\Phi$. We define legitimate trigger operations here as limited to '+', '+ +', '- +', '-' and '- -'. Note that '+ +', '- +' and '- -' mean two simultaneous trigger operations, which cannot be replaced with two sequential single trigger operations.

***Theorem** 4:* Algorithm 2 solves weighted Simplex Cover in polynomial time.

*Proof:* Essentially, the trail of the legitimate trigger operations in searching for a simplex cover is equivalent to a valid trail of trigger operations in solving the corresponding perfect simplex matching problem. Obviously, there is a regular perfect matching corresponding to the initial min-cost regular edge cover (in the sense of the transformation from Theorem 2). The possible trigger operations starting from the initial regular perfect matching with reduction in cost could be $(+e_i, +e_j)$, $(-e_i, -e_j)$, and $(+e_i, -e_j)$ where both $e_i$ and $e_j$ are 3D edges. Denote $clone(e_i)$ as $e_i$'s copy in $G'$ if 3D edge $e_i$ is in $G$ or $e_i$'s copy in $G$ if $e_i$ is in $G'$. If $e_j = clone(e_i)$, then $(+e_i, +e_j)$ and $(-e_i, -e_j)$ in Simplex Matching are equivalent to $+e_i$ and $-e_i$ in Simplex Cover respectively. If $e_j \neq clone(e_i)$, we always assume $e_i$ and $e_j$ are within the same plane (either in $G$ or $G'$), otherwise we can replace the more expensive plane with a copy of the cheaper one. Then we can "double" the trigger actions in Simplex Matching, e.g. two sequential $(+e_i, +e_j)$ and $(+clone(e_i), +clone(e_j))$ are equivalent to one $(+e_i, +e_j)$ in Simplex Cover. Therefore Algorithm 2 is guaranteed to converge to a global optimum within polynomial time because Algorithm 1 from Theorem 3 does the same. ■

### C. Local Search Policy

We now address the practical issues in implementing Algorithm 2 efficiently. Both Algorithms 1 and 2 are essentially local search methods but with the guarantees of convergence and global optimality. However, a good search policy can significantly speed up the convergence and avoid unnecessary duplicated tests if we do not maintain a search history (i.e. the tested instances of $\Phi'$ in Algorithms 1 and 2).

As we show in Sec. VI, the network becomes much sparser after removing unnecessary 2D and 3D edges from simplex cover. We can search for a min-cost simplex cover for each connected component separately. Basically, the more 3D edges a connected component has, the longer search time is required. In addition, unlike a 2D edge, once a 3D edge is chosen for a simplex cover, its three nodes cannot be covered by any other edges. This is true because otherwise it would be better to use one of its 2D "sub-edges" instead of the 3D edge. Therefore, we prefer a search policy that can decompose a large connected component as quickly as possible.

*1) Priority Based Search Policy:* We look further into the five legitimate trigger operations for simplex cover which include two basic trigger operations: '+' and '-' (i.e., add or remove a 3D edge from the current chosen 3D edge set $\Phi$). Note that a '+' operation, by adding an unused 3D edge from a connected component $C$, could decompose $C$ into up to 3 smaller connected components, which can be addressed separately. In contrast, a '-' operation could reunite up to 3 connected components to recover one larger connected component. Therefore we would like to use '+' operations as much as possible providing they can continuously reduce the total cost. In other words, the '+' operation should be granted higher priority over the '-' operation.

Notice that '+ +' cannot be replaced with two sequential '+' operations. But if there are $m'$ 3D edges in a connected component, then before finding an improvement we might need to consider $m'$ '+' operations and $\binom{m'}{2}$ '+ +' operations. Therefore we prefer the '+' operation to the '+ +' operation, with the hope that a '+' operation already decomposes the connected component before any '+ +' operation.

Similarly, a '- +' operation cannot be replaced with sequential '-' and '+' operations. We can examine the temporary and larger connected component resulting from the '-' operation, however, and simultaneously test the '+' operation only for each 3D edge within this temporary connected component.

In summary, we assign priorities for the 5 operations as shown in Table III.

TABLE III
Priorities of trigger operations for Simplex Cover

| + | 5 | + + | 4 | - + | 3 | - | 2 | - - | 1 |
|---|---|---|---|---|---|---|---|---|---|

*2) Breadth vs. Depth First Search:* With the breadth-first search policy, the selected 3D edge set $\Phi$ will not be updated until all the legitimate operations have been tried, while with the depth-first search policy, $\Phi$ is updated immediately once an improvement has been found. We prefer depth-first search since the connected components could be decomposed earlier. Note that the proof of polynomial running time only applies to the breadth-first search version of Algorithm 2, but the depth-first search version runs faster in practice.

*3) Resume vs. Reset:* For the '+' operation, assume the 3D edges within a connected component are represented internally with an order of $e_1, ..., e_i, ....$ If '+' $e_i$ brings an improvement, then in depth-first search we update $\Phi$ by adding $e_i$, and restart search from $e_{i+1}$ (resume) instead of $e_1$ (reset) since $+e_1$ has been tried before without yielding improvement (though with the old $\Phi$).

## VI. Performance Evaluation

We present the numerical results of searching for the min-cost simplex cover, focusing on single-level tree access networks. We consider two types of network topologies. The first is a "random network" where terminals are randomly distributed within a 9x9 square area. The second is a "grid network" where within each cell, three terminals are randomly placed around the center bounded by a radius parameter $r$, set to 0.4.

The integer weight of an edge is defined as its length times 100, rounded to the nearest integer. The length of a 2D edge between two terminals is either its Euclidean distance or Manhattan (rectilinear) distance. The length of a 3D edge between three terminals is the sum of the distances from every terminal to its Steiner point, which is the point that would minimize the length of such an edge. With Euclidean distance, the Steiner point is shown in Fig. 4 [22]. With rectilinear distance, it is known that the Steiner (optional) point is located at $(u, v)$, where the coordinate of terminal $i \in \{1, 2, 3\}$ is $(u_i, v_i)$, and $u$ and $v$ are the medians of $u_i$ and $v_i$ respectively.

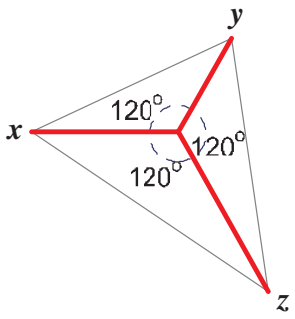


Fig. 4. Steiner point of a triangle in Euclidean space

### A. Performance comparison of indirect and direct approaches in medium-size networks

Fig. 5 and 6 show the logic topologies for two sample single-level access networks (49-node random network and 5X5 grid network), which connect the central office and each remote terminal directly. To provide survivability to the tree access network, we should construct the cheapest augmented network so that every terminal is connected to at least one other terminal for backup purposes. As demonstrated in Sec. III, this

Terminal Backup problem can be transformed into Simplex Cover on a hypergraph by defining a 2D edge between every two terminals and a 3D edge among every three terminals. Table IV shows the basic information about the resulting four hypergraphs (49-node random network with Euclidean distance, 49-node random network with rectilinear distance, 5X5 grid network with Euclidean distance, and 5X5 grid network with rectilinear distance). The resulting hypergraphs are very dense (with $O(n^2)$ 2D edges and $O(n^3)$ 3D edges) and thus it is impractical to solve the problem directly due to the large number of 3D edges. By eliminating unnecessary edges as in Sec. V-A, we can significantly reduce the problem size. All the following performance evaluation is based on the "simplified" hypergraphs after redundant edge removal. Figs. 7-10 show the simplified hypergraphs with effective 2D and 3D edges (including solid and dashed lines) and the optimal simplex cover (solid lines). For visual clarity, *the primary links in the existing tree are not shown* when the simplex covers are shown. We observe that the effective (residual) network becomes very sparse after the elimination of unnecessary 2D and 3D edges.

TABLE IV
Topology information of middle-size networks

| Network | Random network | | 5X5 grid network | |
|---|---|---|---|---|
| | Euclidean | rectilinear | Euclidean | rectilinear |
| Node # | 49 | 49 | 75 | 75 |
| Original 2D edge # | 1176 | 1176 | 2775 | 2775 |
| Original 3D edge # | 11403 | 13055 | 40739 | 47863 |
| Effective 2D edge # | 74 | 75 | 122 | 118 |
| Effective 3D edge # | 11 | 22 | 25 | 30 |
| Largest 3D edge # in a component | 8 | 14 | 21 | 26 |

The major computational complexities of indirect and direct approaches for weighted simplex cover as in Sec. IV and V come from their repeated invocations of the usual 2D perfect matching algorithm as a sub-routine. We adopted Cook Rohe's implementation of Edmonds' blossom algorithm, blossom4, as the weighted matching subroutine [10]. Table V shows the number of weighted matching sub-routines called by the two approaches to find the min-cost simplex cover. In the worst case, the indirect approach invokes the blossom4 sub-routine 18207 times, which is around 70 times as much as that in the direct approach (which invokes the sub-routine only 265 times). On average, the indirect approach calls the sub-routine significantly more times than the direct approach, especially for large networks with rectilinear distance.

TABLE V
Number of invocations on regular weighted matching subroutine

| Network | 49-node random network | | 5X5 grid network | |
|---|---|---|---|---|
| | Euclidean | rectilinear | Euclidean | rectilinear |
| Indirect Approach | 370 | 1395 | 7704 | 18207 |
| Direct Approach | 50 | 79 | 331 | 265 |

### *B. Further profile of direct approach for large-size networks*

In this section, we further investigate the performance of the direct approach for even larger networks (with 243 nodes), which are too large for the indirect approach. Table VI shows the topology information about four 243-node networks as well as the invocation number of the regular matching sub-routine and the time needed to find the optimal simplex cover on Redhat Enterprise Linux 4 with Intel Pentium IV processors at 2.8 Ghz. The proposed direct approach needs 25.97 seconds for the most complicated scenario (i.e., 9X9 grid network with rectilinear distance). This approach seems very promising for finding optimal simplex covers within a reasonable time, even for large-scale networks.

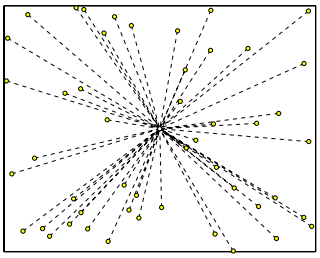

Fig. 5. Logic one-level access topology for random network with 49 remote terminals

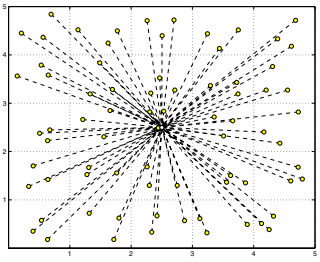

Fig. 6. Logic one-level access topology for 5X5 grid network with 75 remote terminals

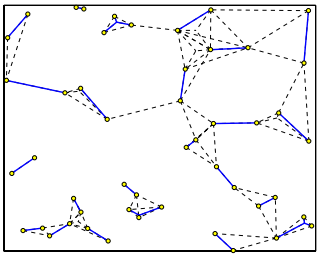

Fig. 7. Optimal simplex cover for 49-node random network (Euclidean distance)

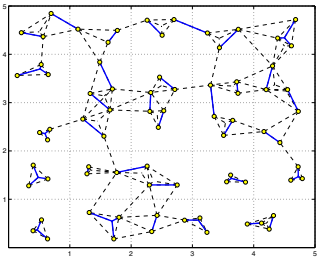

Fig. 8. Optimal simplex cover for 5X5 grid network (Euclidean distance)

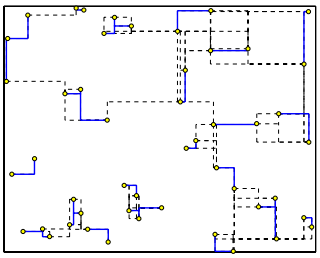

Fig. 9. Optimal simplex cover for 49-node random network (rectilinear distance)

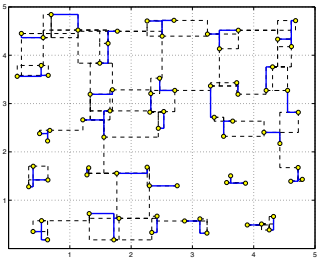

Fig. 10. Optimal simplex cover for 5X5 grid network (rectilinear distance)

TABLE VI
Topology of large-size networks and the performance of direct approach

| Network | 243-node random network | | 9X9 grid network | |
|---|---|---|---|---|
| | Euclidean | rectilinear | Euclidean | rectilinear |
| Node # | 243 | 243 | 243 | 243 |
| Effective 2D edge # | 419 | 395 | 384 | 413 |
| Effective 3D edge # | 31 | 103 | 70 | 113 |
| Largest 3D edge # in a component | 20 | 103 | 49 | 105 |
| # invocations on matching subroutine | 231 | 2825 | 1430 | 2147 |
| CPU Time (Seconds) | 1.40 | 15.16 | 9.41 | 25.97 |

Fig. 11 shows the cost reduction from regular edge cover (the starting point) using trigger operations before finding (and confirming) the optimal simplex cover for the largest connected component of the four large networks. As shown in the convergence behavior graph, our direct approach can find a near-optimal solution very quickly, suggesting that in practice we can stop an order-of-magnitude earlier rather than search for the optimal solution.

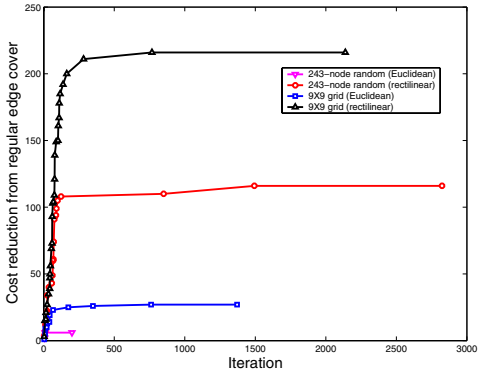


Fig. 11. Evolution of cost reduction from regular edge cover in the search for optimal simplex cover (for the largest connected component)

## VII. Conclusion

Many important problems in topology design of survivable access network can be formulated as problems of survivable fat tree construction. After classifying a taxonomy of problems in this area, we formulate and study two graph-theoretic models, the Terminal Backup and Simplex Cover formulations, which are shown to be equivalent. We develop two polynomial-time approaches to solve the Simplex Cover problem. In particular, complemented with elimination of unnecessary edges and an intelligent local search policy, the direct approach is efficient enough to solve problems with sizes matching today's large-scale access networks.

## Acknowledgment

This research is in part supported by the National Science Foundation NSF grants CNS-0427677, CCF-0448012, and CNS-0430487 and NSF Postdoctoral Fellowship in Mathematical Sciences. We are grateful to the helpful resources introduced to us by Matthew Andrews and Muriel Médard, and to the general discussion on access network topology design with Sandy Fraser and with AT&T Labs Network System Engineering.

## Appendix

### A. Proof of Theorem 1

*Proof:* Given an instance $G$ of the Terminal Backup problem, create an instance $H(G)$ of Simplex Cover as Sec. III.

For every optimal solution $E^*$ in $G$ of Terminal Backup, there exists a solution of the same (or cheaper) cost in $H(G)$. Take a component $C$ of $E^*$, with cost of $\overline{C}$. There cannot be an edge of $C$ whose removal splits $C$ into components

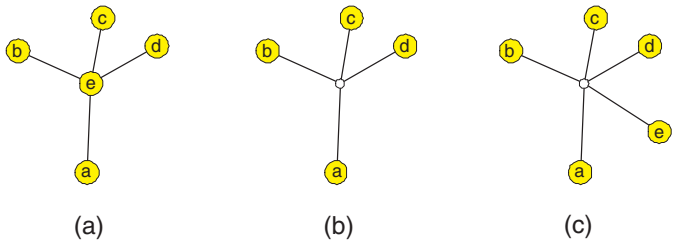


Fig. 12. Demonstration of the equivalence of Terminal Backup and Simplex Cover problems

each containing $\geq 2$ terminals, since then $E^*$ is not optimal. Therefore, $C$ must be a star. If the center of the star is a terminal as in Fig. 12(a), put the edges of $H(G)$ corresponding to the paths between the center and each other terminal into the simplex cover. The center of the star could be also a Steiner node as in Fig. 12(b) and (c). If $C$ contains an even number of terminals as in Fig. 12(b), match them up arbitrarily and choose the edges of $H(G)$ corresponding to the paths between these pairs, e.g. $(a, b)$ and $(c, d)$. These edges will cost at most $\overline{C}$, since the cheapest path connecting terminals $a$ and $b$ is cheaper than the path through the Steiner node that is the center of the star. If $C$ contains an odd number of terminals as in Fig. 12(c), choose a corresponding 3D edge as well, e.g. $(a, b, c)$ and $(d, e)$. The cost of these edges in $H$ are at most the cost of $\overline{C}$. All the nodes in $H(G)$ are covered in this manner, so it is a feasible solution with the same or lower cost as $E^*$.

Conversely, for every optimal solution $H^*$ in $H(G)$ for Simplex Cover, there exists a cheaper solution in $G$ just by taking the union of all the paths corresponding to edges of $H^*$. Therefore, *the cost of an optimal solution in G of Terminal Backup is the same as in $H(G)$ of Simplex Cover.*

If we are given an instance $H$ of Simplex Cover, we can convert it to an instance $G$ of Terminal Backup by putting a Steiner node in the middle of every 3D edge, and making the weights of the 2D edges formed respect the triangle inequality, which is always possible because of the simplex condition. Any solution in $H$ is trivially a solution in $G$. An optimal solution in $G$ will never use only 2 of the 3 "sub-edges" of a 3D edge, since the corresponding "shortcut" edge is always cheaper. Therefore, *an optimal solution in $G$ of Terminal Backup is a solution in $H$ of Simplex Cover as well.* ■

### B. Complexity of Survivable Multi-Level Fat Tree

We show that provisioning survivability to an existing multi-level fat tree is NP-complete. The *Survivable Multi-Level Fat Tree* problem is defined as follows. In a weighted graph $G$ with terminal nodes (including the root), Steiner (non-terminal) nodes, and edges with weights, we are given an existing tree $T$ spanning the terminals that does not contain any Steiner nodes. Label each terminal $v$ by its hop distance from the root in $T$ (i.e., the number of edges), and call this the $level$ of terminal $v$.

Let $P(v)$ be the path from terminal $v$ to the root in $T$. The goal of the survivable multi-level fat tree problem is to select

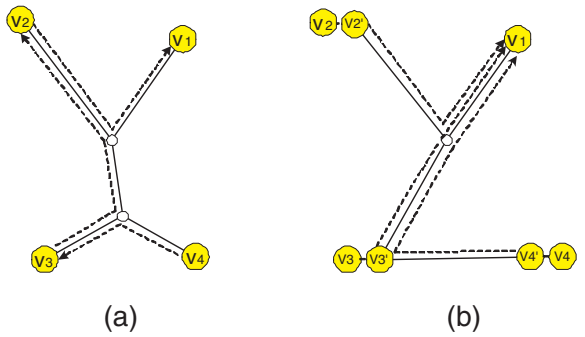


Fig. 13. (a) Form a Steiner tree from survivable multi-level fat tree (b) Form a survivable multi-level fat tree from Steiner tree

the cheapest set of edges $S$ so that every terminal $v$ will have a path to the root in $S \cup T$ that is edge-disjoint from $P(v)$. This will guarantee that $v$ can still connect to the root after any single edge failure. We also have the extra constraint that this alternate path cannot go through any lower nodes (i.e., with a larger level value than $level(v)$ due to the "fatness" of the tree).

***Theorem** 5:* The Survivable Multi-level Fat Tree problem is NP-Complete.

*Proof:* We can prove the problem is NP-Complete by a reduction from *Geometric Steiner Tree* (ND13 [20]). Given a geometric Steiner tree instance, where $A$ is the set of terminals that we must connect located in the Euclidian plane, we form an instance of Survivable Fat Tree as follows. The terminals are exactly the nodes in $A$. Choose an arbitrary order of nodes in $A$, and suppose this order is $v_1, v_2, \ldots$. Let $v_1$ be the root, and let the tree $T$ be a path consisting of edges $(v_i, v_{i+1})$. Only a polynomial number of points will ever be used as Steiner nodes in an optimal solution of the geometric Steiner tree instance, so we can assume that all of these points are given, and make the set of Steiner points in the instance of Survivable Fat Tree be the same.

We now show that every solution to the Survivable Fat Tree instance corresponds to a solution in the Steiner Tree instance, and vice versa. Take a solution $S$ to the survivable fat tree instance above. All the terminals of $A$ are connected by $S$, since every terminal $v_i$ has a path in $S$ to a terminal with higher level. This means that $v_i$ must connect to one of $v_1, v_2, \ldots, v_{i-1}$, and so all nodes are connected to the root by $S$. A solution for this instance of Survivable Fat Tree is therefore a solution for the Steiner Tree instance as well. E.g., in Fig. 13(a), from the added edges (dashed lines with arrows) for the existing multi-level fat-tree (omitted in Fig. 13), we can create a Steiner Tree instance (solid lines)

We now show that a solution to the Steiner Tree instance is also a solution to the Survivable Fat Tree instance. A Steiner tree may not connect the terminals in the correct order (according to $level(v)$), and it might connect them using edges in $T$. Given a Steiner tree, form a solution $S$ to the Survivable Fat Tree instance as follows. For every terminal $v \in A$, replace it in the tree with a Steiner node $v'$ located in the same location (or very close), and then add the 0-cost edge $(v, v')$. This connects all terminals directly to $v_1$ without going through any other terminals, which respects the level order, and does not use any edges in $T$. This is therefore a solution to the Survivable Fat Tree instance, as desired. In Fig. 13(b), from the Steiner Tree instance (solid lines), we can form the added edges (dashed lines with arrows) for the existing multi-level fat-tree. ■

## REFERENCES

[1] M. Andrews and L. Zhang, "The access network design problem," in *IEEE symposium on Foundations of Computer Science, FOCS'98, Palo Alto, CA*, 1998, pp. 40–59.
[2] A. Patzer, "Algorithms for a reliable access network," Master's thesis, Princeton University, 2004.
[3] M. Chiang, J. Huang, D. Xu, Y. Yi, C. Tan, and R. Cendrillon, "Fast copper for broadband access," in *Proc. 44th Allerton Conference*, Sep. 2006.
[4] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian, "Buy-at-bulk network design: approximating the single-sink edge installation problem," in *ACM-SIAM symposium on Discrete algorithms, SODA'97, New Orleans, LA*, 1997, pp. 619–628.
[5] V. V. Vazirani, *Approximation algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.
[6] K. Jain, "A factor 2 approximation algorithm for the generalized Steiner network problem," *Combinatorica, vol. 21, no. 1*, pp. 39–60, 2001.
[7] S. Guha, A. Meyerson, and K. Munagala, "A constant factor approximation for the single sink edge installation problems," in *ACM symposium on Theory of computing, STOC'01, Crete, Greece*, 2001, pp. 383–388.
[8] J. Edmonds, "Maximum matching and a polyhedron with 0-1 vertices," *Journal of Research at the National Bureau of Standards, 69B*, pp. 125–130, 1965.
[9] H. N. Gabow, "Implementation of algorithms for maximum matching on nonbipartite graphs," Ph.D. dissertation, Stanford University, 1974.
[10] W. Cook and A. Rohe, "Computing minimum-weight perfect matchings," *INFORMS J. on Computing*, vol. 11, no. 2, pp. 138–148, 1999.
[11] A. Schrijver, *Combinatorial Optimization - Polyhedra and Efficiency*. Springer-Verlag, 2003.
[12] A. Agrawal, P. Klein, and R. Ravi, "When trees collide: An approximation algorithm for the generalized Steiner problem on networks," *SIAM J. Comput.*, vol. 24, no. 3, pp. 440–456, 1995.
[13] H. Kerivin, D. Nace, and T.-T.-L. Pham, "Design of capacitated survivable networks with a single facility," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, pp. 248–261, 2005.
[14] M. Charikar and A. Karagiozova, "On non-uniform multicommodity buy-at-bulk network design," in *ACM symposium on Theory of computing, STOC'05, Baltimore, MD*, 2005, pp. 176–182.
[15] N. Garg, R. Simha, and W. Xing, "Algorithms for budget-constrained survivable topology design," in *IEEE ICC'02, New York, NY*, May 2002, pp. 2162–2166.
[16] A. Meyerson, K. Munagala, and S. A. Plotkin, "Cost-distance: Two metric network design," in *IEEE symposium on Foundations of Computer Science, FOCS'00, Redondo Beach, CA*, 2000, pp. 624–630.
[17] A. Meyerson, K. Munagala, and S. Plotkin, "Designing networks incrementally," in *IEEE symposium on Foundations of Computer Science, FOCS'01, Las Vegas, NV*, 2001, pp. 406–415.
[18] K. Munagala, "Approximation algorithms for concave cost network flow problems," Ph.D. dissertation, Stanford University, Department of Computer Science, 2003.
[19] M. Médard, S.G.Finn, R. Barry, and R. Gallager, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Transactions on Networking, vol. 7 no. 5*, pp. 641–652, 1999.
[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
[21] E. Anshelevich and A. Karagiozova, "Terminal backup, 3d matching, and covering cubic graphs," in *ACM symposium on Theory of computing, STOC'07, San Diego, CA*, 2007.
[22] Lo-Keng Hua et al., "Application of mathematical methods to wheat harvesting," *Chinese Math, no. 2*, pp. 77–91, 1962.