ARTICLE

Survivable Network Design with Constrained ɧ-Subgraph Flows

Alfonso B. Labao* and Henry N. Adorna

Department of Computer Science, College of Engineering, University of the Philippines, Diliman, Quezon City, 1101, Philippines

ABSTRACT

In this paper, we propose a variant of the survivable network design problem, where subgraphs are given flow constraints, which represent upper bounds on weights of outgoing edges with endpoints that belong to vertex subsets of these subgraphs. The general problem of verifying whether the weights of outgoing edges of any vertex subset (i.e. a graph cut) of an arbitrary subgraph meets a certain upper bound is a computationally hard problem. However, our proposed problem considers special types of subgraphs (termed ɧ-subgraphs) which possess a tree structure. In particular, a ɧ-subgraph consists of a trunk whose vertices are connected by a unique path and which are termed as trunk vertices. Each trunk vertex is connected to a branch which also has a tree structure. This special graph structure of ɧ-subgraphs - along with conditions over the flow constraint function and additional constraints on the optimization problem allow for the construction of a polynomial-time algorithm to solve our proposed problem - using efficient separation oracles under the ellipsoid method of linear programming. Our proposed algorithm provides $(2pl, 2z(S) + 3)$-performance where p is the maximum length of any path that connects the root and leaves of a branch of any ɧ-subgraph, l is the maximum number of leaves of any branch of a ɧ-subgraph, and S is a vertex subset of a branch of a ɧ-subgraph.

1. INTRODUCTION

The standard form of the survivable network design problem considers the problem of assigning an optimal weight to each edge of an input graph, such that the total cost of edges is minimized - subject to the constraint that certain connectivity requirements are met for each vertex pair, (Williamson et al., 1995). By connectivity requirements, we refer to lower bounds

*Corresponding author
Email Address: alfonso.labao@upd.edu.ph
Date received: April 2, 2023
Date revised: May 23, 2023
Date accepted: August 15, 2023

KEYWORDS

approximation algorithms; survivable network design; combinatorial optimization; graph algorithms; algorithmics

on the number of edge-disjoint paths connecting each pair of vertices in the input graph. This problem has several practical applications - particularly in the area of distributed computing with fault tolerance, (Guo et al., 2013), (Son et al., 2022). In several of these applications, vertices of the graph represent individual computers, while edges represent network connections. In order to ensure fault tolerance, a certain number of redundant connections have to be imposed for each pair of computers in the distributed network - which translate to connectivity requirements in the survivable network design problem.

The standard survivable network design problem is an NP-hard problem, (Vazirani, 2013), (Williamson et al., 1995), which implies that unless $P = NP$, polynomial algorithms to arrive at the optimal solution of the problem would be non-existent. Under this context, solving for approximate solutions of the survivable network design problem in polynomial-time would instead make more sense and be more feasible. This leads to the area of approximation algorithms, for which some well known NP-hard problems such as knapsack problem, the subset-sum problem, or the traveling salesman problem have been provided, (Ibarra and Kim, 1975), (Mömke and Svensson, 2011), (Batra et al., 2014).

In the case of the standard survivable network design problem, an early approximation algorithm is provided by (Williamson et al., 1995), which is a $2k$-approximation algorithm, where k represents the largest edge-connectivity requirement given any pair of vertices. The same authors improved this bound further to $2H_k$ using a primal-dual approach, where H_k is the k th harmonic number. A notable approximation algorithm for this problem however is provided in (Jain, 2001), which uses iterative rounding. Namely, the original survivable design problem is first transformed into a relaxed form, i.e. real solutions are allowed instead of integral solutions. Under this relaxed form, standard linear programming procedures would be able to compute for the (non-integral) optimal solution in polynomial-time. Afterwards, a key result in (Jain, 2001) is the fact that any basic feasible solution to the relaxed problem returned by linear programming results in at least one edge with weight of at least a half. The algorithm of (Jain, 2001) then includes such sufficiently weighted edges in a solution set of edges. Subsequently, these edges are removed in the graph resulting in a residual graph. The algorithm iteratively re-computes for an optimal solution under the residual graphs, then removes all sufficiently weighted edges until no candidate edges remain. This results in a 2-approximation guarantee, i.e. the solution returned by the algorithm is within a factor of 2 from the optimal solution. However, for Jain's algorithm to work in polynomial-time, the algorithm uses the ellipsoid method (Bland et al., 1981) to efficiently solve the linear program under an exponential number of constraints. The ellipsoid method requires a polynomial-time separation oracle, which tells the linear program if some constraint has been violated by a candidate solution, (Jain, 2001), (Vazirani, 2013).

Since the formulation of the standard form of the survivable network design problem, several other variants of the problem have been proposed. For instance, (Chekuri et al., 2012) incorporates prize-collecting features to the problem. Another variant of the survivable network design problem is proposed by (Lau and Singh, 2013) which incorporates constraints on vertex degrees, giving a $(2,2b_v + 3)$ bi-criteria approximation algorithm using iterative rounding with relaxation, where $2b_v + 3$ implies that under the approximation algorithm violates the degree bound b_v of vertex v by $2b_v + 3$.

1.1 Our Contributions

In this paper, we propose a variant of the survivable network design problem, whereby in addition to connectivity requirements, certain constraints on the outgoing flows of subgraphs are included. It should be noted that given an arbitrary set of weights for edges of an arbitrary subgraph, the general problem of verifying whether the weights of outgoing edges of each vertex subset of the subgraph complies with a provided constant upper bound is a computationally hard problem. This is because this problem is basically an instance of the MAX-CUT problem, which looks for the maximal cut in a graph. MAX-CUT is shown NP-complete (Ausiello, 2012). To see this fact, suppose that a solution to the MAX-CUT problem is discovered, in the form of a set of vertices whose outgoing edges have weights that are maximal. This implies that all other vertex subsets for a subgraph have outgoing edges whose weights are less than the weights provided by the solution. If the solution is less than or equal to the provided upper bound, then all other subsets likewise comply with the constant upper bound. On the other hand, if the solution to the MAX-CUT problem violates the upper bound, then there is at least one vertex subset that does not comply with the constant upper bound. Due to this computational difficulty, this paper narrows the type of subgraphs considered to certain subgraphs (termed $\mathfrak{h}$ -subgraphs)which follow a tree structure. In particular, a $\mathfrak{h}$ -subgraph consists of a trunk and a set of branches. The trunk of a $\mathfrak{h}$-subgraph is itself a subgraph made up of trunk vertices that are connected by a unique path, such that all edges of the trunk have to belong to this path. Each trunk vertex represents the root of a branch. A branch of a $\mathfrak{h}$-subgraph is a subgraph in the form of a tree (i.e. a graph with no cycles) whose root is a trunk vertex. The collection of $\mathfrak{h}$-subgraphs is termed $\mathcal{H}$, and in order for $\mathcal{H}$ to be considered as a valid input to the problem (i.e. an admissible input), vertex sets of $\mathfrak{h}$-subgraphs have to be disjoint. Moreover, for each $\mathfrak{h}$-subgraph of $\mathcal{H}$, vertex sets of its branches have to be likewise disjoint. Given $\mathcal{H}$, our proposed variant of the survivable network design problem provides upper bounds on the weights of outgoing edges of subsets of branches of each $\mathfrak{h}$-subgraph. However, the upper bounds have to have special properties such as being constant for all subsets of each branch of the $\mathfrak{h}$-subgraph, and of being defined only for subsets that contain the trunk vertex. Moreover, we impose additional constraints on the resulting optimization problem (OP 3) such as requiring that edges that are "closer" to the trunk vertex have greater weight than edges that are "farther" from the trunk vertex. Several other constraints and conditions are incorporated into our proposed problem which are described in detail in this paper.

Under these constraints, we propose a polynomial-time algorithm which use polynomial-time separation oracles. These separation oracles efficiently indicate any violation of the constraints of OP 3, which would form a part of the ellipsoid method needed for efficiently solving the linear program. Our proposed algorithm is a $(2pl, 2z(S) + 3)$ -approximation algorithm, where p is the maximum length of any path that connects the root and leaves of any branch of any $\mathfrak{h}$-subgraph, while l is the maximum number of leaves of any branch of any $\mathfrak{h}$-subgraph, and S is a subset of a branch of a $\mathfrak{h}$-subgraph that is subjected to flow constraints. Our algorithm uses a variant of the iterative rounding with relaxation algorithm shown in (Lau and Singh, 2013). We provide detailed technical proofs in the Appendix to prove the approximation guarantees of our proposed algorithm which uses the concept of laminar sets and properties of submodular / weakly-supermodular functions.

2. Preliminaries

For any $k \in \mathbb{N}$, let $[k]$ denote $\{1,2,..,k\}$. Let $G(V,E)$ be a graph with set of vertices V and set of edges E. A vertex $v \in V$

is denoted in lowercase, while an edge connecting a pair of vertices $i, j \in V$ is denoted by $e_{i,j}$, and the vertices i, j are the *endpoints* of $e_{i,j}$. In some cases, we also denote an arbitrary edge in E simply as e, with no endpoints specified. Throughout the paper, we assume that all graphs G are *undirected*. This implies that $\forall i, j \in V$, we have $e_{i,j} = e_{j,i}$. We now state the following definitions which will be used throughout the paper.

Definition 1 *Given graph $G(V, E)$, a walk from vertex $i \in V$ to another vertex $j \in V$ is a finite sequence of edges $((i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m))$ for some $m > 0$ such that $i_1 = i$, $j_m = j$, and $j_k = i_{k+1}$ for some $k \in \{1,2,\ldots m-1\}$. Given a walk $((i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m))$, the nodes $\{i_1, i_2, \ldots, i_m, j_m\}$ comprise the vertex sequence of the walk. A path from vertex i to vertex j is a walk in which all elements of its vertex sequence are distinct, and the first and last vertices of the sequence are i and j respectively. The vertex i is the start of the path, while vertex j is the end of the path. A path is said to pass through a set of vertices, if the set of vertices correspond to a subset of vertices of the path's vertex sequence. If there exists a path from i to j, then vertices i and j are connected. Two paths from i to j are edge-disjoint if their edges do not share common endpoints aside from the start and end vertices, i.e. no common endpoints except for $\{i, j\}$.*

Definition 2 *Given graph $G(V, E)$, let $S \subseteq V$ denote a subset of vertices with weight x_e associated to each edge $e \in E$. We define $\delta(S)$ as corresponding to the set of edges $\{e_{i,j} \in E\}$ such that $i \in S$ and $j \in V - S$. In other words, $\delta(S)$ denotes the set of edges with one endpoint in S and another endpoint in $V - S$. Let x be a set of weights, where each edge $e \in E$ is given a weight x_e. Given $S, S' \subseteq V$, the flow of S to S' refers to the sum of weights of edges with one endpoint in S and another endpoint in S'.*

Definition 3 *Given graph $G(V, E)$, let $S \subseteq V$ denote a subset of vertices. The subgraph $G_S(S, E')$ induced by S is defined as the graph whose set of vertices is S, and whose set of edges E' is such that for each $e_{i,j} \in E'$, we have $e_{i,j} \in E$, $i \in S$, and $j \in S$.*

Definition 4 *Given graph $G(V, E)$, let $v \in V$ be any vertex, and let $S \subseteq V$ be a subset of vertices. The set S is connected to v if $v \in S$, and for each $i \in S$ such that $i \neq v$, we have that i is connected to v in the subgraph $G_S(S, E')$ induced by S. Similarly, given a set $U \subseteq V$ and let $S \subseteq V$ be subsets of vertices. The set S is connected to U if $U \subseteq S$, and for each $i \in S$ such that $i \notin U$, we have that i is connected to each of the vertices of S.*

Definition 5 *Given graph $G(V, E)$, let $v \in V$. If a certain subset of V is given the notation $\Delta_v \subseteq V$, this means that Δ_v is connected to v. Similarly, given a set $S \subseteq V$, the subset Δ_S denotes a set of vertices that are connected to the set of vertices S.*

Definition 6 *A tree $T(V, E)$ is an undirected graph in which for each $i, j \in V$, vertices i and j are connected by exactly one path. An undirected rooted tree $T(r, V, E)$ is a tree in which one special vertex $r \in V$ is designated as a root. Given a root r, let $((v_0, v_1), (v_1, v_2), \ldots, (v_k, r))$ be any path from some $v^0 \in V$ to r. This path defines a linear order $v_0 \prec v_1 \prec v_2 \ldots \prec v_k \prec r$, in which v_j is the parent of v_{j-1}, and v_{j-1} is the child of v_j for $j \in \{1,2,..,k+1\}$ with $v_{k+1} = r$. Given any parent vertex v_j for some $j > 0$ that corresponds to the endpoint of an edge belonging to some path towards r, the set C denotes the set of all child vertices of v_j, which is the union of all the children of v_j given all possible paths towards r that passes through v_j. A vertex $v_l \in V$ is a leaf if for any path that passes through v_l and ends at the root r, it is the case that v_l has a parent (if v_l is not itself the root), but has no child.*

2.1 Connectivity Requirement Functions

The definitions below describe connectivity requirement functions introduced in the standard survivable network design problem. A connectivity requirement function provides a lower bound on the number of edge-disjoint paths between a pair of vertices of graph.

Definition 7 *Given graph $G(V, E)$, an edge cost function is a non-negative function $c: E \to \mathbb{Q}^+$ that provides a cost c_e for each edge in $e \in E$. The connectivity requirement $r_{i,j}$ for vertices $i, j \in V$ is a number that represents a lower bound on the number of edge-disjoint paths from i to j. The connectivity requirement function f, is a function that assigns a lower bound requirement on the number of edges of $\delta(S)$ for each $S \subseteq V$. In particular for each $S \subseteq V$, $f(S)$ is defined as: $f(S) = max_{i \in S, j \notin S} r_{i,j}$.*

2.2 Submodular and Weakly Supermodular Functions

These definitions describe properties of weakly supermodular functions, which are used in (Jain, 2001) to provide a 2-approximation algorithm for the standard survivable network design problem. In particular, it will be shown later in the paper how weakly supermodular functions facilitate the construction of a laminar collection of vertex sets – which would prove to be crucial in the proof for Theorem 1 below. These definitions follow (Williamson and Shmoys, 2011) and (Vazirani, 2013).

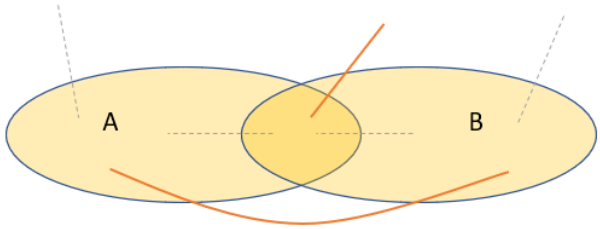
Definition 8 *Given graph $G(V, E)$, a function $g: 2^V \to \mathbb{Z}^+$ is submodular if $g(V) = 0$, and for every two sets $A, B \subseteq V$, the following two conditions hold:*

1. $g(A) + g(B) \geq g(A \cap B) + g(A \cup B)$
2. $g(A) + g(B) \geq g(A - B) + g(B - A)$

Definition 9 *Given graph $G(V, E)$, a function $g: 2^V \to \mathbb{Z}^+$ is weakly supermodular if $g(V) = 0$, and at least one of the following conditions hold:*

1. $g(A) + g(B) \leq g(A \cap B) + g(A \cup B)$
2. $g(A) + g(B) \leq g(A - B) + g(B - A)$

In particular, it can be shown that the function δ shown in Definition 2 is submodular, while the connectivity requirement function f shown in Definition 7 is weakly supermodular. To see this fact for δ, following (Vazirani, 2013), we consider the following graph whereby sets A and B have a non-empty intersection (if A and B do not intersect then δ is trivially submodular).



Here, edges with an endpoint in $A \cap B$ and another in $\overline{A \cup B}$ are counted in both $\delta(A)$ and $\delta(B)$ but not in $\delta(A - B)$ or in $\delta(B - A)$. In addition, edges with an endpoint in $A - B$ and another in $B - A$ are counted in both $\delta(A)$ and $\delta(B)$ but not in $\delta(A \cap B)$ or $\delta(A \cup B)$. v On the other hand, the connectivity requirement function f is weakly supermodular. The proof for the weakly

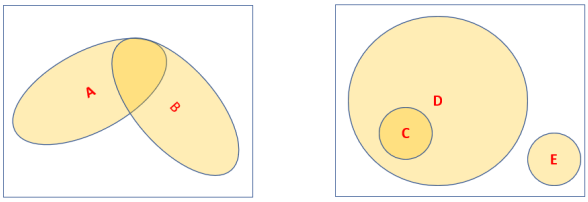
supermodular properties of f are shown in Section 6.1 of the Appendix.

2.3 Laminar and Crossing Sets

The algorithm of (Jain, 2001) used a laminar collection sets for proving the existence of an edge with weight at least $1/2$ at each iteration of the iterative rounding algorithm. The definition of a laminar collection of sets below follow the presentation in (Williamson and Shmoys, 2011) and (Vazirani, 2013).

Definition 10 *Given graph $G(V,E)$, two sets $A \subseteq V$ and $B \subseteq V$ are said to cross if each of the sets $A-B$, $B-A$ and $A \cap B$ is nonempty. A collection $\mathcal{L}$ of subsets of V is laminar if no pair of sets $A, B \in \mathcal{L}$ cross. This implies that if $A, B \in \mathcal{L}$, and $\mathcal{L}$ is laminar, then either A and B are disjoint, or one is contained in another.*

As an illustration for the definition of laminarity, let A, B, C, D, E be sets such that $A \cap B$ is nonempty. In the figure below, the collection of sets A and B cross and is therefore not laminar. On the other hand, the collection of sets C, D and E are laminar given that no two sets cross.



If a collection of sets is laminar, certain nice properties would result. In particular, under a laminar collection of sets, proofs involving counting arguments could be easily applied. An illustration of a counting argument involving a laminar collection of sets is shown in Section 6.2 of the Appendix which shows that there has to be an edge with weight at least $1/2$ in order to avoid a contradiction whereby the number of edges counted would exceed the number of edges in the solution. In addition, Lemma 11 of Section 6.3 of the Appendix applies a similar counting argument over a laminar collection of sets in order to prove the main Theorem of this paper which is Theorem 3 below.

3. Standard Survivable Network Design

We now describe the standard survivable network design problem as shown in (Jain, 2001). To avoid repetition, this paper assumes that all input graphs $G(V,E)$ from this point onwards have costs c_e associated to each edge $e \in E$.

3.1 Problem Description

Given an input graph $G(V,E)$ with costs c_e associated to each edge $e \in E$, the standard survivable network design problem is as follows (OP 1).

$$\min \sum_{e \in E} c_e x_e$$
$$\text{subject to}$$
$$\forall S \subseteq V: \sum_{e \in \delta(S)} x_e \geq f(S)$$
$$\forall e \in E: x_e \in \{0,1\}$$

A *relaxation* of OP 1 which allows for non-integer solutions could be solved in polynomial-time using standard linear programming algorithms. This relaxation of OP 1 is now shown as OP 2 which is as follows.

$$\min \sum_{e \in E} c_e x_e$$
$$\text{subject to}$$
$$\forall S \subseteq V: \sum_{e \in \delta(S)} x_e \geq f(S)$$
$$\forall e \in E: x_e \geq 0$$

A possible source of exponential time consumption in solving OP 2 is the constraint $\forall S \subseteq V: \sum_{e \in \delta(S)} x_e \geq f(S)$ since it requires checking constraint compliance for each subset $S \subseteq V$. However, as mentioned, this hurdle could be solved using the ellipsoid method, (Williamson and Shmoys, 2011), (Vazirani, 2013).

3.2 Existing Approximation Algorithm

Following (Williamson and Shmoys, 2011), an α - *approximation algorithm* for an optimization problem is an efficient (i.e. polynomial-time relative to the size of the input) algorithm that for all instances of the problem produces a solution whose value is within a factor of α from the value of an optimal solution. For the Theorems and Definitions below, we make an assumption over the edges of the input graph after solving for a feasible solution to OP 2. Namely, we assume that all edges in the input graph G' to OP 2 with zero weight under some solution x are removed. This results in a *residual graph* where we have $x_e > 0$ for each $e \in E$. With this assumption, we define the following.

Definition 11 *Given an optimization problem with m linear inequalities, a feasible solution is a solution instance for the problem where no constraint is violated. A solution is a basic feasible solution if it satisfies m linearly independent inequalities with equality. A basic feasible solution cannot be written as the convex combination of two other feasible solutions.*

Definition 12 *Given input graph $G(V,E)$ to OP 2, let x be a feasible solution, where $x := \{x_e\}_{e \in E}$. A set $S \subseteq V$ is tight with respect to connectivity requirements given by f if $\sum_{e \in \delta(S)} x_e = f(S)$.*

Definition 13 *Given input graph $G(V,E)$ to OP 2, let x be a feasible solution, where $x := \{x_e\}_{e \in E}$. The characteristic vector χ_S corresponding to $S \subseteq V$ is a vector in $\{0,1\}^{|E|}$ where each coordinate of χ_S is mapped to a particular edge $e \in E$. If given $e \in E$ we have $e \in \delta(S)$, the value of the coordinate corresponding to e in χ_S is 1. Otherwise, it is 0.*

With the above Definitions, we now state the 2-approximation algorithm from (Jain, 2001) as follows.

Algorithm 1 Iterative Rounding for Standard Survivable Network Design

$F \leftarrow \emptyset$
$i \leftarrow 1$
while F is not a feasible solution **do**
  Solve OP 2 via linear programming on edge set $E - F$ with function f_i to obtain solution x, where: $f_i(S) = f(S) - |\delta(S) \cap F|$
  remove all edges $\{e \in F : x_e = 0\}$
  $F_i \leftarrow \{e \in E - F : x_e \geq 1/2\}$
  $F \leftarrow F \cup F_i$
  $i \leftarrow i + 1$
end while
return F

Algorithm 1: Iterative Rounding for Standard Survivable Network Design

The proof that Algorithm 1 is a 2-approximation algorithm relies on the main Theorem in (Jain, 2001), which is stated as Theorem 1 below.

Theorem 1 *(Jain, 2001): For any weakly supermodular function* $\boldsymbol{f}$*, any basic feasible solution* $\boldsymbol{x}$ *to OP 2 has an element whose value is at least* $\mathbf{1/2}$*, i.e.* $\boldsymbol{x_e \geq 1/2}$ *for at least one edge* $\boldsymbol{e \in E}$*.*

In order to establish Theorem 1, (Jain, 2001) first established results stated in Theorem 2 (below) which states the important fact that if $\boldsymbol{f}$ is weakly supermodular, then in a basic feasible solution $\boldsymbol{x}$ to the standard survivable network design problem of OP 2, there is a laminar collection $\mathcal{L}$ of tight sets. As mentioned before, if a collection of sets is laminar, proofs involving counting arguments could be applied as described in Section 6.2 of the Appendix which establishes the existence of at least one edge with weight $\mathbf{1/2}$ as stated in Theorem 1.

Theorem 2 *(Jain, 2001): Given input graph* $\boldsymbol{G(V, E)}$ *to OP 2, let* $\boldsymbol{f}$ *be a weakly supermodular function* $\boldsymbol{f}$ *for connectivity requirements, and let* $\boldsymbol{x}$ *be any basic feasible solution. Given* $\boldsymbol{x}$*, there is a collection* $\mathcal{L}$ *of subsets of vertices of* $\boldsymbol{V}$ *with the following properties:*

1. for all $\boldsymbol{S \in \mathcal{L}}$, $\boldsymbol{S}$ is tight.
2. the characteristic vectors $\boldsymbol{\chi_S}$ for all $\boldsymbol{S \in \mathcal{L}}$ are linearly independent.
3. $\boldsymbol{|\mathcal{L}| = |E|}$, where $\boldsymbol{E}$ is the set of edges of the new graph $\boldsymbol{G'(V, E)}$, where all edges $\boldsymbol{e}$ in the original input graph with $\boldsymbol{x_e = 0}$ are removed.
4. $\mathcal{L}$ is laminar.

The condition that f is weakly supermodular and the fact that δ is submodular is important to establish laminarity of $\mathcal{L}$ in Theorem 2 above. From (Jain, 2001) the laminar collection $\mathcal{L}$ is constructed from an initial non-laminar collection of sets using an iterative uncrossing process. Namely, any pair of crossing sets could be effectively replaced by a pair of sets that are laminar. For instance, suppose that A and B are two sets in $\mathcal{L}$ that cross. Following Lemma 23.14 of (Vazirani, 2013), we have that one of the following must hold: (1) $A - B$ and $B - A$ are both tight and $\chi_A + \chi_B = \chi_{B-A} + \chi_{A-B}$ or (2): $A \cup B$ and $A \cap B$ are both tight and $\chi_A + \chi_B = \chi_{A \cup B} + \chi_{A \cap B}$ given that f is weakly supermodular. This implies that given crossing sets A and B, we can keep the optimal solution by uncrossing A and B and replacing them with the laminar pair of sets $A - B$ and $B - A$ if (1) holds. On the other hand, if (2) holds, then A and B can be replaced by the laminar pair of sets $A \cup B$ and $A \cap B$. To see that either (1) or (2) holds, following (Vazirani, 2013), we use the weakly supermodular property of f, whereby we either have: $f(A) + f(B) \leq f(A - B) + f(B - A)$ or $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$. If the former holds, given that A and B are both tight sets under the basic feasible solution x, we have $\sum_{e \in \delta(A)} x_e + \sum_{e \in \delta(A)} x_e = f(A) + f(B)$. Moreover, given that x is feasible, we have $\sum_{e \in \delta(A-B)} x_e + \sum_{e \in \delta(B-A)} x_e \geq f(A - B) + f(B - A)$. Given that $f(A) + f(B) \leq f(A - B) + f(B - A)$, this leads to $\sum_{e \in \delta(A)} x_e + \sum_{e \in \delta(A)} x_e \leq \sum_{e \in \delta(A-B)} x_e + \sum_{e \in \delta(B-A)} x_e$. But from the submodularity of δ, the left hand side of the prior equation is at least greater than the right hand side given that $\delta(A)$ and $\delta(B)$ also count edges with one endpoint in $\overline{A \cup B}$ and another in $A \cap B$. Hence, we should have $\sum_{e \in \delta(A)} x_e + \sum_{e \in \delta(A)} x_e = \sum_{e \in \delta(A-B)} x_e + \sum_{e \in \delta(B-A)} x_e$. However, this equality could only be met if edges with one endpoint in $\overline{A \cup B}$ and another in $A \cap B$ is given zero weight in x. Hence, $\chi_A + \chi_B = \chi_{B-A} + \chi_{A-B}$ under the condition that $f(A) + f(B) \leq f(A - B) + f(B - A)$. A similar result applies if $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$ holds instead.

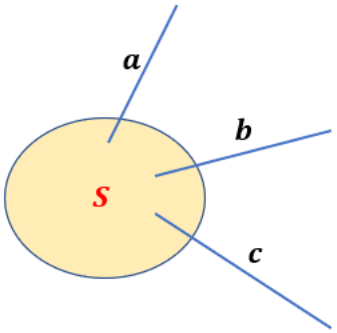
Aside from the weakly supermodular property of f and the submodular property of δ, construction of a laminar collection of sets such that $|\mathcal{L}| = |E|$ also requires the first two properties of tightness and linear independence. More details are explained in Section 6.2 of the Appendix. For the problem considered in this paper which incorporates the standard survivable network design problem, an analogous result regarding the uncrossing process of tight sets to form a laminar collection is described in Lemmas 4-8 of Section 6.3 of the Appendix.

4. Survivable Network Design with Constrained ɧ-subgraph Flows

In this section, we now describe our proposed variant of the survivable network design problem such that outgoing flows in ɧ -subgraphs are constrained. We first describe a list of conditions for an input graph with its set of ɧ-subgraphs to be considered as a valid input to our proposed problem. We also indicate conditions for the validity of the connectivity requirement function f and the flow constraint function z. Finally, we state our proposed problem in terms of an optimization problem OP 3.

To explain the motivation behind our proposed problem, recall that flow constraint functions provides bounds on the weight of edges in graph cuts. For instance, the maximum flow problem is equivalent to the minimum cut problem, which could be efficiently solved using the Ford-Fulkerson algorithm (Williamson and Shmoys, 2011). In this paper, we consider the converse problem of providing upper bounds on the weight of edges in graph cuts, whereby the upper bound is provided by the function z. For instance, in the figure below, if $z(S) = 1$, then

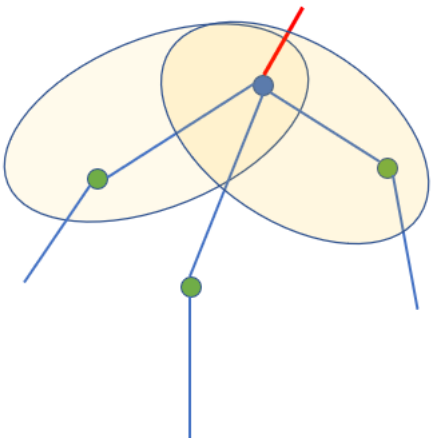
the sum of weights for edges a, b and c has to be less than 1, where a, b, and c are the outgoing edges of vertex set S.



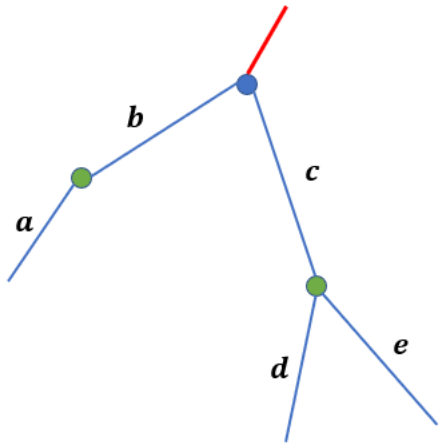
e,

In a general subgraph however with set of vertices V, if the function z is defined for each subset $S \subseteq V$, then the process of checking if each possible subset of vertices $S \subseteq V$ complies with z could not be done in polynomial time given that the number of subsets of vertices of V is exponential. A possible workaround for this could be to check for the subset S that provides the maximum cut under a constrained constant z. If such a subset S is found to comply with a constant z in the sense that the weights of its outgoing edges is less than $z(S)$, where $z(S)$ is constant for all $S \subseteq V$, then all other subsets of V would likewise comply with z. However, this could not be done in polynomial time given that the maximum cut problem is NP-complete (Ausiello et al., 2012).

Given the above difficulties, the problem in this paper does not opt to consider general subgraphs with a general flow constraint function z defined for all subsets of vertices of the subgraph. Instead, specific instances of subgraphs termed ɧ-subgraphs are considered along with a special type of flow constraint function z that is defined only for certain subsets of the ɧ-subgraph. Briefly, a ɧ-subgraph can be described as having a tree structure (conditions [G1]-[G4]) below, with branches that are described in the following figure, where the vertices of the branch are small green circles along with one small blue circle which is termed a "trunk vertex".



Likewise, the flow constraint function is defined only for specific subsets of the ɧ-subgraph, such as the subsets of vertices that are encircled in the figure above. In addition, z is required to be constant for all subsets of vertices in a branch for which it is defined. These are described in conditions [Z1]-[Z3] pertaining to z below. Added to this, the optimization problem (OP 3) is constructed in such a way that given a branch in a ɧ-subgraph, it has to be the case that edges closer to a special vertex termed the "trunk vertex" have to have higher weight than the sum of edges below it. For instance, consider the branch illustrated in the following figure. Here, suppose that the weights of b and c are both 0.5 each. Under the constraints of OP 3, a feasible solution has to set the weight of a as less than or equal to 0.5 and the sum of weights of d and e as less than or equal to 0.5.



Given conditions [G1]-[G4] pertaining to ɧ-subgraphs and conditions [Z1]-[Z3] pertaining to the flow constraint function z, along with the constraints of OP 3, it could now be easily checked in polynomial time if each subset of vertices of a branch of the ɧ-subgraph complies with the flow constraint function z by simply checking the weight of edges connected to trunk vertices. For instance, in the figure above, given that edges b and c have larger weight than the sum of edges below it, if the sum of both b and c is less than or equal to the constraint dictated by z, then all possible subsets of vertices in the branch also comply with z given that z is constant for all subsets of vertices in the branch. This is the intuition behind the separation oracle for the second constraint of OP 3 which is described below.

4.1 Additional Definitions

Definition 14 *Given input graph $G(V, E)$, let x be a set of edge weights, i.e. $x := \{x_e\}_{e \in E}$. A flow constraint function $z: 2^V \times 2^V \to \mathbb{N}$, applies the following upper bound to each $S, S' \subseteq V$:*

$$\sum_{e \in E} x_e \leq z(S, S') \quad \text{where } e \text{ has one endpoint in } S \text{ and another in } S' \qquad (1)$$

In some cases, the flow constraint function z is designed such that for some $S \subset V$, only a subset of edges $\boldsymbol{e} \subseteq \delta(S)$ with one endpoint in S and another endpoint in some pre-defined $S' \subseteq V$ have finite upper bounds. All other edges $e \in \delta(S) \wedge e \notin \boldsymbol{e}$ have infinite upper bounds. Under this context, the flow constraint function for S is simplified to $z: 2^V \to \mathbb{N}$, and is denoted as $z(S)$ for ease of notation as the corresponding S' is already given and understood for each $S \subseteq V$. For this paper, we adopt this convention in defining z for all $S \subseteq V$ as we will specify beforehand the specific edges in $\delta(S)$ that would have finite upper bounds under z.

Definition 15 *Given input graph $G(V, E)$, let $S \subseteq V$ be a subset of vertices. The flow constraint function z is uniform over S if for each pair of subsets $S' \subseteq S$ and $S'' \subseteq S$, we have $z(S') = z(S'')$.*

Definition 16 *Given input graph $G(V, E)$, let x be a set of edge weights. A set $S \subseteq V$ is tight with respect to flow constraints given by z if $\sum_{e \in \delta(S)} x_e = z(S)$ where e has one endpoint in S and another endpoint in some pre-defined S'.*

4.2 Problem Components

This subsection provides definitions that describe properties of ɧ-subgraphs by means of conditions. An illustration of a ɧ-subgraph is shown in Fig. 1. Conditions for flow constraint functions and connectivity requirement functions to be valid or admissible are also stated.

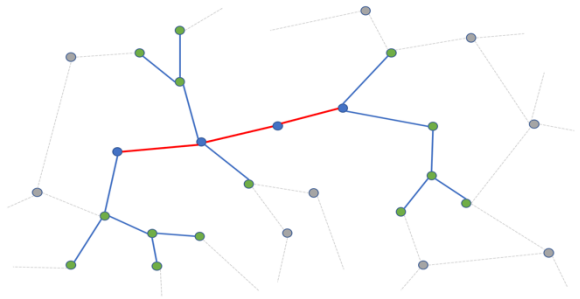


Figure 0: Figure illustrating a ɧ-subgraph inside an input graph to the survivable network design problem. The trunk of the subgraph has trunk vertices (blue circles) that are connected by a unique path (edges of the trunk's unique path are marked as red lines). The trunk vertices serve as roots of branches whose sets of vertices (green circles) are disjoint. All other vertices that do not belong to the ɧ-subgraph are gray circles.

Definition 17 *Given a graph $\boldsymbol{G(V,E)}$, a subgraph of $\boldsymbol{G}$ can be considered as a $\boldsymbol{ɧ}$-subgraph if it meets conditions G1-G4 defined below. The collection of sets of vertices of all $\boldsymbol{ɧ}$-subgraphs in $\boldsymbol{G}$ is $\boldsymbol{\mathcal{H}}$.*

1. **condition [G1]: $\forall \boldsymbol{B}, \boldsymbol{B'} \in \boldsymbol{\mathcal{H}}$, we have $\boldsymbol{B \cap B' = \emptyset}$, i.e. $\boldsymbol{\mathcal{H}}$ is a disjoint collection of subsets of $\boldsymbol{V}$.**

2. **condition [G2]:** $\forall B \in \mathcal{H}$, the subgraph $G_B(B, E')$ induced by B is composed of the following:

   (a) A *trunk* consisting of a set of *trunk vertices* $\{\boldsymbol{v_0^t}, \boldsymbol{v_1^t}, .., \boldsymbol{v_k^t}\} \subset \boldsymbol{B}$ for some $\boldsymbol{k > 0}$ and *trunk edges* $((\boldsymbol{v_0^t}, \boldsymbol{v_1^t}), (\boldsymbol{v_1^t}, \boldsymbol{v_2^t}), \ldots, (\boldsymbol{v_{k-1}^t}, \boldsymbol{v_k^t})) \subset \boldsymbol{E'}$ that define a unique path connecting each pair of trunk vertices. The notation $\boldsymbol{v^t}$ refers to some arbitrary trunk vertex of $\boldsymbol{B}$.

   (b) Given $\boldsymbol{k}$ trunk vertices in $\boldsymbol{B}$, $\boldsymbol{G_B(B, E')}$ also contains a set of disjoint *branches* consisting of rooted trees $\{\boldsymbol{T(v_i^t, V_T, E_T)}\}_{\boldsymbol{i \in [k]}}$ in which the root $\boldsymbol{v_i^t \in V_T}$ of each branch is a trunk vertex.

3. **condition [G3]:** $\forall B \in \mathcal{H}, \forall v^t \in B$, there is no edge in E connecting any trunk vertex v^t to $V - B$.

4. **condition[G4]:** $\forall B \in \mathcal{H}, \forall T(v^t, V_T, E_T)$, there is no edge connecting any $i \in V_T$ with $i \neq v^t$ to any other trunk vertex $v^{t\prime} \in B$ such that $v^{t\prime} \neq v^t$.

Definition 18 *A pair consisting of a graph $G(V,E)$ and a collection $\mathcal{H}$ of subsets of V is admissible if each set of vertices in $\mathcal{H}$ complies with conditions G1-G4. It follows that the set of subgraphs induced by the sets of vertices of $\mathcal{H}$ is the set of ɧ-subgraphs of G.*

Definition 19 *Given an admissible graph $G(V,E)$ with admissible collection $\mathcal{H}$ of subsets of V, for some $B \in \mathcal{H}$, let $T(v^t, V_T, E_T)$ be a branch whose root is some trunk vertex $v^t \in B$. Let $\Delta_{v^t} \subseteq V_T$ be any subset of V_T that is connected to V. We define the set $\tau(\Delta_{v^t})$ as corresponding to the set of edges with one endpoint in Δ_{v^t} and another endpoint in either $V - B$, or in $V_T - \Delta_{v^t}$, i.e. this set is equivalent to the set of edges in $\delta(\Delta_{v^t})$ less the trunk edges for which v^t represents an endpoint.*

Definition 20 *Given an admissible graph $G(V,E)$ with admissible collection $\mathcal{H}$ of subsets of V, the collection $\mathcal{H}$ is p-admissible if for any branch $T(v^t, V_T, E_T)$ with root $v^t \in B$ for any $B \in \mathcal{H}$, the maximum length of any path in E_T from v^t to a leaf vertex in V_T is at most p.*

Definition 21 *Given an admissible graph $G(V,E)$ with admissible collection $\mathcal{H}$ of subsets of V, the collection $\mathcal{H}$ is l-admissible if for any branch $T(v^t, V_T, E_T)$ with root $v^t \in B$ for any $B \in \mathcal{H}$, the maximum number of leaves in the tree $T(v^t, V_T, E_T)$ is at most l.*

Definition 22 *Given an admissible graph $G(V,E)$ with admissible collection $\mathcal{H}$ of subsets of V, the collection $\mathcal{H}$ is pl-admissible if $\mathcal{H}$ is both p-admissible and l-admissible.*

Definition 23 *Given an admissible graph $G(V,E)$ with an admissible collection $\mathcal{H}$ of subsets of V, let x be a set of edge weights. A flow constraint function $z: 2^V \rightarrow \mathbb{N}$ is admissible if it meets the following conditions.*

1. **condition [Z1]:** $\forall B \in \mathcal{H}$, $\forall v^t \in B$ given the branch $T(v^t, V_T, E_T)$, z is uniform over all subsets $\Delta_{v^t} \subseteq V_T$ that are connected to v^t.

2. **condition [Z2]:** $\forall B \in \mathcal{H}$, $\forall v^t \in B$, given the branch $T(v^t, V_T, E_T)$ and a subset $\Delta_{v^t} \subseteq V_T$ connected to v^t, we have:

$$\sum_{i \in \Delta_{v^t}, j \in V - B} x_{e_{i,j}} + \sum_{i \in \Delta_{v^t}, j \in V_T - \Delta_{v^t}} x_{e_{i,j}} \leq z(\Delta_{v^t}) \quad (2)$$

In this case, the pre-defined set S' that is associated with Δ_{v^t} for which $z(\Delta_{v^t})$ provides finite upper bounds for all edges with one endpoint in Δ_{v^t} and another endpoint in S' is $S' := (V - B) \cup (V_T - v^t)$. Thus, the above equation could be simplified to:

$$\sum_{e \in \tau(\Delta_{v^t})} x_e \leq z(\Delta_{v^t}) \quad (3)$$

3. **condition [Z3]:** For all other subsets $S \subseteq V$ that are not subsets $\Delta_{v^t} \subseteq V_T$ connected to the root v^t of some branch $T(v_i^t, V_T, E_T) \subset G_B(B, E')$ of some $B \in \mathcal{H}$, we have $z(S) = \infty$, i.e. no flow constraint is imposed.

From the definition of z above, for any vertex subset $\Delta_{v^t} \subseteq V_T$, for any branch $T(v^t, V_T, E_T)$ with root $v^t \in B$ with $B \in \mathcal{H}$, we have under condition $Z2$ that z actually provides an upper bound on $\tau(\Delta_{v^t})$. This is due to conditions G3-G4 of $\mathcal{H}$.

Definition 24 *Given an admissible graph $G(V,E)$ with an admissible collection $\mathcal{H}$ of subsets of V, let x be a set of edge weights. A connectivity requirement function $f: 2^V \rightarrow \mathbb{N}$ is admissible if it meets the following conditions:*

1. **condition [F1]:** $\forall B \in \mathcal{H}, \forall i \in B, \forall j \in V - \{i\}: r_{i,j} = 0$, i.e. there are zero connectivity requirements between any vertex in B to any other vertex.

2. **condition [F2]:** f is weakly supermodular.

4.3 Optimization Problem 3

The survivable network design with constrained ɧ-subgraph flows problem requires an admissible input graph $G(V,E)$, a pl-admissible collection $\mathcal{H}$ of subsets of V, an admissible connectivity requirement function f and admissible flow constraint function z. Given these inputs, the problem is termed optimization problem 3 (OP 3) and includes an additional constraint (constraint C2), whereby for each branch $T(v^t, V_T, E_T)$ with root $v^t \in B$ for some $B \in \mathcal{H}$, the problem requires that for any vertex in $v \in V_T$, the sum of weights of edges from v to its children should be less than or equal to the weights of the edge connecting v to its own parent. In this case, a vertex i is a child of v if there is a path from i to the root v^t whose first edge is $e_{i,v}$. Given this constraint, OP 3 is described as follows.

Survivable network design problem with constrained ɧ-subgraph flows (relaxed form)

Input:

1. Admissible input graph $G(V,E)$ with a pl-admissible collection of sets of vertices $\mathcal{H}$
2. Admissible flow constraint function z
3. Admissible connectivity requirement function f

Solve for $\boldsymbol{x}$:

$$\min \sum_{e \in E} c_e x_e$$

subject to:

$\forall S \subseteq V$:

$$\sum_{e \in \delta(S)} x_e \geq f(S)$$

$\forall B \in \mathcal{H}, \forall v^t \in B$ given the branch $T(v^t, V_T, E_T)$ we have the following for all $\Delta_{v^t} \subseteq V_T$ that are connected to v^t:

$$\sum_{i \in \Delta_{v^t}, j \in V - B} x_{e_{i,j}} + \sum_{i \in \Delta_{v^t}, j \in V_T - \Delta_{v^t}} x_{e_{i,j}} \leq z(\Delta_{v^t})$$

$\forall B \in \mathcal{H}, \forall v^t \in B$, given the branch $T(v^t, V_T, E_T)$, let $v_p \in V_T$ be any parent vertex with set of child vertices $C \subset V_T$, and let $v_g \in V_T$ be the parent vertex of v_p. We have:

$$\sum_{i \in C} e_{i,v_p} \leq x_{e_{v_p,v_g}}$$

$\forall e \in E: x_e \geq 0$

It could be seen that the survivable network design with constrained ɧ-subgraph flows problem is at least as hard as the standard survivable network design problem. Any instance of the standard survivable network design problem could be reduced to an instance of the survivable network design with constrained ɧ-subgraph flows problem by setting the input $\mathcal{H}$ as empty. Afterwards, $z(S)$ is set to ∞ for all possible subsets $S \subset V$ of vertices of the input graph $G(V,E)$. The only remaining constraints for this reduced problem are those that refer to connectivity requirements given by f, which thereby represents the standard survivable network design problem.

4.4 Proposed Approximation Algorithm

Algorithm 2 Iterative Rounding with Relaxation for Survivable Network Design Problem with Constrained ɧ-Subgraph Flows

initialize solution set $F \leftarrow \emptyset$
set $i \leftarrow 0$
set $f_0 = f$ and $z_0 = z$
while F is not a feasible solution **do**
  $i \leftarrow i + 1$
  given z_i and $f_i(S) := f(S) - |\delta(S) \cap F|$, solve OP 3 via linear programming to obtain a basic feasible solution x on residual graph
  remove all zero-weighted edges $\{e \in F : x_e = 0\}$ from E
  if there are edges $e \in E$ with $x_e \geq 1/2$ **then**
    $F_i \leftarrow \{e \in E - F : x_e \geq 1/2\}$
    $F \leftarrow F \cup F_i$
    for all $e \in F_i$ such that $e \in \tau(\Delta_{v^t})$ for some $\Delta_{v^t} \subseteq V_T$, where V_T is the vertex set of a branch $T(v^t, V_T, E_T)$, with root $v^t \in B$ for some $B \in \mathcal{H}$, let v_e denote the endpoint of e in Δ_{v^t}. **then**
      - include in F the edges in $\boldsymbol{e}$, where $\boldsymbol{e}$ is the set of edges which belong to some path from a leaf of V_T to v^t and which passes through v_e.
      - Remove all edges in $\boldsymbol{e}$ from E.
      - for all $\Delta_{v^t} \subseteq V_T$ for which some edge in $\boldsymbol{e}$ belongs to $\tau(\Delta_{v^t})$, set $z_{i+1}(\Delta_{v^t}) \leftarrow z_i(\Delta_{v^t}) - m$, where m is the number of edges of $\boldsymbol{e}$ belonging to $\tau(\Delta_{v^t})$
  else
    there is a set $\Delta_{v^t} \subseteq V_T$, where V_T is the vertex set of a branch $T(v^t, V_T, E_T)$, with root $v^t \in B$ for some $B \in \mathcal{H}$ such that $|\tau(\Delta_{v^t})| \leq 3$ **then**
      - for all $\Delta_{v^t} \subseteq V_T : z(\Delta_{v^t}) \leftarrow \infty$
  end if
end while
return F

Algorithm 2: Iterative Rounding with Relaxation for Survivable Network Design Problem with Constrained ɧ-Subgraph Flows

Algorithm 2 above shows our proposed approximation algorithm which provides $(2pl, 2z(S)+3)$ approximation guarantee as stated in Theorem 4. The core theory for justifying this approximation guarantee is Theorem 3 below which states that at each iteration of Alg. 2, either there is at least one edge with weight of at least $1/2$, or there is some vertex set Δ_{v^t} that belongs to the branch of some $\mathcal{H}$-subgraph with at most 3 outgoing edges. The proof for Theorem 3 is shown in the Appendix which relies on Theorem 5 – the Theorem that is analogous to Theorem 1 for the standard survivable network design problem To show that Algorithm 2 operates in polynomial-time, we enumerate the needed separation oracles in Section 4.5 below.

Theorem 3 *At each iteration i of Algorithm 2, where x is a basic feasible solution for OP 3, at least one of the following occurs:*

1. *there is at least one edge e such that $x_e \geq 1/2$.*
2. *there is a set $\Delta_{v^t} \subseteq V_T$, where V_T is the vertex set of a branch $T(v^t, V_T, E_T)$, with root $v^t \in B$ for some $B \in \mathcal{H}$ such that $|\tau(\Delta_{v^t})| \leq 3$.*

To illustrate these two options, we provide Figures 1a and 1b below. Here, we slightly abuse the notation by using lowercase letters, i.e. a, b, c, etc to refer to edges. At the same time, if an edge is assigned a value using the $=$ operation, i.e. $a = 1.0$, this implies that in a solution, edge a is provided a weight of 1.0.

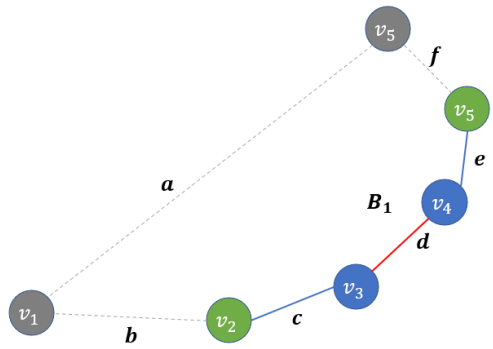


Figure 1a: Figure showing an instance whereby the algorithm chooses the first option. Here, the cost of edge a is 0.25, while the costs of edges b to f is each 1.00.

As a very simple illustration of the first option, consider Figure 1a. Here, node v_1 is required to be connected by one path to node v_5. In this case, node v_1 may use the path provided by edge a, to go to v_5 or it may use the path provided by the subgraph B_1 which goes through edges b, c, d, e, f. But given that the the cost of edge a is 0.25, while the costs of edges b to f is 1.00, the linear programming solution would assign a weight of 1.0 to edge a, while a weight of 0.0 would be assigned to edges b to f. This solution is feasible given that including edge a in the solution already satisfies the connectivity requirement that at least one path connects v_1 to v_5. Given that the weight of a is greater than 0.5, the algorithm chooses the first option and includes a in its solution set.

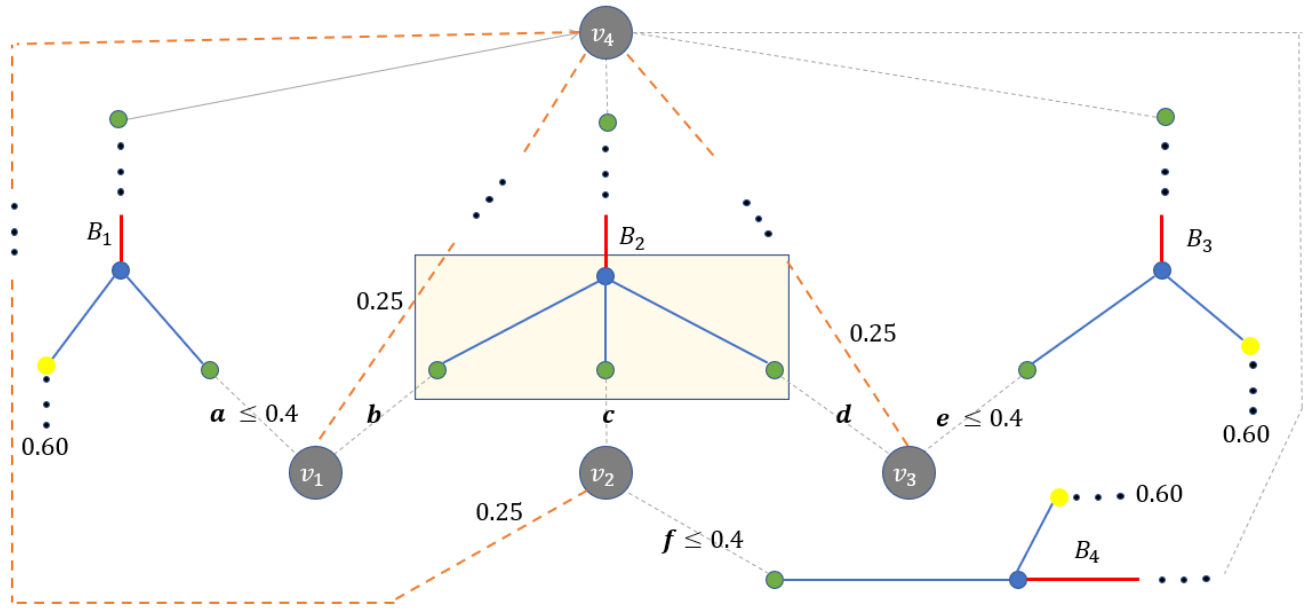


Figure 1b: Figure showing an instance whereby the Algorithm chooses the second option. Here, the costs of edges a, e and f is each 100.0, while the costs of edges b, c and d is each 0.25.

As an illustration of the second option, consider Figure 1b above. Here, vertices v_1, v_2 and v_3 are required to be connected by one path to vertex v_4 as per connectivity requirements. Vertices v_1, v_2 and v_3 can be connected to v_4 either by passing through subgraphs B_1, B_2, and B_3 respectively, or by using longer paths illustrated by dashed orange lines (which are disjoint from the three subgraphs B_1, B_2, and B_3). Suppose that in this example, subgraphs B_1, B_2, and B_3 are provided a flow constraint of 1.0 for all vertex sets connected to its trunk vertices. Suppose as well that in the optimal solution returned by linear programming, the trunk vertices of subgraphs B_1 and B_2 (blue vertices) receive a cumulative incoming weight of 0.60 coming from edges whose parents are yellow vertices. The incoming weight of 0.60 however is distributed across edges (not shown in the figure) whose individual weights are all less than 0.5. This implies that the maximum weight that can be assigned to edges a and b in the graph is 0.4 in order to comply with the 1.0 flow constraint. Suppose as well that in the optimal solution returned by linear programming, edges that belong to the orange path are allocated a weight of 0.25. This implies that v_1, v_2 and v_3 only need to allocate an additional weight of 0.75 to their outgoing edges in order to meet their connectivity requirements with v_4, or that $a + b \geq 0.75$, $c + f \geq 0.75$ and $d + e \geq 0.75$. In fact, given that the costs of edges a, e and f is each 100.0, while the costs

of edges b, c and d is each 0.25, the basic feasible solution returned by linear programming results in a solution of $a = 0.40$, $b = 0.35$, $c = 0.3$, $d = 0.35$, $e = 0.4$ and $f = 0.45$. Assuming that all other edges not shown in the figure aside from a to f all have weight less than 0.5, the algorithm may pick the vertex set Δ_{v^t} comprising of edges encircled in a lightly colored yellow box, where $v_t \in B_2$. In this case, $\tau(\Delta_{v^t}) = \{b, c, d\}$ and $|\tau(\Delta_{v^t})| \leq 3$.

Theorem 4 *Given an admissible graph $G(V, E)$, a pl-admissible collection $\mathcal{H}$ of subsets of V, an admissible connectivity requirement function f and admissible flow constraint function z, Algorithm 2 is a $(2pl, 2z(S) + 3)$ approximation algorithm for the survivable network design with constrained $\mathfrak{h}$-subgraph flows problem. Moreover, the algorithm requires at most $|E| + |V|$ iterations to terminate.*

Proof. At any iteration of Alg. 2, exactly one of the two options stated in Theorem 3 is picked by Alg. 2. If the algorithm picks the first option, then there is an edge e such that $x_e \geq 1/2$, for which Alg. 2 includes e in the solution set F. Since $x_e \geq 1/2$, upon rounding it we have $\lceil x_e \rceil \leq 2x_e$. We note that Alg. 2 only includes in F those edges with weight at least $1/2$. Following (Jain, 2001), we have that a basic feasible solution for iteration i is valid for all iterations $j > i$. To see this, let x^i be the solution at iteration i and x^j be the solution for iteration j for $i < j$. We have the following result for any pair of iterations i, j such that $i < j$:

$$\sum_{e \in E - F_j} c_e x_e^j \leq \sum_{e \in E - F_i} c_e x_e^i \qquad (4)$$

where F_i are the edges included in the solution set of edges of Algorithm 2 at iteration i (and similarly for F_j). If we apply Eq. 4 to iterations 1 to i, we arrive at the following equations which result in at least a 2-performance guarantee for Alg. 2.

$$\sum_{e \in E - F_i} c_e x_e^i \leq \sum_{e \in E - F_{i-1}} c_e x_e^{i-1} \leq \ldots \leq \sum_{e \in E - F_1} c_e x_e^1$$

$$[\sum_{e \in E - F_i} c_e x_e^i + \sum_{e \in F_{i-2} - F_{i-1}} c_e x_e^{i-1} + \ldots + \sum_{e \in F_1} c_e x_e^1] \leq [\sum_{e \in E - F_2} c_e x_e^2 + \sum_{e \in F_1} c_e x_e^1]$$

$$\sum_{e \in E - F_1} c_e x_e^1 + \sum_{e \in F_1} c_e x_e^1 \leq 2 \sum_{e \in E} c_e x_e^1 \leq 2\mathrm{OPT}$$

Under the first option, Alg. 2 checks if some e included in F_i belongs to $\tau(\Delta_{v^t})$ for some $\Delta_{v^t} \subseteq V_T$, where V_T is the vertex set of a branch $T(v^t, V_T, E_T)$, with root $v^t \in B$ for some $B \in \mathcal{H}$. If there is such an edge e that meets this condition, let v_e denote the endpoint of e in this particular Δ_{v^t}. Let p define the path starting from this endpoint v_e of e, and which ends at the root v^t. Under constraint C2 of OP 3, all edges in the path p from v_e to v^t have a weight of at least $1/2$. It follows that the algorithm then includes in F all edges of p, so that in future iterations, this path is effectively removed from the residual graph. However, paths starting from v_e to the leaves of V_T may have weight less than $1/2$. Given that the path from v_e to v^t are removed in the residual graph, it follows that all paths from v_e to the leaves of V_T have to be likewise removed in order to keep constraint C2 of OP3. For these removed edges that belong to some $\tau(\Delta'_{v^t})$, the algorithm reduces all $z(\Delta'_{v^t})$ by the respective number of edges removed from $\tau(\Delta'_{v^t})$. Now, given that $\mathcal{H}$ is admissible, V_T has at most l leaves, and the length of any path from the vertex to a leaf is at most p. It follows that at most pl edges are removed from V_T in this case. Since each edge is rounded off, this leads to a $2pl$-approximation guarantee.

If ever Alg. 2 picks the second option where there are at most 3 edges in some set $\tau(\Delta_{v^t})$ for some $\Delta_{v^t} \subseteq V_T$, and where V_T is the vertex set of a branch $T(v^t, V_T, E_T)$, with root $v^t \in B$ for some $B \in \mathcal{H}$, the algorithm merely removes the upper bound for all $S \subseteq V_T$. Suppose that this occurs at iteration i. For succeeding iterations greater than i, at most a total of 3 edges can be added to $\tau(\Delta_{v^t})$ for any $\Delta_{v^t} \subseteq V_T$, since all edges with zero weight are removed by the algorithm. Moreover, in case some edge in $\tau(\Delta_{v^t})$ was included in F at some prior iteration (since its weight was at least $1/2$), then the rounding-off procedure of Alg. 2 multiplies the weight of the edge by 2. It follows that $z(S)$ is violated by $2z(S) + 3$ for each $S \subseteq V_T$ that corresponds to some subset of vertices Δ_{v^t} connected to v^t.

It takes at most $|E| + |V|$ iterations for the algorithm to terminate, since the algorithm will definitely terminate if all edges from the original graph are removed.

4.5 Time Complexity of Algorithm 2 and Separation Oracles Separation Oracles

Lines 7-17 of Algorithm 2 could be done in polynomial time since performing these steps of the Algorithm merely involves checking if an edge has weight of at least $1/2$ or if there is a set Δ_{v^t} such that $|\tau(\Delta_{v^t})| \leq 3$. The former can be done in polynomial time given that the number of edges in the input graph is polynomial. As for the latter, given that x is a basic feasible solution, it has to be the case that if a positively weighted edge in the branch of a $\mathfrak{h}$-subgraph has one endpoint being the trunk vertex, then there has to be at least one other positively weighted edge connected to its other endpoint (following the constraints of OP 3 and the fact that x is a basic feasible solution). It follows that in order to find the set Δ_{v^t} such that $|\tau(\Delta_{v^t})| \leq 3$, the algorithm merely checks the number of outgoing edges of each trunk vertex of its $\mathfrak{h}$-subgraphs. If a certain trunk vertex is the endpoint of less than or equal to three positively weighted edges, then the algorithm includes in its candidate Δ_{v^t}, the vertices corresponding to the other endpoints of these positively-weighted edges. It then checks if $|\tau(\Delta_{v^t})| \leq 3$. If this condition is met, then the candidate Δ_{v^t} is the set that is desired. If not, this implies that all other candidate subsets Δ_{v^t} in this branch of the $\mathfrak{h}$-subgraph have $|\tau(\Delta_{v^t})| \geq 4$ (from the constraint of OP 3 and the fact that the branch has a tree structure). In this case, the algorithm moves on to the next trunk vertex. Given that the number of trunk vertices is polynomial, the procedures in this step is performed in polynomial time. This leaves us with line 6 of Algorithm 2 which performs linear programming over an exponential number of constraints of OP 3. However, from (Williamson and Shmoys, 2011), this linear program could be solved in polynomial time using the ellipsoid method – as long as polynomial-time separation oracles are provided which indicate if a certain constraint of OP 3 is violated.

For this purpose, the following are polynomial-time separation oracle methods to check constraint violations for OP 3.

1. (Constraint C0): $\forall S \subseteq V: \sum_{e \in \delta(S)} x_e \geq f(S)$
   The separation oracle for this set of constraints is the same as the separation oracle for the standard survivable network design problem described in (Jain, 2001), (Williamson and Shmoys, 2011).

2. (Constraint C2):
   $\forall B \in \mathcal{H}, \forall v^t \in B,$
   given the branch $T(v^t, V_T, E_T)$, let $v_p \in V_T$ be any parent vertex with set of child vertices $C \subset V_T$, and let $v_g \in V_T$ be the parent vertex of v_p.

$$\text{We have: } \sum_{i \in C} e_{i,v_p} \leq x_{e_{v_p,v_g}}$$

The separation oracle for this set of constraints is as follows. For each $B \in \mathcal{H}$, and for each branch $T(v^t, V_T, E_T)$ with root $v^t \in B$, the oracle computes for each vertex $v \in V_T$ the sum of weights all edges with one endpoint being v, and another endpoint being a child of v (where a vertex i is a child of v if there is a path from i to the root v^t, whose first edge is $e_{i,v}$). Let this sum be denoted as x. Afterwards, the oracle checks if the weight of the edge from v to its own parent (if it has one) is at least as large as x. If this condition is not met, it follows that there is a violation of constraint C2.

3. (Constraint C1):
   $\forall B \in \mathcal{H}, \forall v^t \in B$ given the branch $T(v^t, V_T, E_T)$ we have the following for all $\Delta_{v^t} \subseteq V_T$ that are connected to v^t:
$$\sum_{i \in \Delta_{v^t}, j \in V - B} x_{e_{i,j}} + \sum_{i \in \Delta_{v^t}, j \in V_T - \Delta_{v^t}} x_{e_{i,j}} \leq z(\Delta_{v^t})$$

The separation oracle for this set of constraints is as follows. The oracle first checks if no violation has been discovered by the separation oracle for constraint C2. If no violation is seen, this implies that the sum of weights of edges with one endpoint in v^t and another in $V_T - \{v^t\}$ represents the maximum value of $\sum_{i \in \Delta_{v^t}, j \in V - B} + \sum_{i \in \Delta_{v^t}, j \in V_T - \Delta_{v^t}}$ for any $\Delta_{v^t} \subseteq V_T$. Let this sum representing the maximum value be denoted as x. Given condition Z1 of z, we have that $z(\Delta_{v^t})$ is constant for all $\Delta_{v^t} \subseteq V_T$, and therefore could be represented by some constant θ. The oracle simply checks if $x \leq \theta$. If not, this implies that constraint C1 is violated.

5. Conclusion

In this paper, we presented a variant of the survivable network design problem which incorporates flow constraints over subgraphs, where these flow constraints are in the form of upper bounds over weights of outgoing edges of vertex subsets of the subgraphs. While the general problem of verifying if a certain set of edges in a graph cut of an arbitrary subgraph violates an upper bound is computationally difficult, our proposed problem considers certain types of subgraphs termed ɧ-subgraphs. These subgraphs have a tree structure composed of a trunk and several branches. Under this graph structure, along with certain flow conditions on flow constraints and edge weights, we construct an efficient procedure that checks if a subset of outgoing edges of a vertex subset of the subgraph violate the upper bound provided by the flow constraint. This allows for the construction of a polynomial-time algorithm based on iterative rounding and linear programming, where the linear programming component of the algorithm makes use of polynomial-time separation oracles under the ellipsoid method. Our proposed algorithm has $(2pl, 2z(S) + 3)$-performance where p is the maximum length of any path that connects the root and leaves of any branch of a ɧ-subgraph, while l is the maximum number of leaves of any branch of any ɧ-subgraph, and where S is a subset of a branch of a ɧ -subgraph that is subjected to flow constraints. The approximation guarantee of our algorithm relies on the fact that at each iteration of the algorithm, there may either be at least one edge with weight of at least $1/2$, or there is a vertex subset of some branch of a ɧ-subgraph such that the number of outgoing edges of the vertex subset is at most 3.

REFERENCES

AUSIELLO, GIORGIO, et al. Complexity and approximation. 2012. Combinatorial optimization problems and their approximability properties. Springer Science & Business Media

BATRA J, GARG N, KUMAR A, MOMKE T, WIESE A. 2014. New approximation schemes for unsplittable flow on a path. In Proceedings of the twenty-sixth annual ACM-SIAM symposium on discrete algorithms. p. 47–58. SIAM.

BLAND RG, GOLDFARB D, TODD MJ. 1981. The ellipsoid method: A survey. Operations research. 29(6):1039–1091.

CHECKURI C, ENE A, VAKILIAN A. 2012. Prize-collecting survivable network design in node-weighted graphs. In Approximation Randomization and Combinatorial Optimization. Algorithms and Techniques: 15th International Workshop. APPROX 2012. and 16th International Workshop. RANDOM 2012. Cambridge MA. USA. August 15-17 2012. Proceedings. p. 98–109. Springer.

GUO B, QIAO C, WANG J, YU H, ZUO Y, LI J, CHEN Z, HE Y. 2013. Survivable virtual network design and embedding to survive a facility node failure. Journal of Lightwave Technology. 32(3):483–493.

IBARRA OH, KIM CE. 1975. Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM (JACM). 22(4):463–468.

JAIN K. 2001. A factor 2 approximation algorithm for the generalized steiner network problem. Combinatorica. 21(1):39–60.

LAU LC, SINGH M. 2013. Additive approximation for bounded degree survivable network design. SIAM Journal on Computing. 42(6):2217–2242.

MOMKE T, SVENSSON O. 2011. Approximating graphic tsp by matchings. In 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science. p. 560–569. IEEE.

SON S, WI G, PARK K. 2022. Situation-aware survivable network design for tactical environments. Applied Sciences. 12(13):6738.

VAZIRANI VJ. 2013. Approximation algorithms. Springer Science & Business Media.

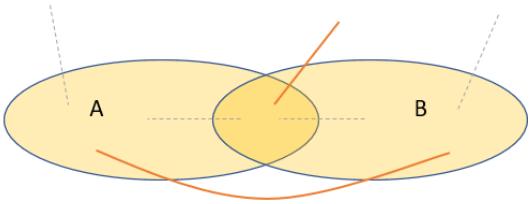
WILLIAMSON DP, GOEMANS MX, MIHAIL M, VAZIRANI VJ. 1995. A primal-dual approximation algorithm for generalized steiner network problems. Combinatorica. 15(3):435–454.

WILLIAMSON DP, SHMOYS DB. 2011. The design of approximation algorithms. Cambridge university press.

6. Appendix

6.1 Weakly Supermodular Property of f

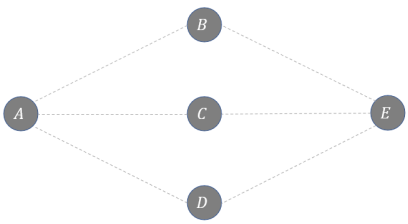
This proof for showing the weakly supermodular property of f follows Lemma 11.20 of (Williamson and Shmoys, 2011). Firstly, for the trivial case we have $f(\emptyset) = 0$, which is weakly supermodular. Otherwise, consider again the following graph:



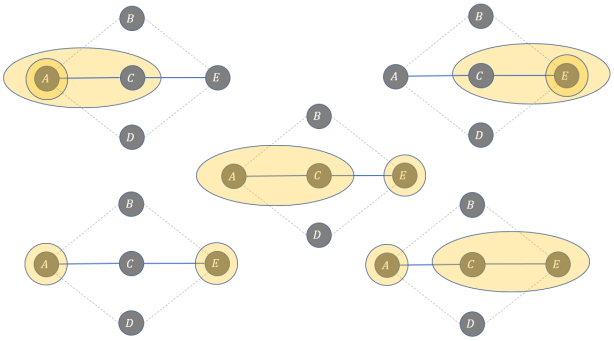
We observe four inequalities: (1) $f(A) \leq \max(f(A - B), f(A \cap B))$, (2) $f(A) \leq \max(f(B - A), f(A \cup B))$, (3) $f(B) \leq \max(f(B - A), f(A \cap B))$, and (4) $f(A) \leq \max(f(A - B), f(A \cup B))$. These inequalities follow by a simple counting argument. For instance, for the first inequality, edges with an endpoint in $A - B$ and another in $A \cap B$ do not contribute to $f(A)$, but they contribute to $\max(f(A - B), f(A \cap B))$. To arrive at the weakly supermodular property for f, we gather the minimum of $f(A - B)$, $f(B - A)$, $f(A \cup B)$ and $f(A \cap B)$. If the minimum is $f(A - B)$, we add inequalities (1) and (4) to arrive at the inequality $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$. If the minimum is $f(B - A)$, we add inequalities (2) and (3) to arrive at the inequality $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$. If the minimum is $f(A \cup B)$, we add inequalities (2) and (4) to arrive at the inequality $f(A) + f(B) \leq f(A - B) + f(B - A)$. Lastly, if the minimum is $f(A \cap B)$, we add inequalities (1) and (3) to arrive at the inequality $f(A) + f(B) \leq f(A - B) + f(B - A)$. In all these cases for the minimum, we have shown that either $f(A) + f(B) \leq f(A - B) + f(B - A)$ or $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$, thereby showing that f is weakly supermodular.

6.2 Illustrations for Theorem 1 and 2

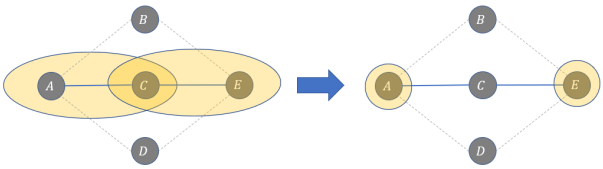
In this section of the Appendix, we provide some simple illustrations of the ideas behind Theorem 1 and Theorem 2 of (Jain, 2001). Naturally, more comprehensive explanations and proofs of these Theorems are described in the seminal paper of (Jain, 2001). These ideas would be applied to Theorems 3 and 5 which are their counterparts for the survivable network design problem with constrained $\mathfrak{h}$-subgraph flows considered in this paper. As mentioned, the proof of Theorem 1 relies on the results of Theorem 2 which states four nice properties of basic feasible solutions of the standard survivable network design problem. The first three properties listed in Theorem 2 (i.e. tightness, linear independence of $S \in \mathcal{L}$, as well as $|\mathcal{L}| = |E|$) are all used to prove the important result that $\mathcal{L}$ is laminar. To see this, consider the following graph, where all edges (dashed lines) have a cost of 1.0.



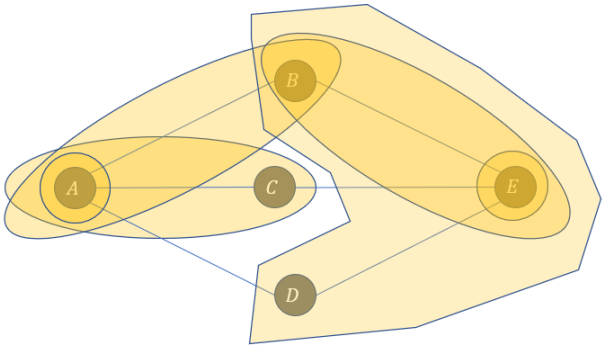
Suppose that in the above graph, vertex A is required to have one path connecting it to vertex E. One basic feasible solution is to give a weight of 1.0 to edges e_{AC} and e_{CE} resulting in a solution with total cost of 2.0. In this case, we can form several possible collections $\mathcal{L}$ of tight sets such that $|\mathcal{L}| = |E|$. These are as follows (orange circles), where the tight sets are linearly independent and laminar.



To see that they are linearly independent, we consider the top left configuration of tight sets. In this case, we have one tight set $S_1 = \{A\}$ and another as $S_2 = \{A, C\}$ such that $\delta(S_1) = \{e_{AC}\}$ and $\delta(S_2) = \{e_{CE}\}$. Since only two edges have positive weight, let the first dimension of the respective characteristic vectors denote edge e_{AC}, and let the second dimension denote e_{CE}. This gives $\chi_{S_1} = [1,0]$ and $\chi_{S_2} = [0,1]$ which are linearly independent. From (Jain, 2001), as long as the solution is a basic and feasible solution to the survivable network design problem (which could always be arrived at using standard linear programming properties), then the respective characteristic vectors are independent and any non-laminar collection of tight sets can be transformed from a non-laminar solution to a laminar solution by means of an "uncrossing" process. For instance, the following figure shows how the non-laminar collection of tight sets at the left is uncrossed to arrive at a laminar collection of tight sets to the right.

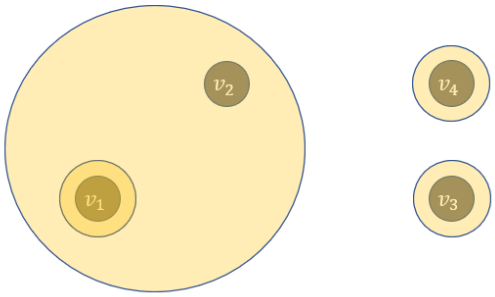


A solution to the above graph however can be a non-basic solution. For instance, a similar solution with a total cost of 2.0 can be arrived at by assigning a weight of 0.33 to all edges. This solution still meets the connectivity requirements of connecting vertex A with vertex B. However, the resulting collection of tight sets would no longer be laminar nor linearly independent. For instance, a possible collection $\mathcal{L}$ of tight sets such that $|\mathcal{L}| = |E|$ is as follows:



Here, we have $S_1 = \{A\}$, $S_2 = \{E\}$, $S_3 = \{A, C\}$, $S_4 = \{B, E\}$, $S_5 = \{B, E, D\}$, and $S_6 = \{A, B\}$, so that $\delta(S_1) = \{e_{AB}, e_{AC}, e_{AD}\}$, $\delta(S_2) = \{e_{BE}, e_{CE}, e_{DE}\}$, $\delta(S_3) = \{e_{AB}, e_{CE}, e_{AD}\}$, $\delta(S_4) = \{e_{AB}, e_{CE}, e_{DE}\}$, $\delta(S_5) = \{e_{AB}, e_{CE}, e_{AD}\}$ and $\delta(S_6) = \{e_{BE}, e_{AC}, e_{AD}\}$. The sets S_3 and S_6 cross, resulting in $\mathcal{L}$ being non-laminar. In fact, for any non-basic feasible solution to the described problem, any collection of tight sets such that $|\mathcal{L}| = |E|$ would have a pair of crossing sets. Moreover, it could be shown that the characterstic vectors of S_1 to S_6 are not linearly independent.

Arriving at a laminar collection $\mathcal{L}$ is important in order to prove the fact any basic feasible linear programming solution x to survivable network design problem has an edge whose weight is at least $1/2$. The proof for this uses a counting argument, whereby if some basic feasible solution x has a weight of less than $1/2$ for each edge in E, then the number of counted endpoints would exceed $2|E|$, providing a contradiction. As an example of this counting argument, consider a survivable network design problem whose corresponding network graph has four vertices v_1, v_2, v_3, v_4 such that each pair of vertices is required to be connected by at least one path. Suppose that a laminar solution of this problem is shown as follows, where $S_1 = \{v_1\}$, $S_2 = \{v_1, v_2\}$, $S_3 = \{v_3\}$, and $S_4 = \{v_4\}$. Given the condition that $|\mathcal{L}| = |E|$, it follows that there are 4 edges with positive weight in the solution. Let S_2, S_3, S_4 be termed as "root sets" given that they are not contained in another tight set. On the other hand, let S_1 be a "child set" given that it is contained in a "parent set" which is S_2. Now, suppose that all edges in a basic feasible solution x all have weight less than $1/2$. Given that connectivity requirements are all integral, this implies that outgoing edges from $\delta(S_1), \delta(S_2), \delta(S_3)$ and $\delta(S_4)$ have to number at least 3, in order to be greater than or equal to 1. The counting argument then goes as follows. Since there have to be at least three edges in $\delta(S_1)$, there are three edges with an endpoint being v_1. In this case, assign a value of two to S_1, and let the surplus of one be given to its parent which is S_2. Similarly, since there have to be at least three edges in $\delta(S_3)$ and $\delta(S_4)$, assign a value of three to S_3 and S_4 given that they have no parent. For S_2, it cannot be the case that all edges in $\delta(S_2)$ are the same as in $\delta(S_1)$ due to the linear independence requirement. It follows that there have to be at least one edge in $\delta(S_2)$ with an endpoint being v_2. It follows that S_2 is assigned a value of two, one for the edge with endpoint being v_2, and another for the surplus of one given by S_1. It follows that the total value for S_1 to S_4 is counted as 10, resulting in a total number of edges of 5. This contradicts the fact that only 4 edges are part of the solution.



On the other hand, if a collection of sets is not laminar, then it is possible that each edge is assigned a weight less than $1/2$ while not arriving at a contradiction under a counting argument. For instance, consider the figure shown before involving six tight sets from a non-basic feasible solution. Here, all edges have weight less than $1/2$, but the solution remains feasible. For the survivable network design with constrained ϧ-subgraph flows problem considered in this paper, a counterpart of Theorem 1 is Theorem 5 described below. In addition, the counting method used to prove Theorem 2 has its counterpart in Lemma 11 below.

6.3 Technical Results

Given a graph $G(V, E)$, let $S \subseteq V$ and $S' \subseteq V$ be two sets of vertices. For ease of notation, we denote the set of edges with one endpoint in S and another endpoint in S' as $e_{[S,S']} \subseteq E$. In case S is a singleton, i.e. $S = \{v\}$, the notation $e_{[S,S']} = e_{[v,S']}$ denotes the set of edges with one endpoint as v and another endpoint being a vertex in S'. If both S and S' are singletons, i.e. $S = \{v\}$ and $S' = \{v'\}$, then $e_{[v,v']} = e_{v,v'}$, i.e. the unique edge connecting v and v'. For all Lemmas, Theorems and Definitions in this Appendix, we assume that the input to OP 3 is a valid input consisting of an admissible graph $G(V, E)$ with a pl-admissible collection of vertex sets $\mathcal{H}$, together with an admissible flow constraint function z, and an admissible connectivity requirement function f.

Definition 25 *Given input graph $G(V, E)$ and solution x to OP 3, define the function $c: 2^E \to \mathbb{R}$ as follows, whereby for all sets of edges $e \subseteq E$, we have:*

$$c(e) = \sum_{e \in e} x_e \tag{5}$$

Definition 26 *Let $G(V, E)$ be the admissible input graph to OP 3 that results in a basic feasible solution x, such that all edges in E have positive weight, and all zero weighted edges are removed. We define the function I as:*

$$I: \mathbb{R}^{|E|} \times V \times V \to \{0,1\}^{|E|}$$

The function I works as follows. Each edge in E is assigned a coordinate in the output vector of I. An input to I consists of a tuple $(x \in \mathbb{R}, S \subseteq V, T \subseteq V)$. Given this input, for each edge $e \in E$ with positive weight x_e under x, and which has one endpoint in S and another endpoint in T, I assigns a value of 1 to the coordinate of its output vector that corresponds to e. Otherwise, I assigns 0.

6.4 Theorem 5

In this Appendix, we first present Theorem 5 below which is needed for proving Theorem 3. The proof for Theorem 5 is built from a series of Lemmas described in the next subsection.

Theorem 5 *Let x be a basic feasible solution of OP 3. Given x, there is a collection $\mathcal{L}$ of subsets of vertices with the following properties:*

1. *for all $S \in \mathcal{L}$, S is tight.*
2. *the characteristic vectors χ_S for all $S \in \mathcal{L}$ are linearly independent.*
3. *$|\mathcal{L}| = |Z| + |H| = |E|$, where Z refers to a collection of tight sets of vertices*

with respect to connectivity requirements, and H is a collection of tight sets of vertices with respect to flow constraints, and E refers to the set of edges that have nonzero weight in x, i.e. $x_e > 0$.

4. *$\mathcal{L}$ is laminar.*

6.5 Technical Lemmas

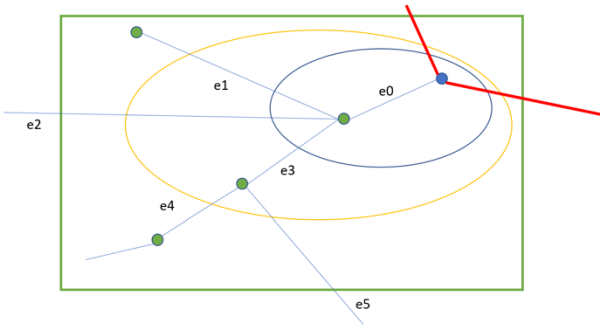


Figure 2: Figure illustrating a tight set $\Delta_{v_0^t}$ (orange circle) that contains the set $\Delta_{v_1^t}$ (blue circle), where both $\Delta_{v_0^t}$ and $\Delta_{v_1^t}$ are subsets of V_T (green box). Both $\Delta_{v_0^t}$ and $\Delta_{v_1^t}$ are connected to the root vertex (blue circle) which is in turn a trunk vertex of the trunk (trunk edges are represented by red edges)

Lemma 1 *Given a basic feasible solution x to OP 3, $\forall B \in \mathcal{H}$, $\forall v^t \in B$, given $T(v^t, V_T, E_T)$, let $\Delta_{v_0^t}, \Delta_{v_1^t} \subseteq V_T$, such that $\Delta_{v_0^t} \subseteq \Delta_{v_1^t}$ and $\Delta_{v_1^t}$ is tight. We have that $\Delta_{v_0^t}$ is also tight, and:*

$$\sum_{i \in \Delta_{v_0^t}, j \in V-B} x_{e_{i,j}} + \sum_{i \in \Delta_{v_0^t}, j \in V_T - \Delta_{v_0^t}} x_{e_{i,j}} = \sum_{i \in \Delta_{v_1^t}, j \in V-B} x_{e_{i,j}} + \sum_{i \in \Delta_{v_1^t}, j \in V_T - \Delta_{v_1^t}} x_{e_{i,j}}$$

Proof. From the given, we have that $\Delta_{v_1^t}$ is tight, or that:

$$\sum_{i \in \Delta_{v_1^t}, j \in V-B} x_{e_{i,j}} + \sum_{i \in \Delta_{v_1^t}, j \in V_T - \Delta_{v_1^t}} x_{e_{i,j}} = z(\Delta_{v_1^t})$$

From $\Delta_{v_0^t} \subseteq \Delta_{v_1^t}$, we enumerate the following sets of edges as illustrated in Fig. 2:

- e0: set of edges with one endpoint being v^t and another endpoint in $\Delta_{v_0^t}$
- e1: set of edges with one endpoint in $\Delta_{v_0^t} - \{v^t\}$ and another in $B - \Delta_{v_1^t}$
- e2: set of edges with one endpoint in $\Delta_{v_0^t}$ and another in $V - V_T$
- e3: set of edges with one endpoint in $\Delta_{v_0^t}$ and another in $\Delta_{v_1^t}$
- e4: set of edges with one endpoint in $\Delta_{v_1^t} - \Delta_{v_0^t}$ and another in $V_T - \Delta_{v_1^t}$
- e5: set of edges with one endpoint in $\Delta_{v_1^t} - \Delta_{v_0^t}$ and another in $V - V_T$

It follows that:

$$\sum_{i \in \Delta_{v_1^t}, j \in V-B} x_{e_{i,j}} + \sum_{i \in \Delta_{v_1^t}, j \in V_T - \Delta_{v_1^t}} x_{e_{i,j}} = c(e1) + c(e2) + c(e4) + c(e5)$$

However, given constraint C2, we have

$$c(e4) + c(e5) \leq c(e3)$$

Since x is feasible, we have $c(e1) + c(e2) + c(e3) \leq z(\Delta_{v_0^t})$. Moreover, from condition Z1 of z, we have $z(\Delta_{v_0^t}) = z(\Delta_{v_1^t})$. This leads to the following set of equations:

$$z(\Delta_{v_1^t}) = c(e1) + c(e2) + c(e4) + c(e5)$$
$$\leq c(e1) + c(e2) + c(e3)$$
$$= z(\Delta_{v_0^t})$$

which implies that $c(e1) + c(e2) + c(e4) + c(e5) = c(e1) + c(e2) + c(e3)$, thereby proving the second claim of the Lemma. In addition, $c(e1) + c(e2) + c(e3) = z(\Delta_{v_0^t})$, or that $\Delta_{v_0^t}$ is tight with respect to its flow constraint.

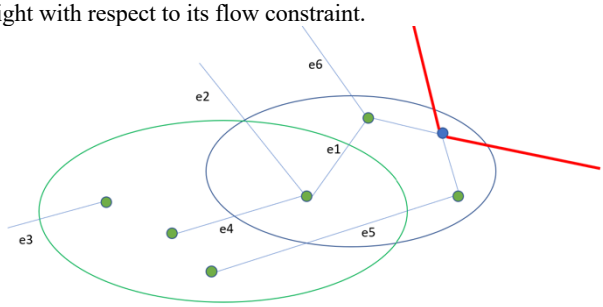


Figure 3: Figure illustrating tight sets $S \in \mathcal{L}$ (blue circle) and $T \in \mathcal{L}$ (green circle). S is tight with respect to flow constraints given by z and is connected to a trunk vertex (blue vertex). T is tight with respect to connectivity requirements given by f.

Lemma 2 *Given a basic feasible solution x to OP 3, let $S, T \in \mathcal{L}$ be two tight sets that cross, where T is tight with respect to connectivity requirements given by f, and $S \subset B$ for some $B \in \mathcal{H}$ is tight with respect to flow constraints given by z. We have $f(T) = f(T - S)$. In addition, $T - S$ is tight with respect to connectivity requirements given by f, and $S - T$ is tight with respect to flow constraints given by z.*

Proof. We first observe that under condition F1 of f, we have zero connectivity requirements such that:

$$\forall B \in \mathcal{H}, \forall i \in B, \forall j \in V - \{i\}: r_{ij} = 0$$

Thus for any $B \in \mathcal{H}$, $f(S) = 0$ for all $S \subset B$. Since $S \cap T \subset B$, we have:

$$f(T) = f(\{T \cap S\} \cup \{T - S\})$$
$$= f(T - S) \quad (6)$$

As $S \subset B$ is tight with respect to flow constraints, it follows that S is a set Δ_{v^t} that is connected to some trunk vertex $v^t \in B$. Given this, We now enumerate the following sets of edges as illustrated in Fig. 3:

- e1: set of edges with one endpoint in $(S - T)$ and another in $T \cap S$
- e2: set of edges with one endpoint in $T \cap S$ and another in $V - (T \cup S)$
- e3: set of edges with one endpoint in $T - S$ and another in $V - (T \cup S)$
- e4: set of edges with one endpoint in $T - S$ and another in $T \cap S$
- e5: set of edges with one endpoint in $T - S$ and another in $S - T$
- e6: set of edges with one endpoint in $(S - T) - \{v^t\}$ and another in $V - (T \cup S)$

It follows that $\delta(T)$ is composed of edges belonging to $e3$, $e2$, $e1$, and $e5$ or that:

$$c(e1) + c(e2) + c(e3) + c(e5) = c(\delta(T)) = f(T)$$

On the other hand, given that S is tight with respect to flow constraints given by z, we have:

$$c(e2) + c(e4) + c(e5) + c(e6) = z(S)$$

Given that x is feasible, we also have:

$$c(e3) + c(e4) + c(e5) \geq f(T - S)$$

But from constraint C2, it should hold that:

$$c(e2) + c(e4) \leq c(e1)$$

Given that the respective endpoints of edges in $e2$ and $e4$ that are in $T \cap S$ are children of parent endpoints in $S - T$ (where the parent-child connection is made by edges in $e1$), given that $S - T$ contains the root v^t. This leads to the following set of equations:

$$\begin{aligned} f(T - S) &\leq c(e3) + c(e4) + c(e5) \\ &\leq c(e3) + c(e2) + c(e4) + c(e5) \\ &\leq c(e3) + c(e2) + c(e1) + c(e5) \\ &= f(T) \end{aligned}$$

However, given that $f(T - S) = f(T)$ as pointed out above, the above inequalities should all hold with equality, therefore implying that $f(T - S) = c(e3) + c(e4) + c(e5)$, or that $T - S$ is tight with respect to connectivity requirements given by f. Lastly, to show that $S - T$ is tight with respect to flow constraints given by z, we use Lemma 1 and the fact that $S - T \subset S$, and that S is tight. This implies that $S - T$ is tight with respect to flow constraints given by z.

Lemma 3 *Given a basic feasible solution x to OP 3, suppose that $T, S \in \mathcal{L}$ are two tight sets that cross, where T is tight with respect to connectivity requirements given by f, and where $S \subset B$ for some $B \in \mathcal{H}$ is tight with respect to flow constraints given by z. We have that the sum of weights for edges with one endpoint in $T \cap S$ and another endpoint in $V - (T \cup S)$ is zero, i.e.*

$$\sum_{i \in T \cap S, j \in V - (T \cup S)} x_{e_{i,j}} = 0$$

Proof. For this proof, we use the same set of edges $e1$-$e6$ enumerated in Lemma 3. This Lemma is equivalent to saying that $c(e2) = 0$. Note that the assumptions of this Lemma are exactly the same as the assumptions of Lemma 2. Hence, we can apply Lemma 2, where we have $f(T) = f(T - S)$, and that $T - S$ is tight with respect to connectivity requirements given by f. This leads to the following set of equations:

$$\begin{aligned} f(T - S) &= c(e3) + c(e4) + c(e5) \\ &= c(e3) + c(e2) + c(e4) + c(e5) \\ &= f(T) \end{aligned}$$

The above equalities would be satisfied if and only if $c(e2) = 0$, thereby proving the statement of the Lemma.

Lemma 4 *Given a basic feasible solution x to OP 3, let $\mathcal{L}$ be a collection of tight sets with respect to either connectivity requirements or flow constraints. Suppose that there are two tight sets $S, T \in \mathcal{L}$ that cross, where T is tight with respect to connectivity requirements given by f, and S is tight with respect to flow constraints given by z. From S and T, we can form new tight sets X and Y such that X and Y are laminar and $\chi_T + \chi_S = \chi_X + \chi_Y$.*

Proof. Firstly, from Lemma 2, $T - S$ is tight with respect to connectivity constraints given by f, and $S - T$ is tight with respect to flow constraints given by z. We can thus set $X := T - S$ and $Y := S - T$ where both X and Y are tight with respect to their own respective constraints. To show that $\chi_T + \chi_S = \chi_X + \chi_Y$, we use the same set of edges $e1$-$e6$ enumerated in Lemma 3. The edges in $\delta(T)$, $\tau(S)$, $\tau(S - T)$ and $\delta(T - S)$ are:

$$\begin{aligned} \delta(T) &= \cup \{e1, e2, e3, e5\} \\ \tau(S) &= \cup \{e2, e4, e5, e6\} \\ \delta(T - S) &= \cup \{e3, e4, e5\} \\ \tau(S - T) &= \cup \{e1, e5, e6\} \end{aligned}$$

It follows that the only edges in $\delta(T) \cup \tau(S)$ that are not found in $\delta(T - S) \cup \tau(S - T)$ are edges in $e2$, or that:

$$(\chi_T + \chi_S) - (\chi_{T-S} + \chi_{S-T}) = 2 \times I[x, T \cap S, V - (T \cup S)]$$

However, as stated in Lemma 3, we have $c(e2) = 0$. Therefore $2 \times I[x, T \cap S, V - (T \cup S)] = 0$, or that $\chi_T + \chi_S = \chi_{T-S} + \chi_{S-T} = \chi_X + \chi_Y$ as claimed.

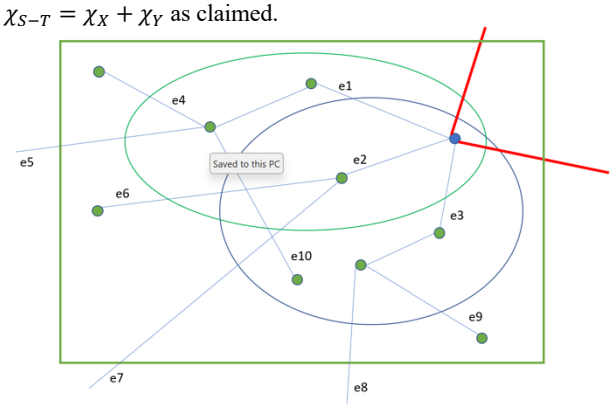


Figure 4: Figure illustrating two crossing sets $S, T \in \mathcal{L}$ such that both S (blue circle) and T (green circle) are tight with respect to flow constraints given by z. Both S and T are connected to the same root represented by a trunk vertex (blue circle) and are subsets of V_T.

Lemma 5 *Given a basic feasible solution x to OP 3, let $\mathcal{L}$ be a collection of tight sets with respect to either connectivity requirements or flow constraints. Suppose that there are two tight sets $S, T \in \mathcal{L}$ that cross, where both S and T are tight with respect to flow constraints given by z. From S and T, we can form new tight sets X and Y such that X and Y are laminar and $\chi_T + \chi_S = \chi_X + \chi_Y$.*

Proof. We consider the following cases for which S and T may cross.

Case 1: $S \subset V_A$ and $T \subset V_B$, where V_A and V_B are sets of vertices of distinct branches $T(v_A^t, V_A, E_A)$ and $T(v_B^t, V_B, E_B)$ respectively, where $T(v^t, V_A, E_A) \subset G_B(B, E')$ and $T(v^t, V_A, E_A) \subset G_{B'}(B', E'')$, for some $B, B' \in \mathcal{H}$ with $B \neq B'$

Case 2: $S \subset V_A$ and $T \subset V_B$, where V_A and V_B are sets of vertices of distinct branches $T(v_A^t, V_A, E_A)$ and $T(v_B^t, V_B, E_B)$ respectively, where $T(v^t, V_A, E_A) \subset G_B(B, E')$ and $T(v^t, V_A, E_A) \subset G_B(B, E')$, for some $B \in \mathcal{H}$

Case 3: $S, T \subset V_T$ and V_T is the set of vertices of a branch $T(v^t, V_T, E_T)$, whose root is a trunk vertex $v^t \in B$ of some $B \in \mathcal{H}$

We first note that cases 1 and 2 do not occur given conditions G1 and G2 for $\mathcal{H}$ to be admissible, whereby under G1, for any pair $B, B' \in \mathcal{H}$, with $B \neq B'$, we have that B and B' are disjoint. In addition, under G2, for each $B \in \mathcal{H}$, any pair of distinct branches are disjoint. It follows that V_A and V_B in both cases are disjoint. Given that $S \subset V_A$ and $T \subset V_B$, and that z can only be tight for subsets of V_A and V_B that are connected to their respective root vertices, it follows that $S \cap T = \emptyset$.

For case 2, we enumerate the possible edges in $\tau(T)$, $\tau(S)$, $\tau(S \cap T)$, and $\tau(S \cup T)$ as follows. These edges are illustrated in Fig. 4.

e1: set of edges from v^t to $T - S$
e2: set of edges from v^t to $T \cap S$
e3: set of edges from v^t to $S - T$
e4: set of edges from $T - S$ to $V_T - (T \cup S)$
e5: set of edges from $T - S$ to $V - V_T$
e6: set of edges from $T \cap S$ to $V_T - (T \cup S)$
e7: set of edges from $T \cap S$ to $V - V_T$
e8: set of edges from $S - T$ to $V - V_T$
e9: set of edges from $S - T$ to $V - (T \cup S)$
e10: set of edges from $T - S$ to $S - T$

From the above enumerations of edges, we thus have:

$$\tau(T) = \bigcup \{e3, e4, e5, e6, e7, e10\}$$
$$\tau(S) = \bigcup \{e1, e6, e7, e8, e9, e10\}$$
$$\tau(T \cap S) = \bigcup \{e1, e3, e6, e7\}$$
$$\tau(T \cup S) = \bigcup \{e4, e5, e6, e7, e8, e9\}$$

We claim that there are no edges in $e10$. To see this, note that S and T correspond to subsets of vertices of V_T that are connected to the root v^t given condition Z2 of z, whereby a subset of vertices of V_T becomes tight with respect to flow constraints given by z if and only if the subset is a set connected to v^t. It follows that for any vertex $i \in S$, there is a path from i towards the root v^t. Similarly, for any vertex $j \in S$, there is a path from j towards the root v^t. Suppose for that there is an edge in $e10$ with positive weight under solution x that connects $i \in T - S \subset T$ to $j \in S - T \subset S$. Given that i is an endpoint of this positively weighted edge, from constraint C2 of OP3, all paths from i to the root v^t should also have positive weight. Similarly, given that j is an endpoint of some positively weighted edge in $e10$, all paths from j to the root v^t should also have positive weight. It follows that we could then construct a positively weighted path from the root v^t towards i, followed by an edge from i to j (using the edge in $e10$) then from j to the root v^t. This creates a cycle which contradicts the tree structure of $T(v^t, V_T, E_T)$. Therefore there should be no edge in $e10$.

From Lemma 1, we have that $T \cap S$ is tight given that $T \cap S \subseteq T$ and $T \cap S \subseteq S$, and both S and T are tight. To see that $T \cup S$ is also tight, we note the following set of equations given that x is feasible and T,S, $S \cap T$ are all tight, and that $c(e10) = 0$ as shown in the preceding paragraph.

$$z(T) = c(\tau(T)) = c(e3) + c(e4) + c(e5) + c(e6) + c(e7)$$
$$z(S) = c(\tau(S)) = c(e1) + c(e6) + c(e7) + c(e8) + c(e9)$$
$$z(T \cap S) = c(\tau(T \cap S)) = c(e1) + c(e3) + c(e6) + c(e7)$$
$$z(T \cup S) \geq c(\tau(T \cup S)) = c(e4) + c(e5) + c(e6) + c(e7) + c(e8) + c(e9)$$

From condition Z1 of z, we have $z(T) = z(S) = z(T \cap S)$, which implies that:

$$c(e1) + c(e3) + c(e6) + c(e7) = c(e3) + c(e4) + c(e5) + c(e6) + c(e7) = c(e1) + c(e6) + c(e7) + c(e8) + c(e9)$$

or equivalently:

$$c(e1) = c(e4) + c(e5)$$
$$c(e3) = c(e8) + c(e9)$$

Combining all of the above leads us to:

$$z(T \cup S) \geq c(e4) + c(e5) + c(e8) + c(e9) + c(e6) + c(e7) = c(e1) + c(e8) + c(e9) + c(e6) + c(e7) = z(S) = z(T)$$

But given that $z(T) = z(S) = z(T \cap S) = Z(T \cup S)$ from condition Z1 of z, the above equations should all hold with equality, which thus implies that $T \cup S$ is tight with respect to flow constraints given by z. We can thus set $X := T \cap S$ and $Y := T \cup S$, where both X and Y are tight. To show that $\chi_T + \chi_S = \chi_X + \chi_Y$, from the above enumeration of edges, we can see that edges which do not belong to $\tau(T \cap S) \cup \tau(T \cup S)$ are those edges in $e10$, or that:

$$(\chi_T + \chi_S) - (\chi_{T \cap S} + \chi_{T \cup S}) = 2 \times I[x, T - S, S - T]$$

However, as pointed out, there are no edges in $e10$ due to the tree structure of $T(v^t, V_T, E_T)$. Thus we have $\chi_T + \chi_S = \chi_X + \chi_Y$ as claimed.

Lemma 6 *Given a basic feasible solution x to OP 3, let $\mathcal{L}$ be a collection of tight sets with respect to either connectivity requirements or flow constraints. Suppose that there are two tight sets $S, T \in \mathcal{L}$ that cross. From S and T, we can form new tight sets X and Y such that X and Y are laminar and $\chi_T + \chi_S = \chi_X + \chi_Y$.*

Proof. We consider the following cases to prove the Lemma:

Case 1: S and T are both tight with respect to connectivity requirements given by f.
Case 2: S and T are both tight with respect to flow constraints given by z.
Case 3: T is tight with respect to connectivity requirements given by f and S is tight with respect to flow constraints given by z.

Cases 2 and 3 are handled by Lemmas 4 and 5 respectively. This leaves us with case 1. However, for case 1, this is the same case of crossing sets considered in (Jain, 2001), whereby under a weakly supermodular function f, any pair of crossing tight sets S and T results in either (1) $f(S) + f(T) \leq f(T - S) + f(S - T)$ or (2) $f(S) + f(T) \leq f(T \cup S) + f(T \cap S)$ occurs. If (1) occurs, we set $X := T - S$ and $Y := S - T$. If (2) occurs, we set $X := T \cup S$ and $Y := T \cap S$. From (Jain, 2001), in either case, both X and Y are tight with respect to connectivity requirements, and $\chi_T + \chi_S = \chi_X + \chi_Y$.

Lemma 7 *Given a basic feasible solution x to OP 3, let $\mathcal{L}$ be a collection of tight sets with respect to either connectivity requirements or flow constraints. Suppose that $S \subset V$ is a tight set (with respect to either connectivity requirements or flow constraints) such that $\chi_S \notin span(\mathcal{L})$ and S crosses a set $T \in \mathcal{L}$. From Lemma 6, there is a tight set S' that does not cross T and such that $\chi_{S'} \notin span(\mathcal{L})$*

Lemma 8 *Given a basic feasible solution x to OP 3, let $\mathcal{L}$ be a collection of tight sets with respect to either connectivity requirements or flow constraints. It follows from Lemma 7 that if $span(\mathcal{L}') \neq \mathbb{R}^{|E|}$, there is a tight set S such that $\chi_S \notin span(\mathcal{L}')$ and $\mathcal{L}' \cup \{S\}$ is a laminar family.*

Thus, given Lemmas 7 and 8 above, a procedure is outlined which shows how to construct the collection $\mathcal{L}$ with properties described in Theorem 5. This construction procedure for building a collection of laminar, linearly independent tight sets thus proves Theorem 5.

6.3 Proof of Theorem 3

The following Definitions are needed for Lemma 9 which forms the core idea behind the proof of Theorem 3. Lemma 9 builds on the results of Theorem 5 which was proved in the previous subsection.

Definition 27 *A forest $\mathcal{F}$ is a partial ordering of sets $S \in \mathcal{L}$. Each tree in $\mathcal{F}$ has a root that consists of a set not contained in any other set in $\mathcal{L}$. Given $S \in \mathcal{L}$ such that $S \subset S'$ for some $S' \in \mathcal{F}$, the set S becomes a child of S' in $\mathcal{F}$, and S' becomes the*

parent of S if S' is the minimal set that contains S. A node $S \in \mathcal{F}$ owns vertex v if S is the lowest (or minimal) set that contains v with respect to the partial ordering defined by $\mathcal{F}$.

Definition 28 *Given input graph $G(V,E)$ and basic feasible solution x to OP 3, let $\mathcal{L}$ be the laminar collection of tight sets with properties described in Theorem 5. Suppose that $S \in \mathcal{L}$ is a tight set with respect to either connectivity requirements or flow constraints. From (Vazirani, 2013), a corequirement function $\mathfrak{c}_f: 2^V \to \mathbb{Q}$ under connectivity requirements given by f is as follows:*

$$\mathfrak{c}_f(S) = \frac{1}{2}|\delta(S)| - f(S) = \sum_{e \in \delta(S)} \frac{1}{2} - x_e$$

In our setting, we use define another corequirement function $\mathfrak{c}_z: 2^V \to \mathbb{Q}$ under flow constraints given by z as follows:

$$\mathfrak{c}_z(S) = \frac{1}{2}|\tau(S)| - z(S) = \sum_{e \in \tau(S)} \frac{1}{2} - x_e$$

It follows that if for some tight set $S \in \mathcal{L}$, we have $\mathfrak{c}_f(S) = 1/2$, or $\mathfrak{c}_z(S) = 1/2$, then $|\delta(S)|$ or $|\tau(S)|$ is odd respectively.

Lemma 9 *Given input graph $G(V,E)$ and basic feasible solution x to OP 3, let $\mathcal{L}$ be the laminar collection of tight sets with properties described in Theorem 5. Suppose that $S \in \mathcal{L}$ is a tight set with respect to connectivity requirements given by f, and which has α children and owns β endpoints, where $\alpha + \beta = 3$. Moreover, let each child S' of S have a corequirement of either $\mathfrak{c}_z(S') = 1/2$ if the child S' is tight with respect to flow constraints given by z, or $\mathfrak{c}_f(S') = 1/2$ if the child S' is tight with respect to connectivity requirements given by f. We have $\mathfrak{c}_f(S) = 1/2$*

Proof. For this Lemma, we follow the proof from (Vazirani, 2013). From the assumptions, each child S' of S has either $\mathfrak{c}_f(S') = 1/2$ or $\mathfrak{c}_z(S') = 1/2$. This implies that each child of S an odd number of outgoing edges. We now consider the following cases:

- S owns no endpoints. In this case, we have $\beta = 3$, and all edges with an endpoint in a child of S has to have an endpoint outside of S (since S owns no endpoint). Therefore, we have $|\delta(S)|$ as the sum of the number of outgoing edges of S's three children. Since each child of S has an odd number of outgoing edges, it follows that $|\delta(S)|$ is odd.
- S owns one endpoint. In this case, we have $\beta = 2$. Let S' and S'' denote the two children of S. Since each child has an odd number of outgoing edges, we have $|\delta(S')| + |\delta(S'')|$ as even (and similarly for either $|\delta(S')| + |\tau(S'')|$, $|\tau(S')| + |\delta(S'')|$ or $|\tau(S')| + |\tau(S'')|$). Adding this number to the edge whose endpoint is owned by S, we have that $|\delta(S)|$ is odd.
- S owns two endpoints. In this case, S has only one child with $|\delta(S)|$ as odd. Adding this number to the two edges whose endpoints are owned by S, we have that $|\delta(S)|$ is odd.
- S has three endpoints. In this case, S has no children, and we trivially have $|\delta(S)| = 3$ which is odd.

It follows that in all cases considered above, we have $|\delta(S)|$ as odd, which implies that $\mathfrak{c}_f(S) = n/2$ for some odd $n \geq 1$. Following (Vazirani, 2013), we have:

$$\mathfrak{c}_f(S) \leq \sum_{S'} \mathfrak{c}_f(S') + \sum_{S''} \mathfrak{c}_z(S'') + \sum_e \frac{1}{2} - x_e$$

where S' ranges over the children of S that are tight with respect to connectivity requirements given by f, and S'' ranges over the children of S that are tight with respect to flow constraints given by z. The first inequality follows from the fact that there may be some edges with an endpoint in a child of S, and another endpoint in S (outside of the child). Given that $\frac{1}{2} - x_e$ is less than $1/2$, we consider the following cases:

1. If S has one child S' and owns two endpoints such that $\mathfrak{c}_f(S') = 1/2$ or $\mathfrak{c}_z(S') = 1/2$, we have $\mathfrak{c}_f(S) \leq 1/2 + [2 \times (\frac{1}{2} - x_e)] < 3/2$.
2. If S has two children and owns one endpoint we have $\mathfrak{c}_f(S) \leq 1 + [1 \times (\frac{1}{2} - x_e)] < 3/2$.
3. If S has three children and owns no endpoints, it should not be the case all edges with an endpoint in a child of S have their opposite endpoints outside of S since this violates linear independence of $\mathcal{L}$. Therefore we have $\mathfrak{c}_f(S) < 3/2$

Given that $|\delta(S)|$ is odd, and $\mathfrak{c}_f(S) < 3/2$ from the above cases, we have $\mathfrak{c}_f(S) = 1/2$ as stated in the Lemma.

Lemma 10 *Given input graph $G(V,E)$ and basic feasible solution x to OP 3, let $\mathcal{L}$ be the laminar collection of tight sets with properties described in Theorem 5. Suppose that $S \in \mathcal{L}$ is a tight set with respect to connectivity requirements given by f, and which has two children $S', S'' \in \mathcal{L}$, and that for S', we either have $\mathfrak{c}_f(S') = 1/2$ or $\mathfrak{c}_z(S') = 1/2$. It follows that S must own at least one endpoint.*

Proof. We follow the proof shown in (Vazirani, 2013) which uses contradiction. Namely, suppose that S does not own any endpoint. Given linear independence of characteristic vectors of sets in $\mathcal{L}$, we have that edges in $\delta(S')$ (or $\tau(S')$)should not be the same edges in $\delta(S)$ or $\delta(S'')$ (or $\tau(S'')$). Now, if $\mathfrak{c}_z(S') = 1/2$, given that S owns no endpoints, we have:

$$\sum_{i \in S', j \in V-S} [1/2 - x_{e_{i,j}}] + \sum_{i \in S'', j \in S'} [1/2 - x_{e_{i,j}}] = \mathfrak{c}_z(S') = \frac{1}{2} \quad (7)$$

Similarly, if instead we have $\mathfrak{c}_f(S') = 1/2$, given that S owns no endpoints, we have:

$$\sum_{i \in S', j \in V-S} [1/2 - x_{e_{i,j}}] + \sum_{i \in S'', j \in S'} [1/2 - x_{e_{i,j}}] = \mathfrak{c}_f(S') = \frac{1}{2} \quad (8)$$

Given that for S', we either have $\mathfrak{c}_f(S') = 1/2$ or $\mathfrak{c}_z(S') = 1/2$, then either $|\delta(S')|$ or $|\tau(S')|$ is odd. Given Eq. 7 or 8, this implies that the parity of the number of edges with an endpoint in S' and another endpoint in $V - S$ is different from the parity of the number of edges with an endpoint in S' and another endpoint in S''. This difference in parity implies that the corequirement $\mathfrak{c}_f(S)$ is different from $\mathfrak{c}_z(S'')$ if S'' is tight with respect to flow constraints. Similarly, if S'' is tight with respect to connectivity requirements, we have $\mathfrak{c}_f(S)$ as different from $\mathfrak{c}_f(S'')$. Suppose that S'' be tight with respect to flow constraints. We have:

$$\mathfrak{c}_f(S) = \mathfrak{c}_z(S'') + \sum_{i \in S', j \in V-S} [1/2 - x_{e_{i,j}}] - \sum_{i \in S'', j \in S'} [1/2 - x_{e_{i,j}}] \quad (9)$$

which implies that:

$$\mathfrak{c}_f(S) - \mathfrak{c}_z(S'') = \sum_{i \in S', j \in V-S} [1/2 - x_{e_{i,j}}] - \sum_{i \in S'', j \in S'} [1/2 - x_{e_{i,j}}]$$

Similarly, if S'' is tight with respect to connectivity requirements, we have:

$$\mathfrak{c}_f(S) = \mathfrak{c}_f(S'') + \sum_{i\in S',j\in V-S} [1/2 - x_{e_{i,j}}] - \sum_{i\in S'',j\in S'} [1/2 - x_{e_{i,j}}] \quad (10)$$

which implies that:

$$\mathfrak{c}_f(S) - \mathfrak{c}_f(S'') = \sum_{i\in S',j\in V-S} [1/2 - x_{e_{i,j}}] - \sum_{i\in S'',j\in S'} [1/2 - x_{e_{i,j}}]$$

Any of the above equations imply that:

$$\frac{1}{2} < \sum_{i\in S',j\in V-S} [1/2 - x_{e_{i,j}}] - \sum_{i\in S'',j\in S'} [1/2 - x_{e_{i,j}}] < \frac{1}{2}$$

This means that S and S'' have the same corequirement whether $S'"$ is tight with respect to connectivity requirements given by f, or is tight with respect to connectivity requirements given by z, which is a contradiction. This means that S has to own at least one endpoint.

Lemma 11 *Given a basic feasible solution x to OP 3, let $\mathcal{L}$ be the laminar collection of tight sets with properties described in Theorem 5, and let $G(V, E)$ be the resulting graph whose edges all have positive weight under x. Suppose that $x_e < 1/2$ for all $e \in E$ and that for all $B \in \mathcal{H}$, we have that all subsets $\Delta_{v^t} \subseteq V_T$ (where V_T is the vertex set of any branch $T(v^t, V_T, E_T)$ connected to some trunk vertex $v^t \in B$), has more than 3 outgoing edges, i.e. $\tau(\Delta_{v^t}) \geq 4$. Given the forest $\mathcal{F}$ constructed from the partial ordering of $\mathcal{L}$, the counted number of endpoints owned by nodes in $\mathcal{F}$ exceed $2|E|$.*

Proof. We borrow notations from (Jain, 2001), which uses the concept of *tokens*. Namely, each endpoint of any edge $e \in E$ corresponds to one token. Given an endpoint of edge e, its token is initially assigned to the set that owns the vertex v corresponding to an endpoint of e. Afterwards, the tokens can be re-assigned to other sets. From the assumptions, each set $S \in \mathcal{L}$ that is not a root in $\mathcal{F}$ can be assigned two tokens. However, roots of trees in $\mathcal{F}$ are assigned strictly more than two tokens. Given that the number of nodes in $\mathcal{F}$ is equal to $|E|$, there should be only $2|E|$ tokens corresponding to the total number of endpoints. However, as mentioned, more than $2|E|$ tokens are counted in total.

We show this contradiction using several cases. Similar to the notations in Theorem 5, let Z refer to a collection of sets in $\mathcal{L}$ that are tight with respect to connectivity requirements, and let H refer to a collection of sets in $\mathcal{L}$ that are tight with respect to flow constraints, i.e. $H \subseteq \mathcal{H}$. Let S be some node in $\mathcal{F}$. We consider the following cases.

Case 1: $S \in Z$ is a leaf.
Case 2: $S \in H$ is a leaf.
Case 3: $S \in H$ has some children.
Case 4: $S \in Z$ has four or more children.
Case 5: $S \in Z$ has three children.
Case 6: $S \in Z$ has two children.
Case 7: $S \in Z$ has one child.

For case 1, given that all edges $e \in E$ have $x_e < 1/2$, there has to be at least 3 edges in $\delta(S)$ from the fact that connectivity requirement function f is integral. Since there are at least three edges with endpoints in S, and S is a leaf, it follows that S can be assigned exactly 2 tokens with a surplus of at least 1. In case S has exactly three outgoing edges, this implies hat $\mathfrak{c}_f(S) = 1/2$. This fact is useful, since it implies that Lemma 9 can be applied by the parent of S.

For case 2, if $S \in H$, we have that S corresponds to some $\Delta_{v^t} \subset V_T$, where V_T is the vertex set of a branch $T(v^t, V_T, E_T)$ whose root is a trunk vertex $v^t \in B$ for some $B \in \mathcal{H}$. From the assumption, S has at least 4 outgoing edges. Therefore, S can keep 2 tokens to itself and give at least a surplus of 2 tokens to its parent if it has one.

For case 3, given that $S \in H$, we have that all of S's children are not in Z - as a result of condition F1 for the connectivity requirement function f, i.e. if $S \in H$, then all vertices in S have zero connectivity requirements. This implies that no subset of vertices of S can be tight with respect to connectivity requirements given by f. Thus, if $S \in H$, all children of S are tight with respect to flow constraints, i.e. all of its children should also belong to H. In addition, in order for S to have a child in H, it has to be the case that both S and its children belong to the vertex set V_T of some branch $T(v^t, V_T, E_T)$ whose root is a trunk vertex $v^t \in B$ of some $B \in \mathcal{H}$. The children of S cannot be connected to a different root given that the set of branches are disjoint for any $B \in \mathcal{H}$. It follows that the children of S are not disjoint.

Moreover, we claim that S should only have one child. Suppose to the contrary that S more than one child. Without loss of generality, let S have two children S' and S''. It cannot be the case that $S' \subseteq S''$ or that $S'' \subseteq S'$ given that S is the parent of both S' and S'', which implies that it is the minimal set which contains both S' and S''. We thus have $S'' \not\supseteq S'$ and $S' \not\supseteq S''$. Given that $\mathcal{L}$ is laminar, this implies that S' and S'' are disjoint. However, this is impossible given that the children of S are not disjoint given that they are connected to the same root vertex v^t as pointed out in the preceding paragraph. Thus, S has only one child.

Given that S has one child S', it cannot be the case that all outgoing edges of S have endpoints in S' since this violates the linear independence of characteristic vectors of sets in $\mathcal{L}$. Moreover, S has to own at least two endpoints given that $x_e < 1/2$ for all $e \in E$. If S has only one endpoint, then the difference in $c(\tau(S))$ and $c(\tau(S'))$ differs by a fraction, contradicting the fact that both S and S' are tight with respect to flow constraints given z, and where z is an integral function. It follows that S has 4 tokens, two of which come from the surplus of its child $S' \in H$, and another two from the endpoints it owns. Therefore, S can keep 2 tokens to itself and give at least a surplus of 2 tokens to its parent if it has one.

For case 4, if $S \in Z$ has four or more children, then S has at least 4 tokens from the surplus of its children. It follows that S can keep 2 tokens and give 2 tokens to its parent if it has one.

For case 5, $S \in Z$ has three children. If one child of S has a surplus of two (i.e. for instance, if the child belongs to H), or if S owns an endpoint, then S has four tokens. Otherwise, each child of S has a corequirement of $1/2$, and it follows that $\mathfrak{c}_f(S) = 1/2$ as well from Lemma 9. As stated, having $\mathfrak{c}_f(S) = 1/2$ is useful since it means that Lemma 9 can be applied by S's parent as its child (S) has a corequirement of $1/2$. Moreover, S has 3 tokens, and it can keep 2 tokens to itself and give 1 token to its parent if ever it has one.

For case 6, $S \in Z$ has two children. If both children have a surplus of at least 2 tokens (for instance, if both children belong to H), then S has at least 4 tokens. Otherwise, if one child has a surplus of 1, then from Lemma 10, S has to own at least one endpoint. If both children have a surplus of 1, then both children have a corequirement of $1/2$. From Lemma 9, this implies that $\mathfrak{c}_f(S) = 1/2$, and Lemma 9 can be applied again by S's parent. In addition, if S owns only one endpoint, it has 3 tokens, 2 from the surplus of its children and 1 from the endpoint it owns. Thus

S can keep 2 tokens to itself and give 1 token to its parent if ever it has one.

For case 7, if $S \in Z$ has only one child S', then S has to own at least 2 endpoints. Similar to case 3, it cannot be the case that all outgoing edges of S have endpoints in S' since this violates the linear independence of characteristic vectors of sets in $\mathcal{L}$. Moreover, if S has only one endpoint, then the difference in $c(\delta(S))$ and $c(\delta(S'))$ differs by a fraction, contradicting the fact that both S and S' are tight with respect to integral functions.

We now note that there are $|E|$ nodes in $\mathcal{F}$ given that $|\mathcal{L}| = |E|$ as per Theorem 5. By definition, if a node is a leaf, then it has to have a parent. Let the leaf node be allocated 2 tokens and let it give all of its surplus to its parent. From the enumerated cases, there always exists a surplus that can be given by a child to its parent. If the parent of the leaf node is in turn a child of another parent node, let it keep 2 tokens to itself and let it give its surplus to its respective parent. Continuing in this way for any subtree $T \in \mathcal{F}$, we eventually reach the root node of T, which would collect strictly more than 2 tokens. In case a node is a root with no children, then cases 1-2 imply that it has strictly more than 2 tokens by default. Given that each root node of $\mathcal{F}$ has strictly more than 2 tokens, whereas each non-root node is allocated 2 tokens, it follows that the total number of tokens collected in $\mathcal{F}$ is greater than $2|E|$. However, given that the number of nodes of $\mathcal{F}$ equals $|E|$, there should only be $2|E|$ endpoints. This gives rise to a contradiction, thereby proving the Lemma.