The Open University of Israel

Department of Mathematics and Computer Science

The Iterative Rounding Method for Optimization Problems

Final Paper submitted as partial fulfillment of the requirements

towards an M.Sc. degree in Computer Science

The Open University of Israel

The Department of Mathematics and Computer Science

By

Ofer Monin

Prepared under the supervision of Prof. Zeev Nutov

January 2015

Abstract

This work provides an introduction to a very simple yet powerful technique that is very useful in a variety of algorithmic problems – the *iterative rounding method*. The generality of the iterative rounding method allows us to apply it on many known optimization problems.

The main goal of this work is to illustrate the power and the potential of iterative rounding algorithms. We explain the basics that are needed to understand the iterative algorithms, and illustrate the method by several examples.

Contents

List of Figures

1 Introduction

The objective of this work is to explore the iterative rounding method. In order to understand this method we analyze a number of specific iterative algorithms. In this work we will also try to summarize some of the basic ideas. It does not include all the material relevant to iterative algorithms. Some of the theorems and lemmas in this work are not proved. In such cases, the reader is referred to other books or papers for further study. A large part of our work follows the book *Iterative Methods in Combinatorial Optimization* (by L. Chi Lau, R. Ravi and M. Singh) [1] very closely. Many of the theorems and Lemmas (as well as the ideas behind the proofs) are taken from the book. However, we also survey some recent developements, and in particular summarize the recent papers of J. Cheriyan and L. Vegh [27] and A. Ene and A. Vakilian [2].

Understanding the iterative method should be easy via an example. Let us examine the classic *Minimum Weight Bipartite Perfect Matching* problem:

Problem 1.1. (Minimum Weight Bipartite Perfect Matching) *Given a bipartite graph $G = (V_1 \cup V_2, E)$ with $|V_1| = |V_2|$ and a weight function $w\colon E \to \mathbb{R}_+$, match every vertex from V_1 with a distinct vertex from V_2 while minimizing the total weight of the matching.*

This is one of the oldest problems in combinatorial optimization (see [23]). We will examine the problem by analyzing the following example (as can be seen in Figure 1):

Example 1.1. *$V_1 = \{a, b\}$ and $V_2 = \{c, d\}$. $E = \{(a, c), (a, d), (b, c), (b, d)\}$. The weight function is defined as follows: $w(a, c) = 1$; $w(a, d) = 4$; $w(b, c) = 4$; $w(b, d) = 2$.*

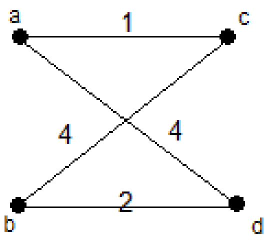

Figure 1: Min Matching in Bipartite Example

It is obvious that the minimum weight matching includes the marked edges as can be seen in Figure 2.

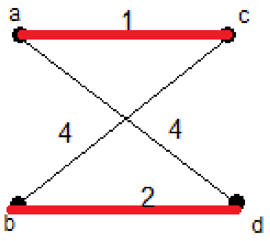

Figure 2: Min Matching in Bipartite Example - Solution

In order to use an iterative algorithm, we first formulate it as a linear programming problem. A linear program is a mathematical way to model a problem. It uses a system of linear constraints and an objective function that should be maximized (or minimized, as in our case). In our example we shall use x_{uv} as an indicator to mark if the pair (u, v) is matched (i.e. $\{u, v\} \in E$); w will denote the weight function (i.e. w_{uv} means the weight of the edge (u, v)). The following are the linear programming constraints:

$$\begin{array}{lll}
\textit{minimize} & \displaystyle\sum_{u,v} w_{uv}\, x_{uv} & \\
\textit{subject to} & \displaystyle\sum_{v:\{u,v\}\in E} x_{uv} = 1 & \forall u \in V_1 \\
 & \displaystyle\sum_{u:\{u,v\}\in E} x_{uv} = 1 & \forall v \in V_2 \\
 & x_{uv} \in \{0, 1\} & \forall \{u, v\} \in E
\end{array}$$

Solving (finding the optimal solution to) the linear program for x_{uv} that may only receive 0 or 1 is hard. It is possible to solve problem if we relax each constraint $x_{uv} \in \{0, 1\}$, and use the following requirement instead:

$$x_{uv} \geq 0 \qquad\qquad \forall \{u, v\} \in E$$

It is known that the obtained relaxation can be solved in polynomial time ([6]). In our example (Example 1.1) the obtained linear program is as follows:

$$\begin{array}{lll}
\textit{minimize} & \sum_{e \in E} w_e x_e & \\
\textit{subject to} & x_{ac} + x_{ad} = 1 & \\
 & x_{bc} + x_{bd} = 1 & \\
 & x_{ac} + x_{bc} = 1 & \\
 & x_{ad} + x_{bd} = 1 & \\
 & x_e \geq 0 & \forall e \in E
\end{array}$$

It is common to write the set of conditions in a linear program as a matrix multiplication. The w and x are vectors corresponding to the edges (w_e and x_e). The A matrix holds the coefficients in the condition. The b vector holds the results for the conditions. In our example we get the following:

$$\begin{array}{lll}
\textit{minimize} & w^T \cdot x & \\
\textit{subject to} & Ax = b & \\
 & x_e \geq 0 & \forall e \in E
\end{array}$$

Where:

$$x = \begin{pmatrix} x_{ac} \\ x_{ad} \\ x_{bc} \\ x_{bd} \end{pmatrix} \quad w = \begin{pmatrix} w_{ac} \\ w_{ad} \\ w_{bc} \\ w_{bd} \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

And:

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

An optimal solution to the linear program may not give us the desired result since we need a "binary" solution, namely, a solution that assigns to each edge a value in $\{0, 1\}$. Finding a good "binary" solution is where the iterative algorithm comes into place. The iterative algorithm is comprised of 3 steps that have to be repeated until the desired solution is received (the formal algorithm is presented in a later section, see Algorithm 1):

1. Find an optimal solution to the current LP.

2. If any x_{uv} is set to 1 in the solution, then we take the edge (u, v) into our solution and delete the pair $\{u, v\}$ from the instance to get a smaller problem.

3. If any variable x_{uv} is set to 0 in the solution, we remove the edge (u, v) from the instance to get a smaller problem.

Running the iterative algorithm until each edge gets value either 0 or 1 produces the optimal solution. This claim is very simple to prove inductively. The fact that the algorithm actually works correctly (i.e. during each iteration, there is at least one edge with 1 or 0) is not trivial and should be proven. We will return to this proof later. We need some definitions and preliminary claims.

The definition of the linear program and how it is handled is presented in the next section. In this section, it is assumed that the reader understands the linear programming method.

Definition 1.1. *Let $P = \{x : Ax = b, x \geq 0\} \subseteq \mathbb{R}^n$ be a set of feasible solutions to a linear program. Then $x \in \mathbb{R}^n$ is an* **extreme point solution** *of P if there does not exist a non-zero vector $y \in \mathbb{R}^n$ such that $x+y, x-y \in P$.*

An extreme point solution is one of the corner points of the set of feasible solution. For example, if our set of feasible solutions is defined by the shape in Figure 3, the extreme points solutions are the solutions on the corners marked in red.

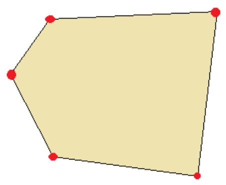

Figure 3: Extreme Point Solution Example

Extreme point solutions are used in the next two lemmas:

Lemma 1.1. *Let $P = \{x : Ax = b, x \geq 0\}$ and assume that the optimum value $\min\{c^T x : x \in P\}$ is finite. Then for any feasible solution $x \in P$, there exists an extreme point solution $x' \in P$ with $c^T x' \geq c^T x$.*

Lemma 1.2. *(***Rank Lemma***) Let $P = \{x : Ax = b, x \geq 0\}$ and let x be an extreme point solution of P such that $x_i > 0$ for each i. Then the number of variables is equal to the number of linearly independent constrains of A. i.e. the rank of A is the number of variables.*

Using the above two lemmas it is possible to prove the correctness of the algorithm described earlier. We do that by proving that in each iteration there is a variable with value of 0/1 in the matching. We will prove that by contradiction. Assume that no x_{uv} receives 0 or 1 during the iteration. Let us denote the number of remaining vertices in V_1 (or V_2) by n.

Step 1: Let's look at a vertex $v \in V_1$. Since each edge received a value less than 1 and the sum of all edges connected to v must be 1, there must be at least 2 edges connected to v. This is true for all vertices in V_1. So there are at least $2n$ vertices in the graph.

Step 2: We have $2n$ constraints (1 per each vertex). It is easy to see that the $2n$ constraints have dependencies between them (since the sum of constraints for V_1 are equal to the sum for V_2). So at most we have $2n - 1$ independent constraints. Looking at the Rank lemma, this means that there are at most $2n - 1$ variables (or vertices).

Step 3: The upper bound we found at Step 2 is $2n - 1$ vertices. The lower bound we found in Step 1 is $2n$ vertices. Therefore we have a contradiction. Hence, there is always a variable of value 0 or 1.

This proves that the algorithm works correctly.

In the next section we give basic background for linear programing and how should linear programs be approached.

2 Linear Programming

In order to fully understand the iterative method we first need to understand linear programming. The subject of linear programming is extensively explored in the world of computer science and is addressed by many books. In this section we use chapter 2.1 of [1] for the main reference as well as [18] chapter 12 and [7]. For further study regarding linear programming and the methods of solving and analyzing linear programs, the user is referred to [4], [5] and [8].

As seen in the previous chapter, a linear program can be (and usually is) expressed as follows (matrix notation):

$$\begin{aligned} \textit{minimize} \quad & c^T x \\ \textit{subject to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

We say that x is *feasible* if it satisfies the constraints of the linear program ($Ax \geq b$ and $x \geq 0$ for the linear program above). A linear program can be *feasible* or *infeasible* depending on whether a feasible solution exists. A solution x^* is called *optimal* if it fulfills $c^T x^* = \min\{c^T x : Ax \geq b, x \geq 0\}$. A linear program is called *unbounded* (from below) if $\forall \lambda \in \mathbb{R}, \exists$ feasible x such that $c^T x < \lambda$.

In some cases a linear program may be presented in a max form (i.e. maximize $c^T x$ and $\leq$ constraints). It is easy to show that any maximization problem can be easily turned into an equivalent minimization problem, and in this work we will also use the maximization version of the problem.

Before understanding the usage of iterative algorithms in linear programs, we first need to understand how extreme points affect the linear programming environment. First, let us define the following:

Definition 2.1. *An extreme point solution x of a polytope (a geometric object with flat sides over a general number of dimensions) P is* **integral** *if each coordinate of x is an integer. A polytope P is said to be* **integral** *if every vertex of P is integral.*

The following lemma was already referred to in the previous chapter. Now we prove it.

Lemma 2.1. *Let $P = \{x : Ax = b, x \geq 0\}$ and assume that $\min\{c^T x : x \in P\}$ is finite. Then for every $x \in P$, there exists an extreme point solution $x' \in P$ such that $c^T x' \leq c^T x$. i.e. there is always an extreme point optimal solution.*

Proof. Let x be an optimal solution. If x is an extreme point we are done. Otherwise, we will show that it is possible to find an extreme point solution. If x is a solution and not extreme point then by definition, there exists $y \neq 0$ such that $x + y \in P$ and $x - y \in P$. This leads to

$$A(x+y) \geq b, x+y \geq 0$$
$$A(x-y) \geq b, x-y \geq 0$$

We want to show that we can find a new optimal solution with more zero coordinates or tighter constraints. Let A' be a submatrix of A obtained by choosing only rows for ehich equality holds at x, and let b' be the vector b restricted to these rows, and b' be a vector comprised of elements from b (so $A'x = b'$). Since x is not an extreme point, there is a $y \neq 0$ such that $A'y \geq 0$ and $A'(-y) \geq 0$, therefor $A'y' = 0$. Since x is optimal we get:

$$\begin{aligned} c^T x &\leq c^T(x+y) \\ c^T x &\leq c^T(x-y) \\ &\Rightarrow c^T y = 0 \end{aligned}$$

We know that $y \neq 0$. So there must be at least one j such that $y_j < 0$ (in y or $-y$). Lets look at $x + \lambda y$ for $\lambda > 0$ and increase λ until $x + \lambda y$ is no longer feasible due to the non-negativity constraints on x:

$$\lambda^* = \min\{\min_{j:y_j<0} \frac{x_j}{-y_j}, \min_{i:A_i x > b_i, A_i y < 0} \frac{A_i x - bi}{-A_i y}\}$$

Since $x + y \geq 0$ and $x - y \geq 0$, either both x_i and y_i are zeros or none of them is zero. Therefore, the coordinates that were at 0, remain at 0. Also,

$A'(x+y) = A'x = b$ since $A'y = 0$. Therefore, the tight constraints remain tight. We assumed $\min\{c^T x : x \in P\}$ is finite, so we get that λ^* is finite and the solution $x + \lambda^* y$ has one more zero coordinate (when $\lambda^* = (x_j)/(-y_j)$) or one extra tight constraint ($\lambda* = (A_i x - b_i)/(-A_i y)$). So $x + \lambda^* y$ is an optimal solution with more zeroes or tight constraints.

Using this method we can convert any optimal solution to an extreme point optimal solution. $\square$

The proof of Lemma 2.1 not only proves that there is an extreme point solution, it also shows us that it is possible to find it.

Lemma 2.2. *Let $P = \{x : Ax = b, x \geq 0\}$. For $x \in P$, let A' be the submatrix of A restricted to rows which are at equality at x, and let A'_x denote the submatrix of A' consisting of the columns corresponding to the nonzeros in x. Then x is an extreme point solution if and only if A'_x has linearly independent columns (i.e. A'_x has full column rank).*

We will skip the proof of this lemma. It can be found in [1].

Lemma 2.3. *(***Rank Lemma***) Let $P = \{x : Ax = b, x \geq 0\}$ and let x be an extreme point solution of P such that $x_i > 0$ for each i. Then any maximal number of linearly independent tight constrains of the form $A_i x = b_i$ for some row i of A equals the number of variables.*

Proof. We know that $x_i > 0$. We have $A'_x = A'$. From Lemma 2.2 it follows that A' has full column rank. Since the number of columns equals the number of non-zero variables in x and row rank of any matrix equals the column rank, we have that row rank of A' equals the number of variables. Then any maximal number of linearly independent tight constraints is exactly the maximal number of linearly independent rows of A' which is exactly the row rank of A' and hence the claim follows. $\square$

In the next part we introduce possible ways to solve the linear program.

Let us consider the problem:

$$\begin{aligned} \textit{minimize} \quad & c^T x \\ \textit{subject to} \quad & Ax \geq b \\ & x \geq 0 \end{aligned}$$

In order to solve it, we first wish to introduce a variable s_j for each constraint in order to obtain the *standard form*:

$$\begin{aligned} \textit{minimize} \quad & c^T x \\ \textit{subject to} \quad & Ax + s = b \\ & x \geq 0 \\ & s \geq 0 \end{aligned}$$

Now we can use the linear program in the standard form (i.e. equality instead of greater or equal: $Ax = b$). We assume that A is of full row rank. If this is not the case we can remove dependent constraints and reduce the matrix size without affecting the optimal solution.

A subset of columns B of the constraint matrix A is called a *basis* if the matrix of columns corresponding to B, i.e. A_B, is invertible. A solution x is called *basic* if and only if there is a basis B such that $x_j = 0$ if $j \notin B$ and $x_B = A_B^{-1} b$. If in addition to being basic, it is also feasible, i.e., $A_B^{-1} b \geq 0$, it is called a *basic feasible solution.* There can be many bases which correspond to the same basic feasible solution. The next theorem shows the equivalence of extreme point solutions and basic feasible solutions.

Theorem 2.1. *Let A be a $m \times n$ matrix with full row rank. Then every feasible x to $P = \{x : Ax = b, x \geq 0\}$ is a basic feasible solution if and only if x is an extreme point solution.*

The proof for this theorem can be found in [1] Theorem 2.1.5.

There are number of ways to solve a linear program. The *simplex* algorithm is one of them (see [12] for further study). It works on the standard form of the problem. The basic idea behind the simplex algorithm is to start with a basic feasible solution and to move to a *neighboring* basic feasible solution which improves the objective function. This is done repeatedly until an optimal basic feasible solution is found. The main problem with the simplex algorithm is that for some inputs it runs in exponential time.

There are several known polynomial time algorithms for linear programming problems, among them the ellipsoid method (see [10]) and the interior point method [11]. There are also efficient algorithms that compute a *near* optimal solution in polynomial time.

Theorem 2.2 (***Optimal Solution Theorem***)**.** *There exists a polynomial time algorithm that given an instance of a linear programming problem, either returns an optimal extreme point solution, or determines that the problem has no feasible solution.*

In the last part of this section we introduce the dual problem. A linear program of the following type:

$$\begin{aligned} &\textit{minimize} && \sum_{j=1}^{n} c_j x_j && \\ &\textit{subject to} && \sum_{j=1}^{n} a_{ij} x_j \geq b_i && \forall 1 \leq i \leq m \\ & && x_j \geq 0 && \forall 1 \leq j \leq n \end{aligned}$$

has the following dual program:

$$\begin{aligned} &\textit{maximize} && \sum_{i=1}^{m} b_i y_i && \\ &\textit{subject to} && \sum_{i=1}^{m} a_{ij} y_i \leq c_j && \forall 1 \leq j \leq n \\ & && y_i \geq 0 && \forall 1 \leq i \leq m \end{aligned}$$

The following theorem is useful. It will not be proved here, but we will use it later on.

Theorem 2.3 (***Strong Duality Theorem***)**.** *If the primal linear program has an optimal solution, so does its dual, and the respective optimal values are equal.*

The subject of linear programming is very complex and can be further studied for a long time. The reader is referred to linear programming literature such as [7].

3 Matching and Vertex Cover in Bipartite Graph

3.1 Matching in Bipartite Graph

In this section we analyze problems over bipartite graphs, based on chapter 3 of [1].

The first problem we examine is *Matchings in Bipartite Graphs.* This problem also appeared in the introduction section. In this section we repeat it with a more formal approach.

Problem 3.1. (Matchings in Bipartite Graphs) *Given a bipartite graph* $G = (V_1 \cup V_2, E)$ *and weight function* $w\colon E \to R$*, the maximum matching problem is to find a set of vertex-disjoint edges of maximum total weight.*

Let's look at the following example (illustrated in Figure 4):

Example 3.1. $V_1 = \{a, b\}$ *and* $V_2 = \{c, d\}$*.* $E = \{(a, c), (a, d), (b, c), (b, d)\}$*. The weight function is defined as follows:* $w(a, c) = 1$*;* $w(a, d) = 4$*;* $w(b, c) = 4$*;* $w(b, d) = 2$*.*

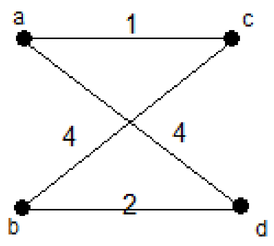

Figure 4: Max Matching in Bipartite Example

The matching includes the marked edges as can be seen in Figure 5.

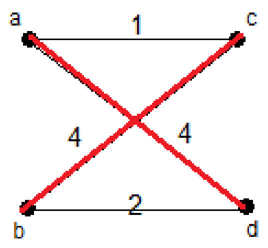

Figure 5: Max Matching in Bipartite Example - Solution

The first thing we do is to find the linear program representing the problem (we actually look at the relaxation of the problem):

$$\begin{aligned}
&\textit{maximize} && \sum_{e\in E} w_e x_e && \\
&\textit{subject to} && \sum_{e\in\delta(v)} x_e \le 1 && \forall v \in V_1 \cup V_2 \\
& && x_e \ge 0 && \forall e \in E
\end{aligned}$$

From now we refer to this problem as $LP_{bm}(G)$. Here $\delta(S)$ (where S is a set of vertices) denotes the set of edges which have exactly one endpoint in S. In our case S has only one vertex. We will use this definition in the entire paper. Let us note that the number of constraints is similar to the number of vertices in the problem. This linear program can be solved optimally in polynomial time as stated in previous theorem (see Theorem 2.2). Let's look at the linear problem for Example 3.1:

$$\begin{aligned}
&\textit{maximize} && \sum_{e\in E} w_e x_e && \\
&\textit{subject to} && x_{ac} + x_{ad} \le 1 && \\
& && x_{bc} + x_{bd} \le 1 && \\
& && x_{ac} + x_{bc} \le 1 && \\
& && x_{ad} + x_{bd} \le 1 && \\
& && x_e \ge 0 && \forall e \in E
\end{aligned}$$

Before continuing we define the characteristic vector:

Definition 3.1. *Let $\chi(F)$ where $F \subseteq E$ denote the vector $\mathbb{R}^{|E|}$ that has a 1 corresponding to each edge $e \in F$, and 0 otherwise. This vector is called the* characteristic vector *of F.*

Now that $\chi(F)$ is defined, we characterize the problem's extreme points solution. We will use the Rank Lemma (Lemma 2.3) in the following way. The Rank Lemma says that the number of variables (i.e. number of edges $|E|$) is equal to the number of tight linearly independent constraints. In the following lemma we see the three requirements (size, tightness and linearly independence).

Lemma 3.1. *Given any extreme point solution x to $LP_{bm}(G)$ such that $x_e > 0$ for each $e \in E$ there exists $W \subseteq V_1 \cup V_2$ such that*

1. *$x(\delta(v)) = 1$ for each $v \in W$.*
2. *The vectors in $\{\chi(\delta(v)) : v \in W\}$ are linearly independent.*
3. *$|W| = |E|$.*

Now we are ready to present the iterative algorithm.

3.1.1 The iterative Algorithm

Algorithm 1 Iterative Bipartite Matching Algorithm

```
1: $F \leftarrow \emptyset$
2: **while** $E(G) \neq \emptyset$ **do**
3:     (a) Find an optimal extreme point solution $x$ to $LP_{bm}(G)$ and remove
       every edge $e$ with $x_e = 0$ from $G$.
4:     (b) If there is an edge $e = \{u, v\}$ with $x_e = 1$, then update $F \leftarrow F \cup \{e\}$,
       $V_1 \leftarrow V_1 \setminus \{u\}$ and $V_1 \leftarrow V_2 \setminus \{v\}$.
5: **end while**
6: Return $F$.
```

3.1.2 Correctness

In order to prove that the algorithm returns an optimal solution we need to prove the following two things. First, that if the algorithm returns a solution, it is optimal. Second, that the algorithm actually works (i.e. the while loop actually ends).

In order to prove that the solution is optimal we use the following claim:

Claim 3.1. *If the algorithm, in every iteration, finds an edge e with $x_e = 0$ or an edge e with $x_e = 1$, then it returns a matching F of weight at least the optimal solution to $LP_{bm}(G)$.*

Proof. By induction on the number of iterations (n).

For the base case ($n = 1$) the result is trivial.

For ($n > 1$) we assume that we either find an edge e with $x_e = 0$ or $x_e = 1$.

If $x_e = 0$ then according to line 3 we remove the edge from the graph G and apply the algorithm on the new G' which is now smaller. By induction the solution to the new problem is optimal for the new G'. Now we only need to understand why this solution is also optimal for the original G. Let us mark the solution for $LP_{bm}(G')$ as $x_{G'}$. Since the edge we removed had $x_e = 0$, we can assume that $w \cdot x_{G'} = w \cdot x$. Since $x_{G'}$ is a solution to the relaxation of the $LP_{bm}(G')$ problem we know that it is at least as good as the optimal solution to the problem. Since $w \cdot x_{G'} = w \cdot x$, x is also optimal.

If $x_e = 1$ then the solution contains e and we still need to find a solution for $LP_{bm}(G')$ which is similar to $LP_{bm}(G)$ without vertices u and v (where u and v are the endvertices of e) and all edges that have u and v as end-vertices. Just like in the previous case we end up with an optimal solution for $LP_{bm}(G')$. Let us mark this solution (ie. set of edges) as F'. Since it is optimal we know that $w(F') \geq w \cdot x_{G'}$. The algorithm as a whole returns the solution $F = F' \cup e$ and we get:

$$w(F) = w(F') + w_e \text{ and } w(F') \geq w \cdot x_{G'}$$

which implies that

$$w(F) \geq w \cdot x_{G'} + w_e = w \cdot x$$

This proves our claim. □

Now we need to prove that the algorithm always finds and edge with $x_e = 0$ or $x_e = 1$, namely, we need the following lemma.

Lemma 3.2. *Given any extreme point solution x of $LP_{bm}(G)$ (with $E \neq \emptyset$) there exists an edge e with $x_e = 0$ or $x_e = 1$.*

Proof. We will prove this by contradiction. Let us assume that no edge e has $x_e = 0$ or $x_e = 1$. This means that $0 < x_e < 1$ for all edges $e \in E$. According to Lemma 3.1 there exists $W \subseteq V_1 \cup V_2$ such that $|W| = |E|$ and the constrains corresponding to W are linearly independent. We want to show that for each $v \in W$ the rank of v is 2 (i.e. $d_E(v) = 2$) and the rank of any $v \notin W$ is 0 (i.e. $d_E(v) = 0$). Since $x(\delta(v)) = 1$ for each $v \in W$ and $0 < x_e < 1$ we must have at least 2 edges per each $v \in W$. And:

$$2|W| = 2|E| = \sum_{v \in V} d_E(v) \geq \sum_{v \in W} d_E(v) \geq 2|W|.$$

Therefore $d_E(v) = 2$ for $v \in W$ and $d_E(v) = 0$ otherwise. Now, since each vertex in W is connected to 2 edges from E, we can see that E is a cycle cover on W (a cycle cover of a graph G is a set of cycles which are subgraphs of G and contain all vertices of G). Let's examine a cycle C on W. Since our graph is bipartite C's length is even. Further more:

$$\sum_{v \in C \cap V_1} \chi(\delta(v)) = \sum_{v \in C \cap V_2} \chi(\delta(v))$$

This contradicts the independence constraint of the second condition in Lemma 3.1. Thus proving that not all edges fulfill $0 < x_e < 1$. □

Combining the claim and the lemma proved that the algorithm performs as required.

3.2 Vertex Cover in Bipartite Graph

Let us examine the following problem:

Problem 3.2. (Vertex Cover in Bipartite Graphs) *Given a bipartite graph $G = (V, E)$ with cost function $c\colon V \to \mathbb{R}_+$ the vertex cover problem asks for a set of vertices V' such that $e \cap V' \neq \emptyset$ for each $e \in E$ and $c(V') = \sum_{v \in V'} c_v$ is minimized.*

For example let us look at the following problem(illustrated in Figure 6):

Example 3.2. *The graph G is divided into two sets of vertices: $V_1 = \{a, b, c\}$ and $V_2 = \{d, e, f\}$. $E = \{(a, d), (a, e), (b, e), (c, d), (c, e), (c, f)\}$. The cost function is defined as follows: $c(a) = 2$; $c(b) = 1$; $c(c) = 3$; $c(d) = 3$; $c(e) = 1$; $c(f) = 2$.*

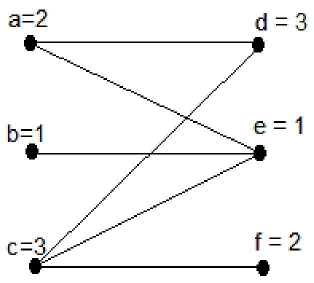

Figure 6: Vertex Cover in Bipartite Example

One possible cover includes the marked vertices as can be seen in Figure 7.

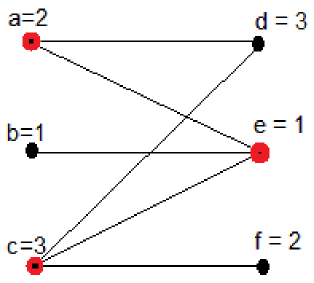

Figure 7: Vertex Cover in Bipartite Example - Solution

The linear program $LP_{pvc}(G)$ for the problem is as follows (x_v is an indicator for vertex v):

$$\begin{aligned} &\textit{minimize} && \sum_{v \in V} c_v x_v && \\ &\textit{subject to} && x_u + x_v \geq 1 && \forall e = \{u, v\} \in E \\ & && x_v \geq 0 && \forall v \in V \end{aligned}$$

For Example 3.2 this is translated to:

$$\begin{aligned} &\textit{minimize} && 2x_a + x_b + 3x_c + 3x_d + x_e + 2x_f && \\ &\textit{subject to} && x_a + x_d \geq 1 && \\ & && x_a + x_e \geq 1 && \\ & && x_b + x_e \geq 1 && \\ & && x_c + x_d \geq 1 && \\ & && x_c + x_e \geq 1 && \\ & && x_c + x_f \geq 1 && \\ & && x_v \geq 0 && \forall v \in V \end{aligned}$$

Since the number of constrains is equal to the number of edges (i.e. bounded) this problem can be solved optimally in polynomial time (see Theorem 2.2).

Definition 3.2. *The* characteristic vector *of $W \subseteq V$ is a vector in $\mathbb{R}^{|V|}$ that has an 1 corresponding to each vertex $v \in W$, and 0 otherwise. This vector is denoted by $\chi(W)$.*

Now we are ready to present the iterative algorithm:

3.2.1 The iterative Algorithm

3.2.2 Correctness

Similarly to the Matching problem, we prove the correctness in two steps. First, we prove that if the algorithm returns a solution, then it is optimal.

Algorithm 2 Iterative Bipartite Vertex Cover Algorithm

```
1: $U \leftarrow \emptyset$
2: while $V(G) \neq \emptyset$ do
3:     (a) Find an optimal extreme point solution $x$ to $LP_{bvc}(G)$ and remove
       every vertex $v$ with $x_v = 0$ and $d_E(v) = 0$ from $G$.
4:     (b) If there is an vertex $v \in V$ with $x_v = 1$, then update $U \leftarrow U \cup \{v\}$,
       $V(G) \leftarrow V(G) \setminus \{v\}$ and $E(G) \leftarrow E(G) \setminus \delta(v)$.
5: end while
6: Return $U$.
```

Second, we prove that the algorithm actually ends and returns a solution.

Claim 3.2. *If the algorithm finds a vertex v with $x_v = 0$ and $d_E(v) = 0$ or a vertex v with $x_v = 1$ inside the* while *loop, then it returns a vertex cover U of cost at most the optimal solution to $LP_{bvc}(G)$.*

Since the proof of this claim is almost the same as Claim 3.1 we will not elaborate on it.

Consider the following lemma (it is derived by the Rank Lemma in a similar manner to Lemma 3.1).

Lemma 3.3. *Given any extreme point x to $LP_{bvc}(G)$ with $x_v > 0$ for each $v \in V$ there exists $F \subseteq E$ such that*

1. *$x_u + x_v = 1$ for each $e = \{u, v\} \in F$.*
2. *The vectors in $\{\chi(\{u, v\}) : \{u, v\} \in F\}$ are linearly independent.*
3. *$|V| = |F|$.*

Using the above lemma we prove the following:

Lemma 3.4. *Given any extreme point solution x to $LP_{bvc}(G)$ there must exist a vertex v with $x_v = 0$ and $d_E(v) = 0$ or a vertex with $x_v = 1$.*

According to this lemma, the algorithm returns a cover of cost at most the cost of an optimal solution of $LP_{bvc}(G)$.

Proof. By contradiction, let us assume that $x_v < 1$ and $x_v = 0$ while $d_E(v) \geq 1$ for all $v \in V$. Let us look at an edge $e = \{u, v\}$. From the constraints in the $LP_{bvc}(G)$ we know that $x_u + x_v \geq 1$. Since we assumed that $x_v < 1$ then $x_u > 0$. In order to see that the assumption is incorrect we use Lemma 3.3. According to the lemma there exists $F \subseteq E$ such that $|F| = |V|$. We want to show that F is acyclic. If we show that F is acyclic then it implies that $|F| \leq |V| - 1$, giving a contradiction. Assume that $C \subseteq F$ is a cycle. C must be even since G is bipartite. Since G is bipartite, C is a disjoint union of two matchings, say M_1 and M_2. The sum of the constraints for edges in M_1 and M_2 is the same, but this contradicts the independence of the constraints according to lemma 3.3. Therefore C is acyclic. $\square$

3.3 Vertex Cover and Bipartite Matching Duality

As the reader probably noticed the vertex cover and maximum matching problems discussed in the previous sections seem very similar. It is possible to prove that on bipartite graphs they have the same value.

Theorem 3.1. *Given an unweighted bipartite graph $G = (V, E)$ we have*

$$\max\{|M| : M \textit{ is a matching}\} = \min\{|U| : U \textit{ is a vertex cover}\}$$

Proof. First, let us remind the reader Theorem 2.3: The dual linear program has an optimal solution that is equal to the optimal solution of the primal problem. For simplicity, let us assume that the cost functions we use are constant 1 (the following can also be proved when the costs are different). Let M^* be a matching returned by Algorithm 1 and U^* be a cover returned by Algorithms 2. We already know that the solutions are optimal for both problems. If we consider the dual problem for the matching, we see that it is the same as the vertex cover primal problem and vice versa. Now we use the strong duality theorem (Theorem 2.3) and get that both programs have optimal solutions of equal value. $\square$

4 Spanning Trees

In this section we explore the spanning tree problems. The exploration of the spanning tree problems is based on chapter 4 of [1].

4.1 Minimum Spanning Tree

Let us examine the following problem:

Problem 4.1. Minimum Spanning Tree (MST) *Given an undirected graph* $G = (V, E)$ *edge costs* $c\colon E \to \mathbb{R}$*, and the task is to find a spanning tree of minimum total edge cost.*

Let us consider the following example (illustrated in Figure 8):

Example 4.1. *For the graph* G *the vertices are:* $V = \{a, b, c, d\}$ *and* $E = \{(a, b), (a, c), (a, d), (b, c), (b, d), (c, d)\}$*. The cost function is defined as follows:* $c(a, b) = 1$*;* $c(a, c) = 1$*;* $c(a, d) = 4$*;* $c(b, c) = 4$*;* $c(b, d) = 2$*;* $c(c, d) = 3$.

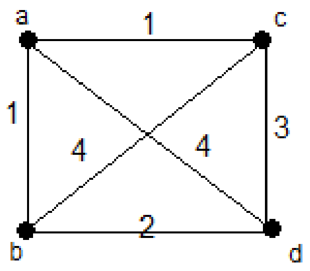

Figure 8: Minimum Spanning Tree Example

For this problem the minimum tree is obvious as can be seen in the following Figure 9.

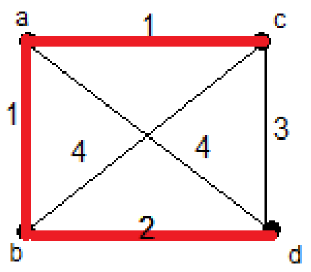

Figure 9: Minimum Spanning Tree Solution Example

4.1.1 The Linear Program Relaxation

Let us define x_e as an indicator that states whether edge e is included in the spanning tree. Let c_e denote the cost of edge e. For a subset F of edges, let $x(F)$ denote $\sum\limits_{e \in F} x_e$. For a subset S of vertices, $\delta(S)$ is the set of edges with exactly one endpoint in S. Finally let $E(S)$ be the set of edges with both endpoints in S.

The following describes the so called *subtour linear program relaxation* for MST:

$$\begin{array}{lll}
\textit{minimize} & \sum\limits_{e \in E} c_e x_e & \\
\textit{subject to} & x(E(S)) \leq |S| - 1 & \forall \emptyset \neq S \subset V \\
& x(E(V)) = |V| - 1 & \\
& x_e \geq 0 & \forall e \in E
\end{array}$$

The constraints demand that for each subset of V, the sum of all x_e for edges inside the set is at most $|S| - 1$. This means that there is at least one edge going out of the set. In addition, we require that sum of all edges is $|V| - 1$. So we know that all vertices are connected once.

Let us note that even though the number of constraints in this problem is exponential in $|V|$, it is still possible to solve it in time polynomial in $|V|$ (see [1] section 4.1.2).

Now we wish to analyze the extreme point solution. Remember that an extreme point solution is a unique solution that fulfills the following: it is defined by n linearly independent tight inequalities (n is the number of variables in the LP). Since our LP has exponential number of inequalities, some may not be satisfied as equalities. In order to analyze the extreme point solution, we need to find a "good" set of tight inequalities. We will ignore edges with $x_e = 0$ as these edges can be removed from the graph without affecting the feasibility of the objective value.

Let us examine the *Uncrossing Technique* (see [9]). This technique is used to find a *good* set of tight inequalities for n extreme point solution in the subtour LP. Let $\chi(F)$ $(F \subseteq E)$ denote the characteristic vector in $\mathbb{R}^{|E|}$ that has a 1 corresponding to each edge $e \in F$ and 0 otherwise. Then:

Proposition 4.1. *For $X, Y \subseteq V$,*

$$\chi(E(X)) + \chi(E(Y)) \leq \chi(E(X \cup Y)) + \chi(E(X \cap Y))$$

and equality holds if and only if $E(X \setminus Y, Y \setminus X) = \emptyset$.

Proof. Immediate from

$$\chi(E(X)) + \chi(E(Y)) = \chi(E(X \cup Y)) + \chi(E(X \cap Y)) - E(X \setminus Y, Y \setminus X)$$

□

Given an extreme point solution x to the subtour LP, let $\mathcal{F} = \{S \subseteq V \mid x(E(S)) = |S| - 1\}$ be the family of tight inequalities for x. The following lemma shows that this family is closed under intersection and union.

Lemma 4.1. *If $S, T \in \mathcal{F}$ and $S \cap T \neq \emptyset$, then both $S \cap T$ and $S \cup T$ are in $\mathcal{F}$. Furthermore, $\chi(E(S)) + \chi(E(T)) = \chi(E(S \cap T)) + \chi(E(S \cup T))$.*

We will skip this proof. It can be found in Lemma 4.1.4. in [1].

Definition 4.1. span$(\mathcal{F})$ *is defined as the vector space generated by the set of vectors $\{\chi(E(S)) \mid S \in \mathcal{F}\}$.*

Definition 4.2. *Two sets X, Y are* intersecting *if $X \cap Y$, $X - Y$ and $Y - X$ are nonempty.*

Definition 4.3. *A family of sets is* laminar *if no two sets are intersecting.*

We use the above 3 definitions in order to state that an extreme point solution is characterized by tight inequalities whose corresponding sets form a laminar family.

Lemma 4.2. *If $\mathcal{L}$ is a maximal laminar subfamily of $\mathcal{F}$, then span$(\mathcal{L})$ = span$(\mathcal{F})$*

The proves for Lemma 4.2 is very mathematical and it is available in [1] under Lemma 4.1.5.

Definition 4.4. *A* singleton *is defined as a subset with only one element.*

Proposition 4.2. *A laminar family $\mathcal{L}$ over the ground set V without singletons has at most $|V| - 1$ distinct members.*

Now we are ready to present the iterative algorithm.

4.1.2 Leaf-finding Iterative Algorithm

Algorithm 3 Iterative Leaf-finding MST Algorithm

```
1: $F \leftarrow \emptyset$
2: while $V(G) \geq 2$ do
3:     (a) Find an optimal extreme point solution $x$ to the subtour $LP$ and
       remove every edge $e$ with $e_v = 0$ from $G$.
4:     (b) Find a vertex $v$ with at most one edge $e = uv$ incident to it, then
       update $F \leftarrow F \cup \{e\}$ and $G \leftarrow G \setminus \{v\}$.
5: end while
6: Return $F$.
```

4.1.3 Correctness

We start by proving that the algorithm terminates.

Lemma 4.3. *For any extreme point solution x to the subtour LP with $x_e > 0$ for every edge e, there exists a vertex v with $d(v) = 1$*

Proof. By contradiction let's assume that each vertex has $d(v) > 1$. This means that $\frac{1}{2}\sum_{v\in V} d(v) \geq |V|$. On the other hand, since there are no edges with $x_e = 0$ all tight inequalities are in the form of $\chi(E(S)) = |S| - 1$. By Lemma 4.2, there are $|\mathcal{L}|$ linearly independent tight constraints of the form $\chi(E(S)) = |S| - 1$, where $\mathcal{L}$ is a laminar family with no singleton sets. As a result $|E| = |\mathcal{L}|$ by the Rank Lemma 2.3. By Proposition 4.2, $|\mathcal{L}| \leq |V| - 1$ and therefore $|E| \leq |V| - 1$ which contradicts the original assumption. □

Once we established that the algorithm terminates, we need to see that it returns a minimum spanning tree.

Theorem 4.1. *The Iterative MST Algorithm returns a minimum spanning tree in polynomial time.*

Proof. By induction on the number of iterations of the algorithm. Let's assume the algorithm finds a vertex v with $d(v) = 1$ (i.e. leaf) in line 4 (step (b)) of the loop. The edge e connects v to the rest of the graph. Then $x_e = 1$ since $x(\delta(v)) \geq 1$ is a valid inequality of the LP. The algorithm removes e from the graph and adds it to the spanning tree. Now we are left with a smaller problem (i.e. to solve the MST problem for $G \setminus \{v\}$). Once we know the solution to the smaller problem, we can add e to the solution for a full MST. It is easy to see that the cost of such tree is optimal. □

5 Directed Graphs

In this chapter we discuss directed graphs and how the iterative technique can help us solve problems on directed graphs. This section is based on chapter 6 of [1].

5.1 Minimum Cost Arborescence

Problem 5.1. *Given a directed graph $D = (V, A)$ and a root vertex $r \in V$, a spanning r-arborescence is a subgraph of D so that there is a directed path from r to every vertex $V - r$. The minimum spanning arborescence problem is to find a spanning r-arborescence with the minimum total cost.*

For example let us look at the following example (illustrated in Figure 10):

Example 5.1. *For the graph G the vertices are: $V = \{a, b, c, d\}$ and $E = \{(a, b), (a, c), (a, d), (b, c), (b, d), (c, d)\}$ (all edges are directed from first vertex to second). The cost function is defined as follows: $c(a, b) = 1$; $c(a, c) = 1$; $c(a, d) = 4$; $c(b, c) = 4$; $c(b, d) = 2$; $c(c, d) = 3$.*

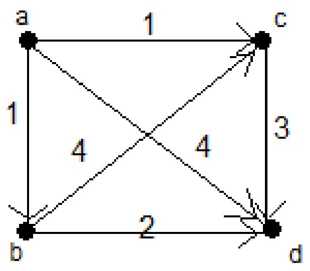

Figure 10: Minimum Cost Arborescence

For this problem the minimum spanning arborescence is obvious as can be seen in the following Figure 11.

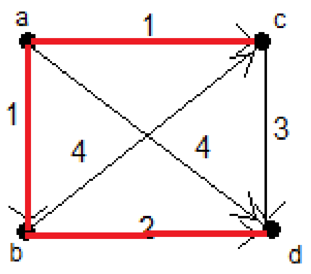

Figure 11: Minimum Cost Arborescence Solution Example

As in previous sections we will first find the linear program and then characterize the extreme point solutions. Only then we will explore the iterative algorithm.

5.1.1 Linear Programming Relaxation

Here we define the linear program for the solution of Problem 5.1

In the following linear program c_a is the cost of arc a and x_a is an indicator for the choosing of arc a. We mark $\delta^{in}(S)$ as the set of all arcs incoming to a vertex $v \in S$ from any vertex outside of $S \subseteq V$. The expression $\chi(\delta^{in}(S))$ represents a vector in $\mathbb{R}^{|A|}$ for set S. This vector has 1 for each arc $a \in \delta^{in}(S)$, and 0 otherwise. This vector is called the characteristic vector of $\delta^{in}(S)$ and is denoted by $\chi(\delta^{in}(S))$. The term $x(\delta^{in}(S))$ means the sum $\sum\limits_{a \in \delta^{in}(S)} x_a$.

$$\begin{array}{lll} minimize & \sum\limits_{a \in A} c_a x_a & \\ subject\ to & x(\delta^{in}(S)) \geq 1 & \forall S \subseteq V - r \\ & x_a \geq 0 & \forall a \in A \end{array}$$

The linear program simply states that for each subset of V without r ($S \subseteq V - r$) we can find at least one arc that is coming into S from $A - S$.

5.1.2 Characterization of Extreme Point Solutions

In this section we analyze the extreme point solution to the minimum spanning arborescence. The reader may note that the conclusions and lemmas in this section are very similar to the ones in the Minimum Spanning Trees section.

Just like in the minimum spanning trees, we use the *uncrossing technique* to find a *good* set of tight inequalities that define an extreme point solution to the directed LP. In this section we will use $\mathcal{F}$ to represent this family of tight inequalities ($\mathcal{F} = \{S | x(\delta^{in}(S)) = 1\}$).

Proposition 5.1. *For $X, Y \subseteq V$,*

$$x(\delta^{in}(X)) + x(\delta^{in}(Y)) \geq x(\delta^{in}(X \cup Y)) + x(\delta^{in}(X \cap Y))$$

and equality holds if and only if $E(X \setminus Y, Y \setminus X) = \emptyset$.

This is similar to proposition 4.1.

Now we want to see that $\mathcal{F}$ is closed under intersection and union:

Lemma 5.1. *If $S, T \in \mathcal{F}$ and $S \cap T \neq \emptyset$ then both $S \cap T$ and $S \cup T$ are in $\mathcal{F}$. Furthermore, $\chi(\delta^{in}(S)) + \chi(\delta^{in}(T)) = \chi(\delta^{in}(S \cap T)) + \chi(\delta^{in}(S \cup T))$*

The proof for this can be found in [1] lemma 6.1.2.

In this section we use the following definition:

Definition 5.1. span($\mathcal{F}$) *is defined as the vector space generated by the set of vectors $\{\chi(\delta^{in}(S)) \mid S \in \mathcal{F}\}$.*

Let us also remind the reader that a family of sets is *laminar* if no two sets are intersecting.

Lemma 5.2. *If $\mathcal{L}$ is a maximal laminar subfamily of $\mathcal{F}$, then span($\mathcal{L}$) = span($\mathcal{F}$)*

This is similar to Lemma 4.2 and will not be proven here.

Corollary 5.1. *Let x be any extreme point solution to the directed LP. Then there exists a laminar family $\mathcal{L}$ such that:*

1. *$x(\delta^{in}(S)) = 1$ for all $S \in \mathcal{L}$.*

2. *The vectors in $\{\chi(\delta^{in}(S)) : S \in \mathcal{L}\}$ are linearly independent.*

3. $|A| = |L|$.

A laminar family $\mathcal{L}$ defines naturally a forest L as follows: Each node of L corresponds to a set in $\mathcal{L}$, and there is an arc from set R to set S if R is the smallest set containing S. R is called the *parent* of S, and S is called the *child* of R. A node with no parent is called a *root*, and a node with no children is called a *leaf*. Given a node R, the *subtree rooted* at R consists of R and all its descendants. The forest L corresponding to the laminar family $\mathcal{L}$ will be used to perform the token counting arguments inductively.

5.1.3 The Iterative Algorithm

The following is an iterative algorithm for the directed LP. This algorithm is very similar to the MST algorithm in the previous chapter.

Algorithm 4 Iterative Minimum Spanning Arborescence Algorithm

```
1: $F \leftarrow \emptyset$
2: while $|V(D)| \geq 2$ do
3:     (a) Find an optimal extreme point solution $x$ to the directed $LP$ and
       remove every arc $a$ with $x_a = 0$ from $D$.
4:     (b) Find an arc $a = uv$ with $x_a = 1$ and update $F \leftarrow F \cup \{a\}$ and
       $D \leftarrow D \setminus \{uv\}$.
5: end while
6: Return $F$.
```

5.1.4 Correctness and Optimality

The first step in proving the correctness and optimality is to prove that the algorithm terminates.

Lemma 5.3. *For any extreme point solution x to the directed LP, either there is an arc with $x_a = 0$ or there is an arc with $x_a = 1$.*

Proof. We explain the idea behind the proof (for a full proof see [1] Lemma 6.1.6). The proof is by contradiction – assume that there is no arc with $x_a = 0$ or $x_a = 1$. We can use the laminar family $\mathcal{L}$ from Corollary 5.1 in order to create the contradiction. The idea is to show that the number of constraints (number of sets in $\mathcal{L}$) is smaller than the number of arcs. The proof is very similar to the MST problem proof. $\square$

Now that we know that the algorithm terminates (since the number of iterations is now limited) we only need to prove that the algorithm runs in polynomial time. The proof for this is roughly the same as the proof for Theorem 4.1. The reader may refer to it for further study of the case.

6 Matching

In the previous sections we met the maximum matching problem in bipartite graphs. In this section we want to explore the matching in a general weighted undirected graph. This section is based on chapter 9 of [1].

6.1 Graph Matching

When we examined the bipartite matching we used a linear programming relaxation to the problem. For general graphs this relaxation does not work. Let us start by defining the problem:

Problem 6.1. *Given an undirected graph $G = (V, E)$ with a weight function $w\colon E \to R$ the maximum matching problem is to find a set of vertex-disjoint edges of maximum total weight.*

6.1.1 Linear Programming Relaxation

Before looking at the linear program relaxation let us recall that $E(S)$ denotes the set of edges with both endpoints in $S \subseteq V$ and $x(F)$ is short for $\sum_{e\in F} x_e$ for $F \subseteq E$.

The linear programming relaxation for the maximum matching problem is given by the following $LP_M(G)$ (remember that $\delta(x)$ is defined as a set of edges that have x as one of their endpoints).

due to Edmonds [13]:

$$\begin{array}{lll} \textit{maximize} & \displaystyle\sum_{e\in E} w_e x_e & \\ \textit{subject to} & x(\delta(v)) \leq 1 & \forall v \in V \\ & x(E(S)) \leq \dfrac{|S|-1}{2} & \forall S \subset V, |S| \textit{ odd} \\ & x_e \geq 0 & \forall e \in E \end{array}$$

6.1.2 Characterization of Extreme Point Solutions

In this section we define $\chi(F)$ in the following way: For a subset of edges $F \subseteq E$, $\chi(F)$ is the characteristic vector $\chi(F) \in \mathbb{R}^{|E|}$ with an 1 corresponding to each edge in F and 0 otherwise.

Lemma 6.1. *Let x be an extreme point solution to $LP_M(G)$ with $0 < x_e < 1$ for each edge $e \in E(G)$. Then there exists a laminar family $\mathcal{L}$ of odd-sets and a set of vertices $T \subseteq V$ such that:*

1. *$x(E(S)) = (|S| - 1)/2$ for each $S \in \mathcal{L}$ and $x(\delta(v)) = 1$ for each $v \in T$.*
2. *The vectors in $\{\chi(E(S)) : S \in \mathcal{L}\} \cup \{\chi(\delta(v)) : v \in T\}$ are linearly independent.*
3. *$E(S)$ is connected for each set $S \in \mathcal{L}$.*
4. *$|E| = |\mathcal{L}| + |T|$.*

The proof for Lemma 6.1 is relatively long and mathematical. The reader can find the detailed proof in [1] under Lemma 9.1.2.

6.1.3 The Iterative Algorithm

Algorithm 5 Iterative Matching Algorithm

```
1: $F \leftarrow \emptyset$
2: while $|V(G)| \neq \emptyset$ do
3:     (a) Find an optimal extreme point solution $x$ to the directed $LP_M(G)$
       and remove every edge $e$ with $x_e = 0$ from $G$.
4:     (b) If there is an edge $e = \{u, v\}$ with $x_e = 1$ then update $F \leftarrow F \cup \{e\}$
       and $G \leftarrow G \setminus \{u, v\}$.
5: end while
6: Return $F$.
```

6.1.4 Correctness and Optimality

If we knew that during each iteration the algorithm would find an edge e with $x_e = 1$, then the returned solution would be optimal, by induction. So what we need to show is that such an edge always exists.

Lemma 6.2. *Given any extreme point x to $LP_M(G)$ there must exist an edge e with $x_e = 0$ or $x_e = 1$.*

Proof-sketch: By contradiction, let's assume that $0 < x_e < 1$ for all edges $e \in E$. Let $\mathcal{L}$ be a laminar family of tight odd-sets and T be a set of tight vertices satisfying the properties of Lemma 6.1. Let $\mathcal{L}' = \mathcal{L} \cup T$ be the extended laminar family. If we show that $|E| = |\mathcal{L}| + |T| = |\mathcal{L}'|$ then by contradiction our proof is done. We will use token counting argument. In the beginning we give each edge a token (total of $|E|$ tokens). Then each edge gives its token to the smallest set in $\mathcal{L}'$ that contains both of its endpoints. Now we redistribute the tokens inductively so that each member in $\mathcal{L}'$ has one token. Naturally there are some tokens left. This implies that $|L| > |\mathcal{L}'|$. The complete proof is found in [1] Lemma 9.1.7.

6.2 Hypergraph Matching

In this section we consider matching problems on hypergraphs. It is easy to understand what what hypergraphs from the following definitions:

Definition 6.1. *A* **hypegraph** $H = (V, E)$ *consists of a set V of vertices and a set E of hyperedges.*

Definition 6.2. *A* **hyperedge** $e \in E$ *is a subset of vertices.*

Definition 6.3. *A subset $M \subseteq E(H)$ of hyperedges is a* **matching** *if every pair of hyperedges in M has an empty intersection.*

The following is the main problem we discuss over hypergraphs. Before handling hypergraphs, let us first review what are approximation algorithms. Sometimes, it is difficult to find a solution to an optimization problem efficiently (in polynomial time). One possible approach, is instead of finding an optimal solution to the problem, to find an approximate solution. In order to classify the approximation we use the concepts of "ρ-approximation algorithm" and "approximation ratio ρ". Here ρ denotes the ratio between the optimum solution and the approximate solution. We say that an algorith has approximation ratio ρ, or that it is a ρ-approximation algorithm if it runs in polynomial time and produces a solution of value at most ρ the value of an optimal solution in the case of a minimization problem, and of value at least $1/\rho$ the value of an optimal solution in the case of a maximization problem,

Problem 6.2. *Given a hypergraph, a weight w_e on each hyperedge e, the* **hypergraph matching problem** *is to find a matching with the maximum total weight.*

Note that the graph matching problem is a special case when every hyperedge has exactly two vertices.

Definition 6.4. *A hypergraph H is called* k**-uniform** *if every hyperedge has exactly k vertices.*

Definition 6.5. *A hypergraph H is called* k**-partite** *if H is k-uniform and the set of vertices can be partitioned into k disjoint sets $V_1, V_2, ..., V_k$ so that each hyperedge intersects every set of the partition in exactly one vertex.*

Note that a bipartite graph is a 2-partite hypergraph.

Theorem 6.1. *For the hypergraph matching problem, there is a polynomial time $(k-1+\frac{1}{k})$-approximation algorithm for k-uniform hypergraphs, and a $(k-1)$-approximation algorithm for k-partite hypergraphs.*

For $k = 3$, the problem in Theorem 6.1 is known as the 3-dimensional matching problem. It is a classic NP-complete problem. In this work we prove the theorem for the 3-partite hypergraphs.

6.2.1 Linear Programming Relaxation

In this section we use $\delta(v)$ to denote the set of hypergraphs that contains v.

The standard linear programming relaxation would be:

$$\begin{aligned} \textit{maximize} \quad & \sum_{e \in E} w_e x_e & \\ \textit{subject to} \quad & \sum_{e \in \delta(v)} x_e \leq 1 & \forall v \in V \\ & x_e \geq 0 & \forall e \in E \end{aligned}$$

As usual x_e is an indicator for edge e; w_e is the weight of edge e. The original linear program would require that at most one edge of all the edges that include the vertex v is chosen into the matching.

Let $\mathcal{B}$ denote the vector of all degree bounds $0 \leq B_v \leq 1$ for each vertex $v \in V$. Let us consider the following general linear problem denoted by $LP_M(H, \mathcal{B})$ (Initially $B_v = 1$ for each $v \in V$).

$$\begin{aligned}
&\textit{maximize} && \sum_{e \in E} w_e x_e && \\
&\textit{subject to} && \sum_{e \in \delta(v)} x_e \leq B_v && \forall v \in V \\
& && x_e \geq 0 && \forall e \in E
\end{aligned}$$

For the analysis of the iterative algorithm, we will use the $LP_M(H, \mathcal{B})$.

6.2.2 Characterization of Extreme Point Solutions

The lemma below follows by a direct application of the Rank Lemma.

Lemma 6.3. *Given any extreme point solution x to $LP_M(H, \mathcal{B})$ with $x_e > 0$ for each edge $e \in E$, there exists $W \subseteq V$ such that:*

1. $x(\delta(v)) = B_v > 0$ *for each* $v \in W$.
2. *The vectors in* $\{\chi(\delta(v)) : v \in W\}$ *are linearly independent.*
3. $|E| = |W|$.

6.2.3 Iterative Algorithm

The algorithm consists of two phases. In the first phase we use an iterative algorithm to provide a "good" ordering of the hyperedges. In the second phase we apply the *local ratio method* to this good ordering to obtain a matching with cost at most twice the optimum.

The *Local-Ratio* routine described below provides an efficient procedure to obtain a 2-approximate solution for the 3-dimensional matching problem.

Algorithm 6 Iterative 3-Dimensional Matching Algorithm

```
1: Find an optimal extreme point solution $x$ to $LP_M(H, \mathcal{B})$ with $B_v = 1$ for
   all $v$.
   $F \leftarrow \emptyset$
2: for $i$ from 1 to $|E(H)|$ do
3:     (a) Find a hyperedge $e$ with $x(N[e]) \leq 2$.
4:     (b) Set $f_i \leftarrow e$ and $F \leftarrow F \cup \{f_i\}$.
5:     (c) Remove $e$ from $H$.
6:     (d) Decrease $B_v$ by $x_e$ for all $v \in e$.
7: end for
8: $M \leftarrow$ Local-Ratio$(F, w)$, where $w$ is the weight vector of the hyper-
   edges.
9: Return $M$.
```

Algorithm 7 Local-Ratio(F, w)

```
1: Remove from $F$ all hyperedges with non-positive weights.
2: if $F = \emptyset$, then return $\emptyset$.
3: Choose from $F$ the hyperedge $e$ with the smallest index. Decompose the
   weight vector $w = w_1 + w_2$ where
       $$w_1(e') = \begin{cases} w(e) & if\ e' \in N[e], \\ 0 & otherwise. \end{cases}$$
4: $M' \leftarrow$ Local-Ratio$(F, w_2)$
5: If $M' \cup \{e\}$ is a matching, return $M' \cup \{e\}$; else return $M'$
```

6.2.4 Correctness and Optimality

We prove the following theorem:

Theorem 6.2. *After the loop (i.e. line 8) of the iterative algorithm, there is an ordering of the hyperedges such that:*
$x(N[e_i] \cap \{e_i, e_{i+1}, ..., e_m\}) \leq 2$ *for all* $1 \leq i \leq m$*, where* m *is the number of hyperedges in* x *with positive fractional value.*

In order to prove Theorem 6.2 we first need to prove the following Lemma:

Lemma 6.4. *Suppose* x *is an extreme point solution to* $LP_M(H, \mathcal{B})$*. If* $x_e > 0$ *for all* $e \in E$*, then there is a hyperedge* e *with* $x(N[e]) \leq 2$*.*

Proving this is done using Lemma 6.3. Let us assume that W is a set of tight vertices as described in Lemma 6.3. We start by proving that in any extreme point solution to $LP_M(H, \mathcal{B})$, there is a vertex in W with maximum degree of 2. By contradiction, let us assume that all vertices in W have a degree of 3 or above. Therefore:

$$|W| = |E| = \frac{\sum_{v \in V} |\delta(v)|}{3} \geq \frac{\sum_{v \in W} |\delta(v)|}{3} \geq |W|.$$

The result implies that all inequalities must be held as equalities. This implies that every hyperedge is contained in W.

Let V_1, V_2, V_3 be the tri-partition of V, and $W_i = W \cap V_i$ for $1 \leq i \leq 3$. Since each hyperedge intersects W_i exactly once we get:

$$\sum_{v \in W_1} \chi(\delta(v)) = \sum_{v \in W_2} \chi(\delta(v))$$

This implies that the characteristic vectors in W are not linearly independent, contradicting Lemma 6.3. Therefore there is a vertex $u \in W$ of degree at most two. Let $e = \{u, v, w\}$ be the hyperedge in $\delta(u)$ with larger weight. Since u is of degree at most two, this implies that $2x_e \geq x(\delta(u))$. And:

$$x(N[e]) \leq x(\delta(u)) + x(\delta(v)) + x(\delta(w)) - 2x_e \leq x(\delta(v)) + x(\delta(w)) \leq B_v + B_w \leq 2.$$

We have a hyperedge e with $x(N[e]) \leq 2$ in an extreme point solution to $LP_M(H, \mathcal{B})$. Now we need to show that the remaining solution in the algorithm (second part of the iterative loop - after removing e and updating B_v) is still an extreme point solution to $LP_M(H, \mathcal{B})$. We will do this by proving the following lemma:

Lemma 6.5. *In any iteration of the loop of the algorithm (lines 2-7) the restriction of the fractional solution is an extreme point solution to $LP_M(H, \mathcal{B})$.*

Proof. Suppose the graph in the current iteration is $H = (V, E)$. Let x_E be the restriction of the initial extreme point solution x to E. We prove by induction on the number of iterations that x_E is an extreme point solution to $LP_M(H, \mathcal{B})$. This holds in the first iteration by first line of the algorithm.

Let $e = \{v_1, v_2, v_3\}$ be the hyperedge found in the first line of the loop in the algorithm. Let $E' = E - e$ and $H' = (V, E')$. Let $\mathcal{B}'$ be the updated degree bound vector. We prove that $x_{E'}$ is an extreme point solution to $LP_M(H', \mathcal{B}')$. Since the degree bounds of v_1, v_2, v_3 are decreased by exactly x_e, it follows that $x_{E'}$ is still a feasible solution. Suppose to the contrary that $x_{E'}$ is not an extreme point solution to $LP_M(H', \mathcal{B}')$. This means that $x_{E'}$ can be written as a convex combination of two different feasible solutions y_1 and y_2 to $LP_M(H', \mathcal{B}')$. Extending y_1 and y_2 by setting the fractional value on e to be x_e, this implies that x_E can be written as a convex combination of two different feasible solutions to $LP_M(H, \mathcal{B})$, contradicting that x_E is an extreme point solution. Hence $x_{E'}$ is an extreme point solution to $LP_M(H', \mathcal{B}')$. □

Now that we also established that the solution remains an extreme point solution we can apply Lemma 6.4 inductively. The algorithm will succeed in finding the ordering we require.

We now wish to obtain an efficient approximation algorithm for the 3-dimensional matching problem. We introduce the Local Ratio Theorem from [14]:

Theorem 6.3 (***Local Ratio Theorem***)**.** *Let $\mathcal{C}$ be a set of vectors in $\mathbb{R}^n$. Let $w, w_1, w_2 \in \mathbb{R}^n$ be such that $w = w_1 + w_2$. Suppose $x \in \mathcal{C}$ is r-approximate with respect to w_1 and r-approximate with respect to w_2. Then x is r-approximate with respect to w.*

Proof. Let x^*, x_1^*, and x_2^* be optimal solutions with respect to w, w_1, and w_2, respectively. Clearly, $w_1 \cdot x_1^* \leq w_1 \cdot x^*$, and $w_2 \cdot x_2^* \leq w_2 \cdot x^*$. Thus $w \cdot x = w_1 \cdot x + w_2 \cdot x \leq r(w_1 \cdot x_1^*) + r(w_2 \cdot x_2^*) \leq r(w_1 \cdot x^*) + r(w_2 \cdot x^*)$. □

Now it remains to prove that the cost of the matching we received is at least half the optimum.

Theorem 6.4. *Let x be an optimal extreme point solution to $LP_M(H, \mathcal{B})$. The matching M returned by the algorithm satisfies $w(M) \geq \frac{1}{2} \cdot w \cdot x$*

Proof. We prove this using induction on the number of hyperedges having positive weights. If the number of hyperedges with positive edges is 0 the case is trivial. Let e be the hyperedge chosen by the Local-Ratio algorithm. e has the smallest index in the ordering. So by Theorem 6.2, we get $x(N[e]) \leq 2$. Let w, w_1, w_2 be the weight vectors from the algorithm. Let y and y' be the characteristic vectors for M and M' from the algorithm. Since $w(e) > 0$ and $w_2(e) = 0$, w_2 has fewer hyperedges with positive weights than w. By the induction hypothesis, $w_2 \cdot y' \geq \frac{1}{2} \cdot w_2 \cdot x$. Since $w_2(e) = 0$, this implies that $w_2 \cdot y \geq \frac{1}{2} \cdot w_2 \cdot x$. By the last step of the algorithm, at least one hyperedge in $N[e]$ is in M. Since $x(N[e]) \leq 2$ and $w_1(e') = w(e)$ for all $e' \in N[e]$, it follows that $w_1 \cdot y \geq \frac{1}{2} \cdot w_1 \cdot x$. Therefore, by Theorem 6.3, we have $w \cdot y \geq \frac{1}{2} \cdot w \cdot x$. This shows that M is a 2-approximate solution to the 3-dimensional matching problem. $\square$

7 Network Design

In this chapter we study the survivable network design problem (SNDP). This chapter is based on chapter 10 of [1]. In addition this chapter analyzes the degree bounded SNDP as described in [2].

Definition 7.1. *Given an undirected graph $G = (V, E)$ and a connectivity requirement r_{uv} for each pair of vertices u, v, a* Steiner network *is a subgraph of G in which there are at least r_{uv} edge-disjoint paths between u and v for every pair of vertices u, v.*

7.1 SNDP

Problem 7.1. *The* Survivable Network Design Problem *is to find a Steiner network with minimum total cost.*

Note that the survivable network design problem generalizes the minimum Steiner tree problem, the minimum Steiner forest problem, and the minimum k-edge-connected subgraph problem.

In the first part of this section we introduce the 2-approximation algorithm to the problem. This was first introduced by Jain [15]. Then we show the connection to the traveling salesman problem and finally we consider the minimum bounded degree Steiner network problem.

7.1.1 Linear Programming Relaxation

First we define skew supermodular set functions.

Definition 7.2. *A function $f\colon 2^v \to \mathbb{Z}$ is called* skew supermodular *if at least one of the following two conditions holds for any two subsets $S, T \subseteq V$:*

$$f(S) + f(T) \le f(S \cup T) + f(S \cap T)$$
$$f(S) + f(T) \le f(S \setminus T) + f(T \setminus S)$$

It is not hard to show that the set function f defined by $f(S) = \max_{u \in S, v \notin S}\{r_{uv}\}$ for each $S \subseteq V$ is a skew supermodular function.

The linear program LP_{sndp}:

$$\begin{aligned} \textit{minimize} \quad & \sum_{e \in E} c_e x_e \\ \textit{subject to} \quad & x(\delta(S)) \geq f(S) \quad \forall S \subseteq V \\ & 0 \leq x_e \leq 1 \quad \forall e \in E \end{aligned}$$

It is not known whether there is a polynomial time separation oracle for a general skew supermodular function f. This linear program for the minimum Steiner network problem, however, can be solved in polynomial time by using a maximum flow algorithm as a separation oracle.

7.1.2 Characterization of Extreme Point Solutions

Similar to previous sections (spanning trees) the extreme point solution to LP_{sndp} is characterized by a laminar family of tight constraints. The following Lemma follows from the uncrossing arguments and the Rank Lemma.

Lemma 7.1. *Let the requirement function f of LP_{sndp} be skew supermodular, and let x be an extreme point solution to LP_{sndp} with $0 < x_e < 1$ for every edge $e \in E$. Then, there exists a laminar family $\mathcal{L}$ such that:*

1. *$x(\delta(S)) = f(S)$ for each $S \in \mathcal{L}$.*
2. *The vectors in $\{\chi(\delta(S)) : S \in \mathcal{L}\}$ are linearly independent.*
3. *$|E| = |\mathcal{L}|$.*

7.1.3 The Iterative Algorithm

The following algorithm is due to Jain [15]:

Algorithm 8 Iterative Minimum Steiner Network Algorithm

```
1: $F \leftarrow \emptyset$, $f' \leftarrow f$;
2: while $f' \not\equiv 0$ do
3:     (a) Find an optimal extreme point solution $x$ to $LP_{sndp}$ with cut requirements $f'$ and remove every edge $e$ with $x_e = 0$.
4:     (b) If there is an edge $e$ with $x_e \geq 1/2$, then add $e$ to $F$ and delete $e$ from the graph.
5:     (c) For every $S \subseteq V$: update $f'(S) \leftarrow \max\{f(S) - d_F(S), 0\}$.
6: end while
7: Return $H = (V, F)$.
```

7.1.4 Correctness and Performance Guarantee

Jain's proof of the algorithm uses a token counting argument.

Theorem 7.1. *Suppose f is an integral skew supermodular function and x is an extreme point solution to LP_{sndp}. Then there exists an edge $e \in E$ with $x_e \geq \frac{1}{2}$.*

It is obvious that if Theorem 7.1 is true then the iterative algorithm terminates (since step (b) in the loop is fulfilled). Now we wish to prove Theorem 7.1. This proof uses the fractional token idea as can be seen in [3].

Proof. We start by assuming in contradiction that $0 < x_e < 1/2$ for each $e \in E$. We will show that $|E| > |\mathcal{L}|$ as contradiction to Lemma 7.1. The fractional token method is used as follows: Assign each edge $e = \{u, v\}$ one token. Now, reassign the tokens as follows:

1. *Rule 1* Let $S \in \mathcal{L}$ be the smallest set containing u and let $R \in \mathcal{L}$ be the smallest set containing v. Let e distribute x_e (which is smaller than 1/2 by assumption) token to S and R.

2. *Rule 2* Let e distribute the remaining $1 - 2x_e$ tokens to the smallest set containing both u and v (denote this set as $T \in \mathcal{L}$).

Now choose any set $S \in \mathcal{L}$. This set has k children ($k \geq 0$) denoted by

$R_1, R_2, ...R_k$. Then:

$$\begin{aligned} x(\delta(S)) &= f(S) \\ x(\delta(R_i)) &= f(R_i) \quad \forall 1 \leq i \leq k \end{aligned}$$

If we subtract the equations we get:

$$x(\delta(S)) - \sum_i x(\delta(R_i)) = f(S) - \sum_{i=1}^{k} f(R_i)$$

We define 3 groups of edges:

$$\begin{aligned} A &= \{e : |e \cap (\cup_i R_i)| = 0, |e \cap S| = 1\} \\ B &= \{e : |e \cap (\cup_i R_i)| = 1, |e \cap S| = 2\} \\ C &= \{e : |e \cap (\cup_i R_i)| = 2, |e \cap S| = 2\} \end{aligned}$$

Edges in A have only one vertex in S but no vertices in the children. Edges in B and C have both endnodes in S but for an edge in B only one side of the edge is in some child of S.

According to Rule 1, for each edge $e \in A$, S receives x_e tokens. For any edge in B, both rules apply; Rule 1 for the edge that is also shared with the children of S, and Rule 2 as trivially seen. So S received $1 - x_e$ tokens. For any edge in C, S receives $1 - 2x_e$ tokens by Rule 2.

We rewrite the previous equation and get:

$$x(A) - x(B) - 2x(C) = f(S) - \sum_{i=1}^{k} f(R_i)$$

According to the token redistribution:

$$\begin{aligned} 0 &< \sum_{e\in A} x_e + \sum_{e\in B}(1-x_e) + \sum_{e\in C}(1-2x_e) \\ &= x(A) + |B| - x(B) + |C| - 2x(C) \\ &= |B| + |C| + f(C) - \sum_{i=1}^{k} f(R_i) \end{aligned}$$

Since the function f is integral, the sum on the last sum (right side of equation) must be at least 1. This means that S receives at least one token.

Now, choose a set R to be the maximal set in $\mathcal{L}$. Choose any edge $e \in \delta(R)$. We want to see why Rule 2 cannot be applied. There is no set T with $|T \cap e| = 2$ because we already chose the maximal set. So the $1 - 2x_e$ tokens cannot be assigned according to the rule.

Since every set in $\mathcal{L}$ receives at least one token but there are still unassigned tokens we get the contradiction $|E| > |\mathcal{L}|$. $\square$

Now we can see why the algorithm is a 2-approximation algorithm. This is done using a simple induction. The base case is trivial. For the induction step, let's look at the solution H. We can write the cost as: $cost(H) = cost(H') + c_{e'}$ where e' is the edge with $x_{e'} \geq 1/2$ as found in Theorem 7.1. Using the induction step we know that $cost(H') \leq 2 \sum_{e\in E-e'} c_e x_e$. Therefore we get that $cost(H) \leq 2 \sum_{e\in E} c_e x_e$ (because $x_{e'} \geq 1/2$).

7.1.5 Connection to the Traveling Salesman Problem

First, we recall the traveling salesman problem (TSP):

Problem 7.2. *Given an undirected graph $G = (V, E)$ and a cost function $c : E \to \mathbb{R}_+$, the Traveling Salesman Problem (TSP) is to find a minimum cost Hamiltonian cycle.*

In this section we show a generalization of the survivable network design problem and the traveling salesman problem. We claim that any extreme point solution of this generalization has an edge e with $x_e \geq \frac{1}{2}$, and in some cases $x_e = 1$. In this section we do not show the proofs, we only generally describe the concept.

7.1.6 Linear Programming Relaxation

The following linear program relaxation models both the survival network design problem and the traveling salesman problem. It will be denoted by LP_f where f is a skew supermodular function.

$$\begin{aligned} \textit{minimize} \quad & \sum_{e \in E} c_e x_e \\ \textit{subject to} \quad & x(\delta(S)) \geq f(S) \quad \forall S \subseteq V \\ & x(\delta(v)) = f(v) \quad \forall v \in W \\ & 0 \leq x_e \leq 1 \quad \forall e \in E \end{aligned}$$

For the survivable network design problem we set $f(S) = \max_{u \in S, v \notin S}\{r_{uv}\}$ for each subset $S \subseteq V$ and set $W = \emptyset$. For the traveling salesman problem we set $f(S) = 2$ for each $S \subset V$, $f(V) = 0$ and $W = V$.

7.1.7 Characterization of Extreme Point Solutions

The following is very similar to lemmas we have already seen:

Lemma 7.2. *Let the requirement function f of LP_f be skew supermodular, and let x be an extreme point solution to LP_f with $0 < x_e < 1$ for every $e \in E$. Then, there exists a laminar family $\mathcal{L}$ such that:*

1. *$x(\delta(S)) = f(S)$ for each $S \in \mathcal{L}$*
2. *The vectors in $\chi(\delta(S)) : S \in \mathcal{L}$ are linearly independent*
3. *$|E| = |\mathcal{L}|$*

7.1.8 Existence of Edges with Large Fractional Value

There is an edge e with $x_e = 1$ in any extreme point solution of the traveling salesman problem (this was proven by Boyd and Pulleyblank [19]). The following theorem generalizes their result as well as Jain's (Theorem 7.1)

Theorem 7.2. *Let f be an integral skew supermodular function, and x be an extreme point solution to LP_f with $x_e > 0$ for all e. Then, there exists an $e \in E$ with $x_e = \frac{1}{2}$. Moreover, if $f(S)$ is an even integer for each subset $S \subseteq V$ then there exists an edge e with $x_e = 1$.*

7.2 Degree-bounded SNDP

Problem 7.3. *The* degree-bounded SNDP *problem is similar to the SNDP problem with a degree bound $b(v)$ on the vertices; namely, each vertex v is incident to at most $b(v)$ edges in the solution graph H.*

In addition we define the following problem:

Problem 7.4. *The* element connectivity SNDP *(Elem-SNDP) problem is similar to the SNDP problem but with two sest of vertices R,W. The vertices in R are called* reliable vertices *or* terminals, *while the vertices in W are called unreliable vertices. The vertices in W and the edges are called* elements. *The goal of the Elem-SNDP problem is to satisfy the connectivity requirement between each pair u, v of terminals, while the paths are element-disjoint (i.e. paths do not share unreliable nodes or edges).*

In this section, we present a *bicriteria* approximation to the problem based on [2].

Theorem 7.3. *There is a polynomial time approximation algorithm for the degree bounded Elem-SNDP problem in undirected graphs that achieves an $(O(1), O(1)b(v))$ bicriteria approximation.*

We will get back to the problems later. First, we need to establish some preliminary statements.

7.2.1 Preliminaries

We now introduce a number of definitions. Let V be a ground set.

Definition 7.3. *A* biset $\mathbb{A} = (A, A')$ *is a pair of sets such that $A \subseteq A' \subseteq V$. A is the* inner *part of $\mathbb{A}$ and A' is the* outer *part of $\mathbb{A}$. $bd(\mathbb{A}) = A' - A$ is the* boundary *of $\mathbb{A}$.*

In addition we need the following definition.

Definition 7.4. *intersection/union/difference of two bisets:*

1. $\mathbb{A} \cap \mathbb{B} = (A \cap B, A' \cap B')$

2. $\mathbb{A} \cup \mathbb{B} = (A \cup B, A' \cup B')$

3. $\mathbb{A} - \mathbb{B} = (A - B', A' - B)$

Also $\mathbb{A}$ *is contained in* $\mathbb{B}$*:* $\mathbb{A} \subseteq \mathbb{B}$ *if and only if* $A \subseteq B$ *and* $A' \subseteq B'$

Now we define disjoint/intersecting bisets:

Definition 7.5. *Two bisets* $\mathbb{A}$ *and* $\mathbb{B}$ *are* disjoint *if* $A \cap B$ *is empty.*

Definition 7.6. *Two bisets* $\mathbb{A}$ *and* $\mathbb{B}$ *are* intersecting *if* $\mathbb{A}$ *and* $\mathbb{B}$ *are not disjoint.*

Definition 7.7. *Two bisets* $\mathbb{A}$ *and* $\mathbb{B}$ *are* strongly disjoint *if* $A' \cap B$ *and* $A \cap B'$ *are empty.*

Definition 7.8. *Two bisets* $\mathbb{A}$ *and* $\mathbb{B}$ *are* overlapping *if* $\mathbb{A}$ *and* $\mathbb{B}$ *are not strongly disjoint.*

Definition 7.9. *A family of bisets is* bilaminar *if, for any two bisets* $\mathbb{A}$ *and* $\mathbb{B}$ *in the family, one of the following holds:* $\mathbb{A} \subseteq \mathbb{B}$*,* $\mathbb{B} \subseteq \mathbb{A}$*, or* $\mathbb{A}$ *and* $\mathbb{B}$ *are disjoint.*

Definition 7.10. *A family of bisets is* strongly bilaminar *if , for any two bisets* $\mathbb{A}$ *and* $\mathbb{B}$ *in the family, one of the following holds:* $\mathbb{A} \subseteq \mathbb{B}$*,* $\mathbb{B} \subseteq \mathbb{A}$*, or* $\mathbb{A}$ *and* $\mathbb{B}$ *are strongly disjoint.*

Definition 7.11. *Let* f *be defined as* $f : 2^V \times 2^V \to \mathbb{Z}$*.* f *is* bisupermodular *if for any two bisets* $\mathbb{A}$ *and* $\mathbb{B}$*:*

$$f(\mathbb{A}) + f(\mathbb{B}) \leq f(\mathbb{A} \cap \mathbb{B}) + f(\mathbb{A} \cup \mathbb{B})$$

f *is* intersecting *bisupermodular if the inequality above holds for any two bisets* $\mathbb{A}$ *and* $\mathbb{B}$ *that intersect.*

f is positively *bisupermodular if the inequality above holds for any two bisets $\mathbb{A}$ and $\mathbb{B}$ such that $f(\mathbb{A}) > 0$ and $f(\mathbb{B}) > 0$.*

f is bisubmodular *if $-f$ is bisupermodular.*

f is binegamodular *if for any two bisets $\mathbb{A}$ and $\mathbb{B}$:*

$$f(\mathbb{A}) + f(\mathbb{B}) \leq f(\mathbb{A} - \mathbb{B}) + f(\mathbb{B} - \mathbb{A})$$

f is biposimodular *if $-f$ is binegamodular.*

f is skew bisupermodular *if for any two bisets $\mathbb{A}$ and $\mathbb{B}$:*

$$f(\mathbb{A}) + f(\mathbb{B}) \leq \max\{f(\mathbb{A} \cap \mathbb{B}) + f(\mathbb{A} \cup \mathbb{B}), f(\mathbb{A} - \mathbb{B}) + f(\mathbb{B} - \mathbb{A})\}$$

f is positively *skew bisupermodular if the inequality above holds for any two bisets $\mathbb{A}$ and $\mathbb{B}$ such that $f(\mathbb{A}) > 0$ and $f(\mathbb{B}) > 0$.*

Definition 7.12. *Let F denote a set of undirected edges. $\delta_F(\mathbb{A})$ defines the set of all edges $e \in F$ such that e has one endpoint in A and another endpoint in $V - A'$. $\chi(\delta_F(\mathbb{A}))$ is defined as the characteristic vector of $\delta_F(\mathbb{A})$.*

Definition 7.13. *Let G denote a set of directed edges. $\delta^-_G(\mathbb{A})$ defines the set of all edges $e \in G$ with head in A and tail in $V - A'$. $\delta^+_G(\mathbb{A})$ defines the set of all edges $e \in G$ with head in $V - A'$ and tail in A .*

The following lemmas are brought here without proof.

Lemma 7.3. *For any set F of edges and any positive weight function w on F, the function $f(\mathbb{A}) = w(\delta_F(\mathbb{A}))$ is both bisubmodular and biposimodular.*

Lemma 7.4. *For any set F of directed edges and any positive weight function w on F, the function $f(\mathbb{A}) = w(\delta^-_F(\mathbb{A}))$ is bisubmodular.*

Lemma 7.5. *For any two bisets $\mathbb{A}$ and $\mathbb{A}$, we have:*

$$|bd(\mathbb{A})| + |bd(\mathbb{B})| = |bd(\mathbb{A} \cap \mathbb{B})| + |bd(\mathbb{A} \cup \mathbb{B})|$$

$$|bd(\mathbb{A})| + |bd(\mathbb{B})| = |bd(\mathbb{A} - \mathbb{B})| + |bd(\mathbb{B} - \mathbb{A})| + 2|bd(\mathbb{A}) \cap bd(\mathbb{B})|$$

Before going to the problem definitions we define covering.

Definition 7.14. *Let $G = (V, E)$ be an undirected graph and let $f : 2^V \times 2^V \to \mathbb{R}$ be a biset function. G* covers *f if $|\delta_G(\mathbb{A})| \geq f(\mathbb{A})$ for each $\mathbb{A}$. For directed graph we need the inequality $|\delta^-_G(\mathbb{A})| \geq f(\mathbb{A})$*

Now we define the problems that we consider. Let $G = (V, E)$ be an undirected graph with weights $w(e)$ on edges and degree bounds $b(v)$ on the nodes.

Problem 7.5. ***Degree-bounded Element-SNDP:*** *The vertices of V are partitioned into two sets, the set R of reliable vertices and the set W of unreliable vertices. The vertices of W and the edges are called* elements. *The element-connectivity of a pair u, v of nodes is the maximum number of element-disjoint paths from u to v; these paths do not share any unreliable nodes or edges, but they may share reliable nodes. Two vertices u and v are l-element-connected if their element connectivity is at least l. In addition to the weights and bounds, we add integer requirement $r(u, v)$ for $u, v \in R$. The goal of the* Degree-bounded Elem-SNDP *is to select a minimum weight subgraph H of G such that each pair u, v of terminals is $r(u, v)$-element-connected in H and $|\delta_H(v)| \leq b(v)$ for each v.*

Problem 7.6. ***Degree-bounded VC-SNDP:*** *The vertex-connectivity of a pair u, v of vertices is the maximum number of paths between u and v that are internnally vertex disjoint. Two vertices u and v are l-vertex-connected if their vertex connectivity is at least l. In the* Degree-bounded VC-SNDP *problem, in addition to the graph G, we are given integer requirements $r(u, v)$ for each pair u, v of vertices. The goal is to select a minimum-weight subgraph H of G such that each pair u, v of vertices is $r(u, v)$-vertex-connected in H and $|\delta_H(v)| \leq b(v)$ for each vertex v.*

Problem 7.7. ***Degree-bounded k-Connected Subgraph:*** *A graph H is k-vertex-connected if each pair u, v of vertices is k vertex connected in H. In the* Degree-bounded k-Connected Subgraph *problem, the goal is to select a minimum-weight spanning subgraph $H = (V, E')$ of G such that H is k-vertex connected and $|\delta_H(v)| \leq b(v)$ for each vertex v.*

Problem 7.8. ***Degree-bounded Rooted k-Connectivity:*** *A vertex r is k-vertex-connected in H to each other vertex if, for any vertex $v \in V - \{r\}$, the pair r, v is k vertex connected in H. In the* Degree-bounded Rooted k-Connectivity *problem, we are given a root vertex r and the goal is to select a minimum-weight subgraph H of G such that the root r is k-vertex-connected in H to every other vertex and $|\delta_H(v)| \leq b(v)$ for each vertex v.*

Problem 7.9. ***Degree-bounded Residual Cover:*** *In the* Degree-bounded Residual Cover *problem, we are given a function $f \colon 2^V \times 2^V \to \mathbb{Z}$ satisfying $f(\mathbb{A}) = r(\mathbb{A}) - |bd(\mathbb{A})| - |\delta_F(\mathbb{A})|$ for each biset $\mathbb{A}$, where r is a biset function and $F \subseteq E$ is a set of edges, and the goal is to select a minimum weight set $F' \subseteq E - F$ of edges such that $|\delta_{F'}(\mathbb{A})| \geq f(\mathbb{A})$ for each biset $\mathbb{A}$ and $|\delta_{F'}(v)| \leq b(v)$ for each vertex v.*

We describe approximation algorithms for the problems defined. We start with the Degree-bounded Residual Cover problem that has an $(O(1), O(1)b(v))$ approximation if function r satisfies a certain condition:

Theorem 7.4. *Consider an instance of the* Degree-bounded Residual Cover *problem in which the function f satisfies $f(\mathbb{A}) = r(\mathbb{A}) - |bd(\mathbb{A})| - |\delta_F(\mathbb{A})|$, where r is an integer-valued biset function and $F \subseteq E$ is a set of edges. Let* OPT *be the weight of an optimal solution for the instance. Suppose that r and f satisfy the following conditions:*

1. *For each biset (A, A') and each vertex $v \in A' - A$, we have $r((A, A')) \leq r((A, A' - v))$.*
2. *The function f is positively skew bisupermodular.*

Then there is a polynomial time iterated rounding algorithm that selects a set $F' \subseteq E - F$ of edges such that $w(F') \leq 3OPT$ and $|\delta_{F'}(v)| \leq |\delta_F(v)| + 6b(v) + 5$ for each vertex v.

Let us assume that Theorem 7.4 is correct. Choose $F = \emptyset$, $f = f_{elt}$ and $r = r_{elt}$, where f_{elt} and r_{elt} are defined as follows.

Let $r_{elt} \colon 2^V \times 2^V \to \mathbb{Z}_+$ be a biset function such that $r_{elt}(\mathbb{A}) = max_{u \in A, v \in V - A'} r(uv)$ if $bd(\mathbb{A}) \subseteq W$ and $r_{elt}(\mathbb{A}) = 0$ otherwise.

Let $f_{elt}\colon 2^V \times 2^V \to \mathbb{Z}$ be a biset function such that $f_{elt}(\mathbb{A}) = r_{elt}(\mathbb{A}) - |bd(\mathbb{A})|$.

It is easy to see that the r we chose satisfies the first condition (removing one vertex from A' only increases the choices of $r(uv)$). We need to show that f is positively skew bisupermodular. This was already proven by Fleisher el al (see [24]). This clearly brings us a $(O(1), O(1)b(v))$ approximation for the *Elem-SNDP* problem.

Now, we wish to prove Theorem 7.4. We prove the theorem by giving an iterative algorithm that fulfills the theorem requirements. First, we present the linear program relaxation of the problem LP_{Undir}:

$$\begin{array}{lll}
\textit{minimize} & \sum_{e\in E} w(e)x(e) & \\
\textit{subjectto} & x(\delta_E(\mathbb{A})) \geq f(\mathbb{A}) & \forall \mathbb{A}\colon f(\mathbb{A}) > 0 \\
 & x(\delta_E(v)) \leq b(v) & \forall v \in X \\
 & 0 \leq x(e) \leq 1 & \forall e \in E
\end{array}$$

Where f is an integer-valued biset function and the set $X \subseteq V$ is a subset of the vertices that have degree bounds and $b\colon X \to \mathbb{R}$ are the degree bounds on the vertices in X.

Where G, r, F, X and b are the Degree-bounded Residual Cover parameters.

Now we need to prove that this algorithm terminates and provides a feasible cover of f. In order to prove this, we first have to consider the following theorem:

Theorem 7.5. *Consider an iteration of* Undir-algo*. Let $G' = (V, E')$ be the residual subgraph at the beginning of this iteration. Let F' be the set of edges selected in the previous iterations. Let X' be the set of vertices that have degree bounds, and let $b'\colon X' \to \mathbb{R}$ be the degree bounds on X' at the beginning of this iteration. Let $f'\colon 2^V \times 2^V \to \mathbb{Z}$ be the function satisfying $f'(\mathbb{A}) = r(\mathbb{A}) - |A' - A| - |\delta_{F\cup F'}(\mathbb{A})|$ for each biset $\mathbb{A}$. If x is a basic solution to LP_{Undir} for the input (G', f', X', b'), one of the following holds:*

1. *There is an edge $e \in E'$ such that $x(e) = 0$*

Algorithm 9 Undir-Algo - Algorithm for the Degree-bounded Residual Cover

```
1: $E' \leftarrow E - F$;
2: $F' \leftarrow \emptyset$;
3: $X' \leftarrow X$;
4: $b'(v) \leftarrow b(v)$ for each $v \in X$;
5: **while** $E'$ is non-empty **do**
6:     Let $f' : 2^V \times 2^V \to \mathbb{Z}$ be defined as $f'(\mathbb{A}) = r(\mathbb{A}) - |A' - A| - |\delta_{F \cup F'}(\mathbb{A})|$ for each biset $\mathbb{A}$.
7:     Compute an optimal basic solution to $LP_{Undir}$ for the input $(G' = (V, E'), f', X', b')$.
8:     If there is an edge $e \in E'$ such that $x(e) = 0$ then $E' \leftarrow E' - e$
9:     If there is an edge $e = uv \in E'$ such that $x(e) \geq 1/3$ then
10:        $F' \leftarrow F' \cup e$.
11:        $E' \leftarrow E' - e$.
12:        If $u \in X'$ then $b'(u) \leftarrow b'(u) - x(e)$
13:        If $v \in X'$ then $b'(v) \leftarrow b'(v) - x(e)$
14:    Else
15:        Let $v \in X'$ be a vertex such that $|\delta_{E'}(v)| \leq |\delta_F(v)| + 3b(v) + 5$
16:        $X' \leftarrow X' - v$.
17: **end while**
18: Return $F'$.
```

2. *There is an edge $e \in E'$ such that $x(e) \geq 1/3$*

3. *There is a vertex $v \in X'$ such that $|\delta_{E'}(v)| \leq |\delta_F(v)| + 3b(v) + 5$*

The proof of Theorem 7.5 is very long and complex and therefore beyond the scope of this work. The proof can be found in [2]. Assuming that theorem is correct, it is easy to see that the algorithm terminates. In addition, since only edges that have $x(e) \geq 1/3$ are part of the solution, the weight of the solution is at most $3OPT$. Now we want to prove the upper bound of the degree of each vertex in the solution.

Lemma 7.6. *Consider an iteration of* Undir-algo. *Let F' be the set of edges selected in the previous iterations, let X' be the set of vertices that have degree bounds, and let $b' \colon X' \to \mathbb{R}$ be the degree bounds on X' at the beginning of the iteration. For each vertex $v \in X'$, we have $|\delta_{F'}(v)| \leq 3(b(v) - b'(v))$, where $b(v)$ is the initial degree bound on v.*

The proof of the lemma as it can be found in [2]. Combining the lemma with Theorem 7.5 we get:

Theorem 7.6. *Let F' be the solution constructed by* Undir-algo. *The set F' satisfies the following:*

1. *$|\delta_{F'}(\mathbb{A})| \geq f(\mathbb{A})$ for each biset A.*

2. *The total weight of F' is at most $3OPT$.*

3. *For each vertex v we have $|\delta_{F'}(v)| \leq |\delta_F(v)| + 6b(v) + 5$*

Theorem 7.6 directly leads to Theorem 7.4. Now we get our desired proof.

Now let us return to the result of Theorem 7.4. We already saw that it implies an $(O(1), O(1)b(v))$ approximation for the *Elem-SNDP*. From this we get the following result:

Theorem 7.7. *There is a polynomial time $(3, 3b(v) + 5)$ approximation for the* Degree-bounded Elem-SNDP *problem in undirected graphs.*

7.2.2 Further Analysis of the SNDP Problems

Now let us consider problem on directed graphs, so let $G = (V, E)$ be an undirected graph with weights $w(e)$ on edges, in-degree bound $b^-(v)$ and out-degree bounds $b^+(v)$ on the vertices.

Problem 7.10. ***Degree-bounded Rooted k-Outconnectivity:*** *In the* Degree-bounded Rooted k-Outconnectivity *problem, we are given a root vertex r and the goal is to select a minimum-weight subgraph H of G such that, for each vertex $v \in V - r$, there are k internally vertex disjoint paths in H from r to v and, for each vertex $v \in V$, $|\delta^-_H(v)| \leq b^-(v)$ and $|\delta^+_H(v)| \leq b^+(v)$.*

Let us consider the *Directed Degree-bounded Residual Cover* problem:

Problem 7.11. *Given a function $f\colon 2^V \times 2^V \to \mathbb{Z}$ satisfying $f(\mathbb{A}) = r(\mathbb{A}) - |bd(\mathbb{A})| - |\delta^-_F(\mathbb{A})|$ for each biset $\mathbb{A}$, where r is a biset function and $F \subseteq E$ is a set of edges. The goal of the problem is to select a minimum weight set $F' \subseteq E - F$ of edges such that $|\delta^-_{F'}(\mathbb{A})| \geq f(\mathbb{A})$ for each biset $\mathbb{A}$, and $|\delta^-_{F'}(v)| \leq b^-(v)$ and $|\delta^+_{F'}(v)| \leq b^+(v)$ for each vertex v*

In order to get an approximation for the problem we use the following theorem:

Theorem 7.8. *Consider an instance of the* Directed Degree-bounded Residual Cover *problem in which the function f satisfies $f(\mathbb{A}) = r(\mathbb{A}) - |bd(\mathbb{A})| - |\delta^-_F(\mathbb{A})|$, where r is an integer-valued biset function and $F \subseteq E$ is a set of edges. Let* OPT *be the weight of an optimal solution for the instance. Suppose that r and f satisfy the following conditions:*

1. *For each biset (A, A') and each vertex $v \in A' - A$, we have $r((A, A')) \leq r((A, A' - v))$.*

2. *The function f is positively skew bisupermodular.*

Then there is a polynomial time iterated rounding algorithm that selects a set $F' \subseteq E - F$ of edges such that $w(F') \leq 3OPT$, $|\delta^-_{F'}(v)| \leq 3b^-(v) + 5$ and $|\delta^+_{F'}(v)| \leq |\delta^+_F(v)| + 6b^+(v) + 3$ for each vertex v.

The proof for the theorem is similar to the proof of Theorem 7.4. Naturally, we see that the *Directed Degree-bounded residual Cover* achieves an $(O(1), O(1)b^-(v), O(1)b^+(v))$ approximation.

Let us examine Theorem 7.8 over the *Rooted k-Outconnectivity* problem (Problem 7.10). Let r be defined as $r = r_{rc}$ where $r_{rc}\colon 2^V \times 2^V \to \mathbb{Z}_+$ is a biset function such that $r_{rc}(\mathbb{A}) = k$ if $A \neq \emptyset$ and $r \in V - A'$, and $r_{rc}(\mathbb{A}) = 0$ otherwise. Let f be defined as $f_{rc}\colon 2^V \times 2^V \to \mathbb{Z}$ is a biset function such that $f_{rc}(\mathbb{A}) = r_{rc}(\mathbb{A}) - |bd(\mathbb{A})|$. r_{rc} fulfills the first requirement while f_{rc} is a positively intersecting bisupermodular (see [25]). As a corollary from Theorem 7.8 we get the following:

Corollary 7.1. *There is a polynomial time* $(3, 3b^-(v) + 5, 6b^+(v) + 3)$ *approximation for the* Degree bounded Rooted k-Outconnectivity *problem in directed graphs.*

Now that we have a solution for the *Degree bounded Rooted k-Outconnectivity* problem, we can easily find a solution to the *Degree bounded Rooted k-Connectivity* problem. We will do that by finding a simple reduction from the *Degree bounded Rooted k-Connectivity* problem to the *Degree bounded Rooted k-Outconnectivity* problem. First step of the reduction is changing the undirected graph into a directed graph. This is easily done by changing each undirected edge into two directed edges. The same applies to the bound requirements (each in and out requirements are equal to the bound requirement of the original edge). The weights are also applied in similar way. Now, it is possible to see that using the previous corollary we have an approximation for the new problem. This results in a solution with weight twice as big as the weight of the *Degree bounded Rooted k-Connectivity* problem. So we get:

Theorem 7.9. *There is a polynomial time* $(9, 9b(v) + 8)$ *approximation for the* Degree bounded Rooted k-Connectivity *problem in undirected graphs.*

Our next step is to explore the *Degree Bounded k-Connected Subgraph* problem. In this part we don't provide the entire process that results in the approximation. Instead, we touch the key elements of the proof.

First, let us define r_{sg} and f_{sg}: Let $r_{sg}\colon 2^V \times 2^V \to \mathbb{Z}_+$ be a biset function such that $r_{sg}(\mathbb{A}) = k$ if $A \neq \emptyset$ and $A' \neq V$, and $r_{sg}(\mathbb{A}) = 0$ otherwise.

Let $f_{sg}\colon 2^V \times 2^V \to \mathbb{Z}$ be a biset function such that $f_{sg}(\mathbb{A}) = r_{sg}(\mathbb{A}) - |bd(\mathbb{A})|$.

Consider an instance of the problem in which the number of nodes is at least k^3. Let F^* be an optimal solution for the problem and let OPT be the weight of the solution. Our first step is to pick an arbitrary set R_1 with k vertices. It is possible to create an instance of the *Rooted k-Outconnectivity* problem using the given instance. When we consider this new instance, an algorithm by Frank and Tardos [26] and Corollary 7.1 we can find a solution F_1 for the *Rooted k-Outconnectivity.* Let F_1' be the set of undirected edges corresponding to F_1. We get that $w(F_1') \leq w(F_1) \leq 6OPT$ and $\delta_{F_1'}(v) \leq \delta^-_{F_1}(v) + \delta^+_{F_1}(v) \leq 9b(v)+8$. The second step is similar to the first with one difference. We choose our new set R_2 using an approach developed by Cheriyan and Vegh [27]. Again, we define an instantiation of the *Rooted k-Outconnectivity* problem. Using the Corollary 7.1 we again construct a solution F_2. Let F_2' be the set of undirected edges corresponding to F_2. Once again we get $w(F_2') \leq w(F_2) \leq 6OPT$ and $\delta_{F_2'}(v) \leq \delta^-_{F_2}(v) + \delta^+_{F_1}(v) \leq 9b(v) + 8$. Now, let us look at the function $g(\mathbb{A}) = r_{sg}(\mathbb{A}) - |bd(\mathbb{A})| - |\delta_{F_1' \cup F_2'}(\mathbb{A})|$. The set $F^* - (F_1' \cup F_2')$ covers g. The Cheriyan and Vegh approach makes sure that we choose R_2 in a way that g is positively skew bisupermodular. It is easy to see that for each biset (A, A') and each vertex $v \in A' - A$, $r_{sg}((A, A')) \leq r_{sg}((A, A' - \{v\}))$. So now we get that r_{sg} and g satisfy the two conditions of Theorem 7.4. We choose $F = F_1' \cup F_2'$, $f = g$ and $r = r_{sg}$ and we get a cover F_3' of g from Theorem 7.4. We can see that the weight of F_3' is at most $3OPT$ and $|\delta_{F_3'}(v)| \leq |\delta_{F_1' \cup F_2'}(v)| + 3b(v) + 5 \leq 2(9b(v)+8) + 6b(v) + 5$. The final solution to the k-Connected problem is $F = F_1' \cup F_2' \cup F_3'$. From all the above it is possible to prove that:

Theorem 7.10. *There is a polynomial time (15, 42b(v) + 37) approximation for the* Degree-bounded k-Connected Subgraph *problem in undirected graphs with at least k^3 nodes.*

8 Constrained Optimization Problems

In this chapter we examine constrained optimization problems such as the vertex cover problem. This section is based on chapter 11 of [1].

8.1 Vertex Cover

Recall the vertex cover problem:

Problem 8.1. *Given a graph $G = (V, E)$ and a cost function c on vertices, the goal in the vertex cover problem is to find a set of vertices with minimum cost which covers every edge, i.e. for every edge at least one endpoint is in the vertex cover.*

In previous chapters we handled the vertex cover problem in bipartite graphs. In general graphs the vertex cover is NP-hard. In this section we prove a 2-approximation for the problem based on Nemhauser and Trotter [20]. Later we explore the partial vertex cover problem.

8.1.1 Linear Programming Relaxation

We have already seen the LP_{vc} in previous section.

$$\begin{array}{lll} \textit{minimize} & \displaystyle\sum_{v \in V} c_v x_v & \\ \textit{subject to} & x_u + x_v \geq 1 & \forall e = \{u, v\} \in E \\ & x_v \geq 0 & \forall v \in V \end{array}$$

The following is a theorem by Nemhauser and Trotter [20]:

Theorem 8.1. *Let x be an extreme optimal solution to LP_{vc}. Then $x_v \in \{0, \frac{1}{2}, 1\}$ for each $v \in V$.*

We prove this theorem later. For now, we use it without proof. The following theorem derives from Theorem 8.1:

Theorem 8.2. *There exists a 2-approximation algorithm for the vertex cover problem.*

Proof. Assume x is the optimal extreme point solution to LP_{vc}. Let look at the vertex cover constructed by choosing each vertex v such that $x_v = \frac{1}{2}$ or $x_v = 1$. From Theorem 8.1 we get that the cover we chose is feasible and it costs two times as much as the fractional solution x. □

8.1.2 Characterization of Extreme Point Solutions

We again use the characterization we have seen before.

Definition 8.1. *For a set $W \subseteq V$ let $\chi(W)$ denote the* characteristic *vector in $\mathbb{R}^{|V|}$: the vector has an 1 corresponding to each vertex $V \in W$, and 0 otherwise.*

Lemma 8.1. *Given any extreme point x to LP_{vc} with $x_v > 0$ for each $v \in V$, there exists $F \subseteq E$ such that*

1. *$x_u + x_v = 1$ for each $e = \{u, v\} \in F$.*
2. *The vectors in $\{\chi(\{u, v\}) : \{u, v\} \in F\}$ are linearly independent.*
3. *$|V| = |F|$.*

8.1.3 Iterative Algorithm

Algorithm 10 Iterative Vertex Cover Algorithm

```
1: $F \leftarrow \emptyset$;
2: Find an optimal extreme point solution $x$ to $LP_{vc}$.
3: while $E \neq \emptyset$ do
4:     (a) For all vertices $v \in V$ with $x_v = 1$ include $v \in W$ and remove $v$ from $G$.
5:     (b) For all vertices $v \in V$ with $x_v = 0$, remove $v$ from $G$.
6:     (c) For all vertices $v \in V$ with $x_v = \frac{1}{2}$, include $v \in W$ and remove $v$ from $G$.
7: end while
8: Return $W$.
```

8.1.4 Correctness and Performance

Now we go back to Theorem 8.1. By proving this theorem we also prove the correctness of the algorithm.

Proof. In the first step of the loop (step (a) line 4), we remove all vertices with $x_v = 1$. After this step any vertex left with $x_v = 0$ is isolated. So we remove each vertex with $x_v = 0$. The following is applied to each connected component of the graph that is left: From Lemma 8.1 we know that there is a subset F of edges with linearly independent tight constraints and that $|F| = |V|$ (and therefore F contains a cycle. Let us denote this cycle by C. This cycle must be an odd cycle (otherwise the characteristic vectors in $\{\chi(\{u, v\} : \{u, v\} \in E(C)\}$ are linearly dependent). So the unique solution to these equations is $x_v = 1/2$ for each $v \in C$. Since $x_v + x_u = 1$, this implies that $x_u = 1/2$. So all vertices in C have $x_v = \frac{1}{2}$. Proving Theorem 8.1. □

8.2 Partial Vertex Cover

In this section we extend our 2-approximation algorithm to the partial vertex cover problem.

Problem 8.2. *Given a graph $G = (V, E)$ and a cost function c on vertices and a bound L, the* partial vertex cover problem *is to find a set of vertices with minimum cost which covers at least L edges.*

The partial vertex cover problem is NP-hard since it generalizes the vertex cover problem when $L = |E|$. We provide a 2-approximation algorithm based on iterative rounding of a natural linear programming relaxation.

8.2.1 Linear Programming Relaxation

We define a *pruning* step as performing a guess for the *costliest* vertex in the optimal solution. The *pruned* graph is the graph in which we remove all vertices with cost more than *costliest* we have guessed.

Since there are no more than $n = |V|$ possible costliest vertices, it is possible to find the cheapest solution in polynomial time using pruning.

The following linear program LP_{pvc} is the linear program for the pruned instance. Let x_v denote the existence of v in the solution and let y_e denote the existence of $e \in E$ in the partial vertex cover.

$$\begin{aligned}
&\textit{minimize} && \sum_{v \in V} c_v x_v && \\
&\textit{subject to} && \sum_{v \in e} x_v \geq y_e && \forall e \in E \\
& && \sum_{e \in E} y_e = L && \\
& && 0 \leq x_v \leq 1 && \forall v \in V \\
& && 0 \leq y_e \leq 1 && \forall e \in E
\end{aligned}$$

As we proceed with the iterative algorithm, we will work with a graph where edges could be of size one only. For example, when we have a variable with $x_v = 0$, we will remove v from all edges containing v. Such edges will contain only one vertex but not two vertices.

8.2.2 Characterization of Extreme Point Solutions

We give a simple characterization of the extreme point solutions based on the Rank Lemma.

Lemma 8.2. *Given any extreme point x to LP_{pvc} with $0 \leq x_v \leq 1$ for all $v \in V$ and $0 \leq y_e \leq 1$ for all $e \in E$, there exists a subset $F \subseteq E$ of edges such that:*

1. *$\sum_{v \in e} x_v = y_e$ for each $e \in F$.*
2. *The constraints in $\{\sum_{v \in e} x_v = y_e : e \in F\} \cup \{\sum_{e \in E} y_v = L\}$ are linearly independent.*
3. *$|F| + 1 = |V| + |E|$.*

8.2.3 Iterative Algorithm

Algorithm 11 Iterative Partial Vertex Cover Algorithm

```
1: $W \leftarrow \emptyset$;
2: while $E \neq \emptyset$ do
3:     (a) Find an optimal extreme point solution $(x, y)$ to $LP_{pvc}$. If there is an edge $e \in E$ with $y_e = 0$, then remove $e$ from $G$. If there is a vertex $v \in V$ with $x_v = 0$, then remove $v$ from $G$ and from all edges containing it, i.e, $e \leftarrow e \setminus \{v\}$ for all $e \in E$.
4:     (b) If there is a vertex $v \in V$ with $x_v \geq \frac{1}{2}$, then include $v \in W$ and remove $v$ and all the edges incident at $v$ from $G$. Update $L \leftarrow L - |\{e : v \in e\}|$.
5:     (c) If $G$ contains a single vertex $v$, then include $v \in W$ and remove $v$ and all the edges incident at $v$ from $G$. Update $L \leftarrow L - |\{e : v \in e\}|$.
6: end while
7: Return $W$.
```

8.2.4 Correctness and Performance Guarantee

First, we want to show that the algorithm terminates:

Lemma 8.3. *Let G be a graph with $|V(G)| \geq 2$. Then at least one of the following must hold:*

1. *There exists a vertex v with $x_v \in \{0, 1\}$.*

2. *There exists an edge e with $y_e = 0$.*

3. *There exists an edge e with $y_e = 1$ and therefore $x_v = \frac{1}{2}$ for some $v \in e$.*

Proof. By contradiction, we assume that the lemma is not correct. This means that $0 < x_v < 1$ for all $v \in V$ and $0 < y_e < 1$ for all edges. From Lemma 8.2 we know that there is a subset F such that $|F| + 1 = |E| + |V|$. So $|F| \leq |E|$ and therefore $|V| \leq 1$. This is false. So the contradiction fails. $\square$

Now that we know that the algorithm terminates we need to show that the algorithm provides a 2-approximation algorithm for the correct guess of the costliest vertex.

In the second step of the loop we pick a vertex with $x_v \geq \frac{1}{2}$. If $y_e = 1$ then $x_v \geq \frac{1}{2}$ for some $v \in e$ as $|e| \leq 2$. We can see using simple induction that when we pick the vertices with $x_v \geq \frac{1}{2}$ we pay a cost of maximum twice the optimal fractional solution. The last vertex picked (step c) has a cost that is no more than the cost of the costliest vertex. Since the LP value of the costliest vertex was set to 1 in the preprocessing step, the cost of the last vertex picked is also charged to the LP solution. Therefore we get a 2-approximation for the partial vertex cover problem.

9 Conclusions

In this paper we described using a number of examples how to use the iterative method. We examined the vertex cover, minimum spanning tree, matching, minimum cost arborescence, and the survivable network design problems. For each problem we found the corresponding linear program relaxation. We investigated the linear program and were able to prove a number of theorems regarding the extreme point solutions of each linear program. Once this was done we showed the iterative algorithm and proved its correctness. For some problems we only found a good approximation algorithm.

This work only touches some of the problems that can be attacked by the iterative rounding method. It is believed that the iterative methods can be used for more general classes of problems. Many other problems can be (and are being) explored using iterative algorithms. There are a number of books and articles that expand the iterative method. It is possible to find early usage of the iterative relaxation method in the proof of a discrepancy theorem by Beck and Fiala [17] and the approximation algorithm for the bin packing problem by Karmarkar and Karp [16]. Again we refer to Jain [15] for the first explicit usage of the iterative rounding method in an approximation algorithm. The reader is referred to [1] or [21] and [22] for further study.

10 Bibliography

References

[1] L. Chi Lau, R. Ravi and M. Singh, *Iterative Methods in Combinatorial Optimization*, Cambridge University, 2011

[2] A. Ene and A. Vakilian *Improved Approximation Algorithms for Degree-bounded Network Design Problems with Node Connectivity Requirements*, 2013

[3] R. Ravi, *Iterative Methods in Combinatorial Optimization*, Carnegie Mellon University, 2009

[4] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, available online from

www.ecp6.jussieu.fr/pageperso/bondy/books/gtwa/gtwa.html.

[5] S. Iwata, *Submodular function minimization*, Math. Programming, Ser. B, 112(1), 45-64, 2008.

[6] F.S. Hillier and G.J. Lieberman, *Introduction to Operations Research (6th Ed.)*, Mcgraw-Hill, 1995.

[7] V. Chvatal (1983), *Linear Programming*, Freeman

[8] R. J. Vanderbei, *Linear Programming: Foundations and Extensions* (3rd Ed.), Springer, 2007.

[9] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, John Wiley and Sons, New York (1998).

[10] M. Grtschel, L. Lovasz, A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1: 169-197, 1981.

[11] N. Karmarkar, *A New Polynomial Time Algorithm for Linear Programming*, Combinatorica, 4: 373395, 1984

[12] D. Gale, *Linear Programming and the Simplex Method*, Notices of the AMS, 364-369, 2007

[13] J. Edmonds, *Paths, trees and flowers*, Canadioan Journal of Mathematics, 17, 449-467, 1965.

[14] R. Bar-Yehuda, K. Bendel, A. Freund, D. Rawitz, *A unified framework for approximation algorithms*, ACM Computing Surveys 36(4), 422-463, 2004.

[15] K. Jain, *A factor 2 approximation algorithm for the generalized Steiner network problem*, Combinatorica, 21, 39-60, 2001. (Preliminary version in Proc. 39th IEEE FOCS, 1998.)

[16] N. Karmarkar, R. Karp, *An efficient approximation scheme for the one dimensional bin-packing problem*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS), 312-320, 1982.

[17] J. Beck and T. Fiala, *"Integer-making" theorems*, Discrete Applied Mathematics, 3, 1-8, 1981.

[18] V. Vazirani, *Approximation Algorithms*, Springer, 2001.

[19] S.C. Boyd, W.R. Pulleyblank, *Optimizing over the subtour polytope of the travelling salesman problem*, Mathematical Programming, 2, 163-187, 1990.

[20] G.L. Nemhauser, L.E. Trotter, *Vertex packings: structural properties and algorithms*, Mathematical Programming, 8, 232-248, 1975.

[21] J. Knemann, O. Parekh, D. Pritchard, *Max-weight integral multicommodity flow in spiders and high-capacity trees*, in proceedings of Workshop on Approximation and Online Algorithms (WAOA), 1-14, 2008.

[22] D. Pritchard, *Approximability of sparse integer programs*, in Proceedings of European Symposium of Algorithms (ESA), 83-94, 2009.

[23] A. Schrijver, *On the history of combinatorial optimization (till 1960)*, Handbook of Discrete Optimization, 24, Eds. K. Aardal, G.L. Nemhauser and R. Weismantel, 2005.

[24] L. Fleischer, K. Jain, and D. P Williamson. *Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems.* Journal of Computer and System Sciences, 838-867, 2006.

[25] A. Frank. *Rooted k-connections in digraphs.* Discrete Applied Mathematics, 1242-1254, 2009.

[26] A. Frank and E. Tardos. *An application of submodular flows.* Linear algebra and its applications, 329348, 1989.

[27] J. Cheriyan and L. Vegh. *Approximating minimum-cost k-node connected subgraphs via independence-free graphs.* In Proc. of IEEE FOCS, 2013.

תקציר

עבודה זו עוסקת בשיטת העיגול האיטרטיבית, שהיא שיטה פשוטה יחסית אך יעילה לפתרון בעיות אופטימיזציה קלאסיות. הכלליות של השיטה מאפשרת שימוש בשיטה זו עבור מגוון גדול של בעיות אופטימיזציה.
המטרה העיקרית בעבודה היא להדגים את היכולות של שיטת העיגול האיטרטיבית. אנו נדון בנושאים הבסיסיים הנדרשים כדי להבין את השיטה, וביניהן בעיות תכנון לינארי וניתוח פתרונות נקודות קצה של בעיות תכנון לינארי. בהמשך, נציג מספר בעיות מגוונות כגון זיווג בגרף דו-צדדי, עצים פורשים, עצי שטיינר, ועוד. עבור כל בעיה נדגים איך להשתמש בשיטת העיגול האיטרטיבית על מנת לקבל אלגוריתם פולינומי המוצא פתרון אופטימלי או קרוב לאופטימלי.

תוכן העניינים

האוניברסיטה הפתוחה

המחלקה למתמטיקה ומדעי המחשב

שיטת העיגול האיטרטיבית עבור בעיות אופטימיזציה

עבודה מסכמת זו הוגשה כחלק מהדרישות לקבלת תואר

מוסמך למדעים .M.Sc במדעי המחשב

באוניברסיטה הפתוחה

החטיבה למדעי המחשב

מאת

עופר מונין

העבודה הוכנה בהדרכתו של **פרופ׳ זאב נוטוב**

ינואר 2015