Solving Saddle Point Formulations of Linear Programs with Frank-Wolfe

by

Matthew Hough

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2023

Abstract

The problem of solving a linear program (LP) is ubiquitous in industry, yet in recent years the size of linear programming problems has grown and continues to do so. State-of-the-art LP solvers make use of the Simplex method and primal-dual interior-point methods which are able to provide accurate solutions in a reasonable amount of time for most problems. However, both the Simplex method and interior-point methods require solving a system of linear equations at each iteration, an operation that does not scale well with the size of the problem.

In response to the growing size of linear programs and poor scalability of existing algorithms, researchers have started to consider first-order methods for solving large scale linear programs. The best known first-order method for general linear programming problems is PDLP [2, 3]. First-order methods for linear programming are characterized by having a matrix-vector product as their primary computational cost.

We present a first-order primal-dual algorithm for solving saddle point formulations of linear programs, named FWLP (Frank-Wolfe Linear Programming). We provide some theoretical results regarding the behavior of our algorithm, however no convergence guarantees are provided. Numerical investigations suggest that our algorithm has error $\mathcal{O}(1/\sqrt{k})$ after k iterations, worse than that of PDLP, however we show that our algorithm has advantages for solving very large LPs in practice such as only needing part of the matrix A at each iteration.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Stephen Vavasis. Steve's guidance throughout the past two years has been paramount to my success and I very much look forward to learning more from him over the coming years as I complete my PhD.

I would also like to thank Dr. Lindon Roberts for mentoring me as an undergraduate student in Australia. Lindon taught me so much about continuous optimization and mathematics research in general. I find myself applying the lessons I learned while working with Lindon on a weekly basis.

Over the past few years the advice and encouragement of Prof. Coralia Cartis, Prof. Diane Donovan, Dr. Fred Roosta, and Prof. Simon Lucey has been a massive help and I am very fortunate to have had their support. Thank you!

I am also very appreciative of my readers Prof. Henry Wolkowicz and Asst. Prof. Walaa Moursi for their thorough reading of my thesis and insightful comments.

Most importantly, I would like to thank my friends and family. I am so grateful for their presence in my life. Special thanks go to my roommate Antonio and girlfriend Tina.

Dedication

For Judy, Alex, Betty, and Lex.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Linear programming has been an essential tool in optimization since George Dantzig's work on the Simplex method in the 1940s. Industrial optimization problems such as scheduling, chip design, budget allocation, and many more have long been modeled as linear programs (LPs). Software for solving linear programs predates operating systems. The state-of-the-art algorithms used in LP solvers, namely the Simplex method and primal-dual interior-point methods (IPMs) are widely considered to be mature in that they reliably return solutions with high accuracy. Despite the differences between the Simplex method and primal-dual interior-point methods, both algorithms rely on the solution of a system of linear equations.

In recent years, data science applications have given birth to problems of very large scale. Now, it is not uncommon for linear programming problems to have billions of variables. This poses a problem for mature LP solvers that require solving a system of linear equations at each iteration. First-order methods for linear programming aim to solve LPs in such a way that their most expensive operation at each iteration is the product of a matrix and a vector. Their goal is to provide an alternative to the practitioner over LP solvers such as Simplex or IPMs for large-scale problems. First-order methods are not without fault, however. While their strength is their low cost per iteration, their weakness is their inability to provide highly accurate solutions. The practitioner thus must make a trade-off between accuracy and efficiency for the problem at hand.

Thesis outline

In this thesis we propose a first-order primal-dual algorithm for linear programming inspired by the Frank-Wolfe algorithm [20]. We call our algorithm FWLP. The thesis begins by introducing the necessary background for the algorithm in Chapter 3, and the related literature in Chapters 4 and 5, before introducing FWLP and related theoretical results in Chapter 6. While FWLP has been observed to converge empirically, we are unable to provide a convergence proof for the algorithm. Nonetheless, we devise separate potential functions for the case where the dual iterate is infeasible and feasible and show that in each case such potential functions are equivalent by a constant factor to the standard primal-dual gap used to measure optimality of saddle-point problems. We prove that these potential functions decrease by a constant factor at each iteration. This work can be found in Section 6.1 of Chapter 6. Section 6.2 relates FWLP to the Generalized Fictitious Play algorithm proposed in [30]. In Chapter 6, Sections 6.4 and 6.5, we perform numerical experiments on five random linear programs in order to determine parameters of FWLP that work well in practice, and to test the effect restarting FWLP has on the observed convergence rate of the algorithm. In Section 6.6, we investigate the convergence rate of FWLP numerically. Our results suggest that FWLP reduces the error to $\mathcal{O}(1/\sqrt{k})$ after k iterations, which is slower than the best known first-order method for linear programming, PDLP [2, 3]. However, in Section 6.3 we describe how FWLP can be implemented in a way that only requires part of the matrix A at each iteration, making iterations very efficient in comparison to other linear programming algorithms.

Chapter 2

Notation

Throughout this thesis we use boldface for vectors. For example, $\boldsymbol{x} \in \mathbb{R}^n$ is a vector, while $\xi \in \mathbb{R}$ is a scalar. The variable $k \in \mathbb{N}$ is always used to denote the kth iteration in an algorithm, and we write $\boldsymbol{x}_k$ to denote the value of $\boldsymbol{x}$ at the kth iteration. We also use subscripts to index the entries of a vector, however this will always be done with a variable other than k, and the vector being indexed will not be boldfaced. For example, x_i refers to the ith entry of $\boldsymbol{x}$. If we want to index multiple entries of a vector, we use the notation $\boldsymbol{x}_S$, where S is a set of the indices to be selected. $\boldsymbol{x}_S$ is a vector of length $|S|$. Matrices are always capitalized and are denoted by letters A and B. We use Matlab notation to index matrices, so $A(:,j)$ refers to the jth column of A (as a column vector), $A(i,:)$ refers to the ith row of A (as a row vector), and $A(i,j)$ refers to the ijth entry of A, a scalar. The identity matrix is denoted I. Where the size of a matrix cannot be easily inferred from context, we use subscript notation to describe its size. For example, $A_{m\times n}$ means A is an $m \times n$ matrix. When dealing with operators, Id refers to the identity operator. $\mathcal{N}_C$ denotes the normal cone of the convex set $\mathcal{C}$ and $P_{\mathcal{C}}(\boldsymbol{x})$ denotes the projection of $\boldsymbol{x}$ onto the set $\mathcal{C}$. The projection of $\boldsymbol{x}$ onto the nonnegative orthant receives special attention and is denoted $[\boldsymbol{x}]_+$.

Chapter 3

Background

3.1 Continuous Optimization

Given a continuous function $f : \mathbb{R}^n \to \mathbb{R}$ and a set $\mathcal{C} \subseteq \mathbb{R}^n$, the continuous optimization problem is

$$\begin{array}{ll} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in \mathcal{C}. \end{array} \tag{3.1}$$

We call f the *objective function* and $\mathcal{C}$ the *feasible set*. To solve (3.1) is to find a global minimizer, defined as follows.

Definition 3.1.1. A point $\boldsymbol{x} \in \mathbb{R}^n$ is a **global minimizer** of (3.1) if $\boldsymbol{x} \in \mathcal{C}$, and it satisfies $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{C}$.

Definition 3.1.1 is not particularly useful from an algorithmic standpoint, since an algorithm usually only has local information about the objective function. Algorithms generally search for a local minimizer:

Definition 3.1.2. A point $\boldsymbol{x} \in \mathbb{R}^n$ is a **local minimizer** of (3.1) if $\boldsymbol{x} \in \mathcal{C}$ and there exists a neighborhood $\mathcal{N}$ such that $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{N} \cap \mathcal{C}$.

Suppose the feasible set $\mathcal{C}$ is defined by the equations and inequalities

$$\mathcal{C} = \{\boldsymbol{x} \in \mathbb{R}^n : c_i(\boldsymbol{x}) = 0 \text{ for } i \in \mathcal{E}, c_i(\boldsymbol{x}) \geq 0 \text{ for } i \in \mathcal{I}\},$$

where c_i are all differentiable, $\mathcal{E}$ and $\mathcal{I}$ are mutually exclusive sets of indices, and $m := |\mathcal{E} \cup \mathcal{I}|$. In addition suppose now that f is differentiable and consider the following definition:

Definition 3.1.3. Given $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$, the **Lagrangian** $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ associated with (3.1) is defined by

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = f(\boldsymbol{x}) + \sum_{i \in \mathcal{E} \cup \mathcal{I}} y_i c_i(\boldsymbol{x}).$$

The following theorem gives the first-order necessary conditions for $\boldsymbol{x}^*$ to be a local minimizer of (3.1).

Theorem 3.1.4. Suppose that $\boldsymbol{x}^*$ is a local minimizer of (3.1) and an appropriate constraint qualification[1] holds. There must exist a $\boldsymbol{y}^* \in \mathbb{R}^m$ such that the following conditions, known as *KKT conditions* are all satisfied:

$$\begin{aligned}
\nabla_x \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{y}^*) &= 0, && && (3.2)\\
c_i(\boldsymbol{x}^*) &= 0, && \text{for all } i \in \mathcal{E}, && (3.3)\\
c_i(\boldsymbol{x}^*) &\geq 0, && \text{for all } i \in \mathcal{I}, && (3.4)\\
y_i^* &\leq 0, && \text{for all } i \in \mathcal{I}, && (3.5)\\
y_i^* c_i(\boldsymbol{x}^*) &= 0, && \text{for all } i \in \mathcal{E} \cup \mathcal{I}. && (3.6)
\end{aligned}$$

First-order necessary conditions are the basis of most continuous optimization algorithms. Iteratively, algorithms search for points that satisfy the necessary optimality conditions of the problem.

In the next section, we consider the special case of continuous optimization where f is linear and $\mathcal{C}$ is a polyhedron.

3.2 Linear Programming

Let m, n be positive integers, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$, and A be an $m \times n$ real matrix. We call any continuous optimization problem of the form

$$\begin{array}{ll}
\underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} & \boldsymbol{c}^\top \boldsymbol{x} \\
\text{subject to} & A\boldsymbol{x} = \boldsymbol{b}, \\
& \boldsymbol{x} \geq \boldsymbol{0},
\end{array} \qquad \text{(LP)}$$

[1]For a background on constraint qualifications, see [54].

a *linear program*, or LP for short[2]. By observing that both the objective and constraint functions are affine, we conclude that linear programming is a special case of convex programming, i.e. the minimization of a convex objective subject to convex constraints. It follows that any local minimizer of (LP) is a global minimizer (cf. [48, Theorem 2.5]).

3.2.1 Linear Programming Geometry

Consider the feasible set of (LP),

$$P = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}.$$

P is a polyhedron in standard form. Throughout this thesis we will make the following standard assumptions.

Assumption 1. The m rows of A are linearly independent.

Assumption 2. The feasible set P of (LP) is non-empty.

In fact, Assumption 1 can be made without loss of generality ([6, Theorem 2.5]). This makes sense, as linearly dependent rows in A would correspond to redundant constraints that are unnecessary for the representation of P. Assumption 1 implies that $\operatorname{rank}(A) = m$, and since the rows of A are n-dimensional, this requires $m \leq n$.

The following gives a geometric definition of a 'corner' point of P.

Definition 3.2.1. The vector $\boldsymbol{x} \in P$ is a **vertex** of P if there exists a $\boldsymbol{c} \in \mathbb{R}^n$ such that $\boldsymbol{c}^\top \boldsymbol{x} < \boldsymbol{c}^\top \boldsymbol{y}$ for all $\boldsymbol{y} \in P$ different from $\boldsymbol{x}$.

We now give the definition of what it means to be at a 'corner' point of P in the algebraic sense.

Definition 3.2.2. A point $\boldsymbol{x} \in \mathbb{R}^n$ is a **basic solution** of P if $A\boldsymbol{x} = \boldsymbol{b}$, and there exist indices $B(1), \ldots, B(m)$ such that

(a) The columns $A_{B(1)}, \ldots, A_{B(m)}$ are linearly independent.

(b) If $i \notin \{B(1), \ldots, B(m)\}$, then $\boldsymbol{x}_i = 0$.

If in addition, $\boldsymbol{x} \geq \boldsymbol{0}$, then $\boldsymbol{x}$ is a **basic feasible solution** (BFS).

[2]For a more complete introduction to linear programming, refer to [6].

If $\boldsymbol{x}$ is a basic solution, the variables $\boldsymbol{x}_{B(1)}, \dots, \boldsymbol{x}_{B(m)}$ are known as the basic variables, while the remaining variables are known as the nonbasic variables. The columns $A_{B(1)}, \dots, A_{B(m)}$ are called the basic columns, and since they are linearly independent, they form a basis of $\mathbb{R}^m$. If we set $B := \{B(1), \dots, B(m)\}$, we use A_B to denote the *basis matrix*, formed by

$$A_B = \begin{bmatrix} A_{B(1)} & A_{B(2)} & \dots & A_{B(m)} \end{bmatrix}.$$

Definition 3.2.3. A basic solution $\boldsymbol{x} \in \mathbb{R}^n$ of P is said to be **degenerate** if more than $n - m$ entries of $\boldsymbol{x}$ are zero.

Intuitively, there can only be finitely many corners to a polyhedron. This is captured in the following fact:

Fact 3.2.4 ([6, Corollary 2.1])**.** For any P, there can only be a finite number of basic solutions.

We've been calling basic solutions 'corner points' but have not yet confirmed the relation between the algebraic definition of a basic solution and the geometric concept of a vertex of P. It can be shown that vertices of P are equivalent to basic feasible solutions. While the definition of a vertex is useful in theory, it is the algebraic definition of a BFS that is useful when designing algorithms.

Fact 3.2.5 ([6, Theorem 2.3])**.** Let $\boldsymbol{x} \in P$. Then $\boldsymbol{x}$ is a vertex of P if and only if $\boldsymbol{x}$ is a basic feasible solution of P.

Under our assumption that P is a non-empty polyhedron in standard form, it can be shown that P always has at least one basic feasible solution [6, Corollary 2.2]. Moreover, if we consider the linear programming problem corresponding to P, (LP), either the optimal cost is unbounded, or there exists a vertex of P that is optimal for the linear program.

3.2.2 Linear Programming Duality

The *dual* of (LP) is given by the following linear program.

$$\begin{aligned} \underset{\boldsymbol{y} \in \mathbb{R}^m}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} \\ \text{subject to} \quad & A^\top \boldsymbol{y} \leq \boldsymbol{c}. \end{aligned} \tag{DLP}$$

Assuming (LP) and (DLP) both have feasible solutions, the weak duality theorem in linear programming tells us that the objective value of any feasible point of (DLP) is always a lower bound on the objective value of (LP).

Consider the partial Lagrangian function associated with (LP), ignoring the sign constraints. This can be viewed as the objective in (LP) plus a weighted sum of the constraint violations $\boldsymbol{b}_i - A(i,:)\boldsymbol{x}$.

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x}) \tag{3.7}$$

Let $d(\boldsymbol{y}) = \min\{\mathcal{L}(\boldsymbol{x},\boldsymbol{y}) : \boldsymbol{x} \geq \boldsymbol{0}\}$ and suppose there exists an optimal solution $\boldsymbol{x}^*$ to (LP). Since in $d(\boldsymbol{y})$ we are minimizing over a supserset of the feasible set in (LP), we have that

$$d(\boldsymbol{y}) \leq \boldsymbol{c}^\top \boldsymbol{x}^* + \boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x}^*) = \boldsymbol{c}^\top \boldsymbol{x}^* \tag{3.8}$$

for any $\boldsymbol{y} \in \mathbb{R}^m$, where the equality follows from the fact that $A\boldsymbol{x}^* = \boldsymbol{b}$. We can view the problem of maximizing $d(\boldsymbol{y})$ over all $\boldsymbol{y} \in \mathbb{R}^m$, as looking for the greatest lower bound to (LP). Moreover, note

$$\begin{aligned} d(\boldsymbol{y}) &= \min_{\boldsymbol{x}\geq\boldsymbol{0}}\{\boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x})\} \\ &= \boldsymbol{y}^\top\boldsymbol{b} + \min_{\boldsymbol{x}\geq\boldsymbol{0}}(\boldsymbol{c} - A^\top\boldsymbol{y})^\top\boldsymbol{x} \end{aligned} \tag{3.9}$$

where the second term in the above is either zero when $\boldsymbol{c} - A^\top\boldsymbol{y} \geq \boldsymbol{0}$, or unbounded otherwise. In looking for a $\boldsymbol{y}$ that maximizes $d(\boldsymbol{y})$, we are implicitly looking for a $\boldsymbol{y}$ that satisfies the constraint $\boldsymbol{c} - A^\top\boldsymbol{y} \geq \boldsymbol{0}$. Otherwise, a maximum is not obtained. That is, we want $\max\{d(\boldsymbol{y}) : \boldsymbol{c} - A^\top\boldsymbol{y} \geq \boldsymbol{0}\}$, which is equivalent to (DLP).

If instead, we let $p(\boldsymbol{x}) = \max\{\mathcal{L}(\boldsymbol{x},\boldsymbol{y}) : \boldsymbol{y} \in \mathbb{R}^m\}$ and suppose there exists an optimal solution $\boldsymbol{y}^*$ to (DLP), we obtain the following for all $\boldsymbol{x} \geq \boldsymbol{0}$.

$$p(\boldsymbol{x}) \geq \boldsymbol{b}^\top\boldsymbol{y}^* + (\boldsymbol{c} - A^\top\boldsymbol{y}^*)^\top\boldsymbol{x} \geq \boldsymbol{b}^\top\boldsymbol{y}^* \tag{3.10}$$

Here the first inequality comes from the fact that in $p(\boldsymbol{x})$ we are maximizing over a superset of the feasible set in (DLP), and the second inequality comes from the feasibility of $\boldsymbol{y}^*$ for (DLP) and the assumption that $\boldsymbol{x} \geq \boldsymbol{0}$. Hence, we can view the problem of minimizing $p(\boldsymbol{x})$ over all $\boldsymbol{x} \geq \boldsymbol{0}$, as looking for the least upper bound to (DLP). Since we have

$$\begin{aligned} \min_{\boldsymbol{x}\geq 0} p(\boldsymbol{x}) &= \min_{\boldsymbol{x}\geq 0}\max_{\boldsymbol{y}}\{\boldsymbol{c}^\top\boldsymbol{x} + \boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x})\} \\ &= \min_{\boldsymbol{x}\geq 0}\{\boldsymbol{c}^\top\boldsymbol{x} + \max_{\boldsymbol{y}}\boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x})\} \\ &= \min_{\boldsymbol{x}\geq\boldsymbol{0}}\begin{cases}\boldsymbol{c}^\top\boldsymbol{x}, & A\boldsymbol{x} = \boldsymbol{b}, \\ +\infty, & \text{otherwise,}\end{cases} \end{aligned}$$

and we only want to consider the bounded case as before, we find that our search for a least upper bound to (DLP) is equivalent to $\min\{p(\boldsymbol{x}) : A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$. But this is the same problem as (LP).

From weak duality, we now have that

$$\max\{d(\boldsymbol{y}) : \boldsymbol{y} \in \mathbb{R}^m\} = \boldsymbol{b}^\top \boldsymbol{y}^* \leq \boldsymbol{c}^\top \boldsymbol{x}^* = \min\{p(\boldsymbol{x}) : \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{x} \in \mathbb{R}^n\} \tag{3.11}$$

If in addition, we assume that (LP) has an optimal solution, then the theorem of strong duality says that (DLP) also has an optimal solution, and the inequality in (3.11) becomes an equality. This implies the following equation.

$$\max_{\boldsymbol{y} \in \mathbb{R}^m} \min_{\boldsymbol{x} \geq \boldsymbol{0}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = \min_{\boldsymbol{x} \geq \boldsymbol{0}} \max_{\boldsymbol{y} \in \mathbb{R}^m} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) \tag{3.12}$$

3.2.3 Certificates of infeasibility

Consider the constraints of (LP), $A\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$. If there exists a vector $\boldsymbol{p}$ such that $\boldsymbol{p}^\top A \geq \boldsymbol{0}^\top$ and $\boldsymbol{p}^\top \boldsymbol{b} < 0$, then for any $\boldsymbol{x} \geq \boldsymbol{0}$, we have $\boldsymbol{p}^\top A\boldsymbol{x} \geq 0$. However, since $\boldsymbol{p}^\top \boldsymbol{b} < 0$, it is impossible for $\boldsymbol{p}^\top A\boldsymbol{x} = \boldsymbol{p}^\top \boldsymbol{b}$, i.e. $A\boldsymbol{x} \neq \boldsymbol{b}$ must hold for all $\boldsymbol{x} \geq \boldsymbol{0}$. Such a vector $\boldsymbol{p}$ is called a *certificate of infeasibility*, because whenever we can find such a $\boldsymbol{p}$, we can certify that there do not exist any solutions to the standard form constraints. The following lemma, known as Farkas' lemma, states that whenever (LP) is infeasible, a certificate of infeasibility must exist.

Theorem 3.2.6 (Farkas' lemma)**.** Exactly one of the following holds:

(a) There exists an $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x} \geq \boldsymbol{0}$ such that $A\boldsymbol{x} = \boldsymbol{b}$.

(b) There exists a $\boldsymbol{p} \in \mathbb{R}^m$ such that $\boldsymbol{p}^\top A \geq \boldsymbol{0}$ and $\boldsymbol{p}^\top \boldsymbol{b} < 0$.

3.3 Saddle Point Formulations

Consider a convex-concave function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. The saddle point problem is

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} f(\boldsymbol{x}, \boldsymbol{y}) \tag{3.13}$$

where $\mathcal{X}, \mathcal{Y}$ are non-empty, closed, and convex sets. Solutions to (3.13) are known as *saddle points* of f, defined as follows.

Definition 3.3.1. A point $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in \mathcal{X} \times \mathcal{Y}$ is a saddle point of f if it satisfies

$$f(\boldsymbol{x}^*, \boldsymbol{y}) \leq f(\boldsymbol{x}^*, \boldsymbol{y}^*) \leq f(\boldsymbol{x}, \boldsymbol{y}^*), \quad \text{for all } \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} \in \mathcal{Y}.$$

By observing that the Lagrangian defined in (3.7) is a convex-concave function, and that the sets $\mathbb{R}^n_+$ and $\mathbb{R}^m$ are closed and convex, we find that (3.13) generalizes the linear program (LP), assuming that the inner maximization is bounded. Moreover, it is not hard to see that if $\boldsymbol{x}^*$ is an optimal solution to (LP) and $\boldsymbol{y}^*$ is an optimal solution to (DLP), the pair $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is a saddle point for $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y})$ over the feasible set $\mathbb{R}^n_+ \times \mathbb{R}^m$. Because for any $\boldsymbol{x} \in \mathbb{R}^n_+$ and $\boldsymbol{y} \in \mathbb{R}^m$,

$$\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{y}) = \boldsymbol{c}^\top \boldsymbol{x}^* + \boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x}^*) = \boldsymbol{c}^\top \boldsymbol{x}^* = \boldsymbol{b}^\top \boldsymbol{y}^* \leq \boldsymbol{b}^\top \boldsymbol{y}^* + (\boldsymbol{c} - A^\top \boldsymbol{y}^*)^\top \boldsymbol{x} = \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}^*)$$

where the second equality comes from the feasibility of $\boldsymbol{x}^*$, the third equality comes from strong duality, and the inequality comes from the feasibility of $\boldsymbol{y}^*$. Since $\boldsymbol{c}^\top \boldsymbol{x}^* = \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{y}^*)$, we get that

$$\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{y}) \leq \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{y}^*) \leq \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}^*).$$

With this in mind, we may choose to instead solve the saddle point formulation of (LP).

3.4 The Frank-Wolfe Algorithm

Consider the optimization problem

$$\underset{\boldsymbol{x} \in \mathcal{C}}{\text{minimize}} \quad f(\boldsymbol{x}) \tag{3.14}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is convex, and $\mathcal{C} \subseteq \mathbb{R}^n$ is a non-empty, compact, and convex set. The Frank-Wolfe algorithm [20] (see Algorithm 3.1), also known as the Conditional Gradient method, iteratively solves (3.14), with the defining feature being that at each iteration it optimizes a linear function over the convex constraint set $\mathcal{C}$. Throughout this thesis, we use the step-size $1/(k+1)$ for the Frank-Wolfe algorithm, analyzed in [21]

Algorithm 3.1 The Frank-Wolfe algorithm.

```
1: Choose $\boldsymbol{x}_0 \in \mathcal{C}$.
2: for $k = 1, 2, \ldots$ do
3:     Find a step that decreases the first-order approximation of $f$:
           $\boldsymbol{s}_k := \operatorname{argmin}\{f(\boldsymbol{x}_k) + \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{s}\rangle : \boldsymbol{s} \in \mathcal{C}\}$    (3.15)
4:     Update $\boldsymbol{x}_{k+1}$ as a convex combination between $\boldsymbol{x}_k$ and $\boldsymbol{s}_k$:
           $\boldsymbol{x}_{k+1} := \left(1 - \frac{1}{k+1}\right)\boldsymbol{x}_k + \frac{1}{k+1}\boldsymbol{s}_k$    (3.16)
```

Since $f(\boldsymbol{x}_k)$ is a constant in (3.15), this step reduces to $\boldsymbol{s}_k := \operatorname{argmin}\{\langle \nabla f(\boldsymbol{x}_k), \boldsymbol{s}\rangle : \boldsymbol{s} \in \mathcal{C}\}$. For many convex sets $\mathcal{C}$, this linear optimization problem can be much less costly to solve than a projected gradient step, and often exhibits a closed-form solution [34].

3.5 Splitting Algorithms

Suppose X is some Hilbert space and consider the operators $A, B : X \rightrightarrows X$. The next few definitions will be useful in what follows.

Definition 3.5.1 ([5, Definition 23.1]). The **resolvent** of A is $J_A = (\operatorname{Id} + A)^{-1}$.

Definition 3.5.2 ([5, Corollary 23.11 (ii)]). The **reflected resolvent** of A is $R_A = 2J_A - \operatorname{Id}$.

Definition 3.5.3 ([5, Definition 4.1 (ii)]). Suppose $T : X \to X$. Then T is **nonexpansive** if for all $x, y \in X$,

$$\|Tx - Ty\| \leq \|x - y\|,$$

and T is **firmly nonexpansive** if $2T - \operatorname{Id}$ is nonexpansive (cf. [5, Proposition 4.4]).

Definition 3.5.4 ([5, Definition 20.1]). A is **monotone** if for any $(x, u), (y, v) \in \operatorname{gr} A$,

$$\langle x - y, u - v\rangle \geq 0.$$

Definition 3.5.5 ([5, Definition 20.20]). A is **maximally monotone** if it is monotone and it is impossible to properly enlarge $\operatorname{gr} A$ without losing monotonicity.

Definition 3.5.6 ([5, Definition 22.1 (iv)]). *A is **β-strongly monotone** if there exists a $\beta > 0$ such that for any $(x, u), (y, v) \in \operatorname{gr} A$,*

$$\langle x - y, u - v \rangle \geq \beta \|x - y\|^2.$$

For the remainder of this section we will make the assumption that X is some Hilbert space and that A and B are maximally monotone operators.

Splitting algorithms in optimization solve the monotone inclusion problem

$$0 \in Ax + Bx \tag{3.17}$$

where $A, B : X \rightrightarrows X$ are maximally monotone and X is some Hilbert space. This is a generalization of the convex optimization problem

$$\underset{x \in X}{\text{minimize}} \quad f(x) + g(x) \tag{3.18}$$

where f and g are proper, lower semicontinuous, and convex functions on X. Under appropriate constraint qualifications [51, Theorem 23.8], the sum-rule for subgradients holds: $\partial(f + g) = \partial f + \partial g$, allowing us to apply Fermat's Theorem to equivalently write (3.18) as the problem of finding $x \in X$ such that

$$0 \in \partial(f + g)x = \partial f(x) + \partial g(x).$$

By Fact 3.5.7, the above is a monotone inclusion problem in the framework of (3.17).

Fact 3.5.7 ([5, Theorem 20.25]). *Suppose $f : X \to (-\infty, +\infty]$ is convex, lower semicontinuous, and proper. The subgradient ∂f is maximally monotone.*

The two splitting algorithms relevant to our work are the Forward-Backward splitting method (FB) [5, Chapter 26.5] and the Douglas-Rachford algorithm, originally proposed in [16] to solve certain partial differential equations, but extended to the case of solving (3.17) in [43]. FB iteratively applies the operator $T = J_B \circ (\operatorname{Id} - A)$ until a fixed point is reached, where A is assumed to be firmly nonexpansive. Since it can be shown that $\operatorname{zer}(A + B) = \operatorname{Fix} T$ (for example, see the proof of [5, Corollary 28.9]), FB solves the monotone inclusion problem (3.17).

The Douglas-Rachford algorithm iteratively applies the firmly nonexpansive operator $T = \operatorname{Id} - J_A + J_B R_A$. Combettes [14, Lemma 2.6(iii)] showed that $\operatorname{zer}(A + B) = J_A(\operatorname{Fix} T)$. Hence, to solve (3.17) with DR, one would apply T to an initial point x_0 until a fixed point x^F is reached. To obtain the solution, one would need to finally apply the resolvent: $x^* = J_A x^F$.

Chapter 4

Literature Review

4.0.1 Saddle point problems

The saddle point problem (3.13) is generalized by the variational inequality problem (VIP) first introduced in 1966 by Hartman and Stampacchia [31]. The VIP is to find a $\boldsymbol{z}^* \in \mathcal{Z}$ such that

$$\langle F(\boldsymbol{z}^*), \boldsymbol{z} - \boldsymbol{z}^* \rangle \geq 0, \quad \forall \boldsymbol{z} \in \mathcal{Z}, \tag{4.1}$$

where F is a continuous and monotone mapping from $\mathbb{R}^p$ to itself and $\mathcal{Z} \subseteq \mathbb{R}^p$ is nonempty, closed and convex. Consider the sets $\mathcal{X}$ and $\mathcal{Y}$ from Section 3.3, as well as the function f, assuming it is continuously differentiable. By choosing $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ and $F(\boldsymbol{z}) = (\nabla_{\boldsymbol{x}} f(\boldsymbol{z}), -\nabla_{\boldsymbol{y}} f(\boldsymbol{z}))$, (4.1) becomes equivalent to finding $(\boldsymbol{x}^*, \boldsymbol{y}^*) \in \mathcal{X} \times \mathcal{Y}$ such that they are optimal for (3.13), i.e. finding a point at which there does not exist a feasible descent direction. Note that the constructed F is monotone by Proposition 4.0.1 and the convexity assumption on f.

Proposition 4.0.1. Consider continuously differentiable $f : \mathbb{R}^p \to \mathbb{R}$. f is convex if and only if ∇f is monotone.

Proof. Suppose f is convex. By the subgradient inequality, we have that

$$f(\boldsymbol{x}) - f(\boldsymbol{y}) \geq \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p, \tag{4.2}$$

and

$$f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p. \tag{4.3}$$

Adding (4.2) and (4.3) gives the following

$$0 \geq \langle \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{y} - \boldsymbol{x} \rangle,$$

which is equivalent to the monotonicity condition:

$$\langle \boldsymbol{x} - \boldsymbol{y}, \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}) \rangle \geq 0$$

For the other direction, let $g(t) = f((1-t)\boldsymbol{x} + t\boldsymbol{y})$ be defined on $t \in [0,1]$. Differentiating $g(t)$ gives $g'(t) = \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f((1-t)\boldsymbol{x} + t\boldsymbol{y}) \rangle$. Now choose any $t_1, t_2 \in [0,1]$ and set $\boldsymbol{a} = (1-t_1)\boldsymbol{x} + t_1\boldsymbol{y}$ and $\boldsymbol{b} = (1-t_2)\boldsymbol{x} + t_2\boldsymbol{y}$. We now observe that g is also monotone:

$$\begin{aligned}(t_1 - t_2) \cdot (g'(t_1) - g'(t_2)) &= (t_1 - t_2) \cdot \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f((1-t_1)\boldsymbol{x} + t_1\boldsymbol{y}) - \nabla f((1-t_2)\boldsymbol{x} + t_2\boldsymbol{y}) \rangle \\ &= \langle \boldsymbol{a} - \boldsymbol{b}, \nabla f(\boldsymbol{a}) - \nabla f(\boldsymbol{b}) \rangle \\ &\geq 0.\end{aligned}$$

Since $g'(t)$ is monotone in one dimension, the epigraph of $g(t)$ is clearly convex, and $g(t)$ is a convex function. The convexity of f follows almost immediately, since for all $\lambda \in [0,1]$,

$$f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) = g(1-\lambda) \leq \lambda g(0) + (1-\lambda)g(1) = \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}). \tag{4.4}$$

□

Projection-based algorithms are widely used for solving VIPs and subsequently saddle point problems. The original projection method used for solving the VIP is the Goldstein-Levitin-Polyak gradient projection method, proposed first by Goldstein [26] and then independently a year later by Levitin and Polyak [40]. The method uses the iteration scheme

$$\boldsymbol{z}_{k+1} = \mathbf{P}_{\mathcal{Z}}(\boldsymbol{z}_k - \alpha F(\boldsymbol{z}_k)), \quad k = 1, 2, 3, \ldots \tag{4.5}$$

where $\alpha > 0$ is fixed. Under the assumptions that F is Lipschitz continuous and strongly monotone, the Goldstein-Levitin-Polyak method can be shown to converge in the sense that $F(\boldsymbol{z}_k) \to F(\boldsymbol{z}^*)$ at a rate of $\mathcal{O}(1/k)$, assuming $\alpha > 0$ is sufficiently small. Under such assumptions, the gradient projection method is a special case of the Forward-Backward splitting method. Of course, for any $\alpha > 0$, $0 \in Ax + Bx \iff 0 \in \alpha Ax + \alpha Bx$. If B is a cone, we have that $\alpha A + \alpha B = \alpha A + B$ by the definition of a cone. Under the assumption that F is β-strongly monotone and L-Lipschitz, we obtain

$$\begin{aligned}\langle \boldsymbol{w} - \boldsymbol{z}, F(\boldsymbol{w}) - F(\boldsymbol{z}) \rangle &\geq \beta \|\boldsymbol{w} - \boldsymbol{z}\|^2, \\ &\geq \frac{\beta}{L^2} \|F(\boldsymbol{w}) - F(\boldsymbol{z})\|^2,\end{aligned}$$

for all $\boldsymbol{w}, \boldsymbol{z} \in \mathcal{Z}$. The first inequality comes from the β-strong monotonicity of F, and the second inequality comes from the Lipschitz assumption on F. If we set $\alpha := L^2/\beta$, it follows that αF is firmly nonexpansive. To see the connection between the Goldstein-Levitin-Polyak method and FB, let $B = \mathcal{N}_{\mathcal{Z}}$, i.e. the normal cone of $\mathcal{Z}$, and $A = \alpha F$. The FB operator becomes $T = \mathbf{P}_{\mathcal{Z}}(\mathrm{Id} - \alpha F)$, i.e. $T\boldsymbol{z} = \mathbf{P}_{\mathcal{Z}}(\boldsymbol{z} - \alpha F(\boldsymbol{z}))$, which is exactly the operator used in the Goldstein-Levitin-Polyak gradient projection method.

For many use cases, the assumption of strong monotonicity may be too strong. This motivates the development of the extragradient method (EGM) due to Korpelevich [38], and its variants. EGM uses the iteration scheme for $k = 1, 2, 3, \ldots$

$$\bar{\boldsymbol{z}} = \mathbf{P}_{\mathcal{Z}}(\boldsymbol{z}_k - \alpha F(\boldsymbol{z}_k)), \tag{4.6}$$

$$\boldsymbol{z}_{k+1} = \mathbf{P}_{\mathcal{Z}}(\boldsymbol{z}_k - \alpha F(\bar{\boldsymbol{z}})), \tag{4.7}$$

where $\alpha > 0$ is fixed. Of note here is the fact that we are now evaluating F twice and computing two projections at each iteration. Korpelevich proves that EGM converges if F is monotone and Lipschitz continuous on $\mathcal{Z}$, assuming $\alpha > 0$ is small enough [38]. In view of Proposition 4.0.1, the assumption that F is monotone is much more reasonable for solving (3.13), since we have that f is convex. Recent work [27, 10] has shown that under the assumptions of strong monotonicity and Lipschitz continuity of F, EGM converges in $\mathcal{O}(1/k)$ iterations for (4.1).

A classic algorithm for solving (3.13) directly is the proximal point method (PPM) proposed by Martinet in 1970 [45] and extended by Rockafellar in 1976 [52]. The PPM iterates by applying the proximal operator to the saddle point objective:

$$\boldsymbol{z}_{k+1} = (\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}) \in \arg\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} \left\{ f(\boldsymbol{x}, \boldsymbol{y}) + \frac{1}{2\eta}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2 + \frac{1}{2\eta}\|\boldsymbol{y} - \boldsymbol{y}_k\|_2^2 \right\} \tag{4.8}$$

where the function f and sets $\mathcal{X}$ and $\mathcal{Y}$ are as in Section 3.3. As with the Goldstein-Levitin-Polyak method and EGM, PPM is evaluating a quadratic at each iteration. However, it comes with the added complexity of having to solve a minimax problem at each iteration as well. The benefit of PPM here over the aforementioned two algorithms is that it achieves a $\mathcal{O}(1/k)$ rate of convergence without any assumptions other than f being convex [29]. Eckstein and Bertsekas showed in 1992 that PPM is equivalent to the Douglas-Rachford splitting method [19].

It is common for the saddle point objective to have a specific structure. One such structure is the bilinear saddle point problem, which models the Lagrangian for linear programming. Bilinear saddle point problems take the form $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{x}^\top A\boldsymbol{y} - g(\boldsymbol{y})$, where f and g are convex. A popular algorithm that exploits this structure is the

Alternating Direction Method of Multipliers (ADMM) proposed in the 1970s by Glowinski and Marrocco [25] and Gabay and Mercier [22]. ADMM solves problems of the form

$$\begin{array}{cl} \underset{\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{z} \in \mathbb{R}^m}{\text{minimize}} & f(\boldsymbol{x}) + g(\boldsymbol{z}) \\ \text{subject to} & A\boldsymbol{x} + B\boldsymbol{z} = \boldsymbol{c}. \end{array} \tag{4.9}$$

where $A \in \mathbb{R}^{p\times n}$ and $B \in \mathbb{R}^{p\times m}$. Observe that

$$\begin{array}{cl} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} & f(\boldsymbol{x}) \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b}. \end{array} \tag{4.10}$$

is of the form in (4.9), and note the Lagrangian for (4.10) is bilinear:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{x}^\top A\boldsymbol{y} - \boldsymbol{b}^\top \boldsymbol{y}. \tag{4.11}$$

ADMM solves $\max_{\boldsymbol{y}} \min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y})$ iteratively by first minimizing the augmented Lagrangian $\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) + \frac{\rho}{2}\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2$, and then taking a gradient ascent step in $\boldsymbol{y}$:

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{y}_k), \tag{4.12}$$

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \rho\left(A\boldsymbol{x}_{k+1} - \boldsymbol{b}\right), \tag{4.13}$$

where $\rho > 0$ is fixed. The primal dual hybrid gradient algorithm (PDHG), otherwise known as the Chambolle and Pock method, also exploits the bilinear saddle point problem, iterating as follows

$$\boldsymbol{x}_{k+1} = \underset{\boldsymbol{x} \in \mathcal{X}}{\operatorname{argmin}}\, f(\boldsymbol{x}) + \boldsymbol{x}^\top A\boldsymbol{y}_k + \frac{1}{2\eta}\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2, \tag{4.14}$$

$$\boldsymbol{y}_{k+1} = \underset{\boldsymbol{y} \in \mathcal{Y}}{\operatorname{argmin}}\, \boldsymbol{b}^\top \boldsymbol{y} - \boldsymbol{y}^\top A(2\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \frac{1}{2\eta}\|\boldsymbol{y} - \boldsymbol{y}_k\|_2^2. \tag{4.15}$$

Notably, both ADMM and PDHG are a special case of the Douglas-Rachford algorithm [43, 49], and both achieve an ergodic convergence rate of $\mathcal{O}(1/k)$ [32, 13]. In very recent work, Lu and Yang develop a generic algorithm for which PPM, PDHG, and ADMM are all special cases [44]. They then show in a simplified proof to that of the PDHG and ADMM convergence proofs, that this generic algorithm has a $\mathcal{O}(1/k)$ ergodic convergence rate.

4.0.2 Frank-Wolfe for saddle point problems

Researchers have been studying the Frank-Wolfe algorithm (FW) since its discovery in 1956 by Frank and Wolfe [20]. Frank and Wolfe showed in their original paper a worst-case convergence rate of $\mathcal{O}(1/k)$, but it took over a decade for this rate to be proven tight [11]. Within the next twenty years, the original Frank-Wolfe algorithm was generalized to solve problem (3.14) [18, 17, 50]. With this came the invention of variations to the original algorithm, including the Away-Step Frank-Wolfe algorithm [56, 28] and the Fully-Corrective Frank-Wolfe algorithm [33]. FW has gained interest in recent years thanks to the fact that it only performs a linear minimization step at each iteration. The Frank-Wolfe iteration is relatively cheap in comparison to that of the aforementioned algorithms for saddle point problems, which all require the minimization of one or more quadratic functions per iteration.

Until recently, very little work focused on developing algorithms based off of FW for saddle point problems. In her 1984 PhD thesis [30], Hammond developed a generalization of FW for solving the VIP (4.1). Her method converged under the assumptions that $\mathcal{Z}$ is strongly convex[1] and that there exists no point $\boldsymbol{z} \in \mathcal{Z}$ satisfying $F(\boldsymbol{z}) = 0$. Hammond notes that these assumptions are stronger than one might wish, in particular because $\mathcal{Z}$ being strongly convex implies that it cannot be polyhedral. Hammond conjectured that if F is strongly monotone and $\mathcal{Z}$ is a bounded polyhedron, her generalized FW algorithm will solve (4.1). Recent work by Lan [39] proposes to first apply some smoothing scheme to the inner part of the saddle point problem $I(\boldsymbol{x}) := \max_{\boldsymbol{y}\in\mathcal{Y}} f(\boldsymbol{x}, \boldsymbol{y})$, obtaining some $\tilde{I}(\boldsymbol{x})$, and then running FW on $\min_{\boldsymbol{x}\in\mathcal{X}} \tilde{I}(\boldsymbol{x})$. A major downside of this is that projection onto $\mathcal{Y}$ is still required for $\tilde{I}$, while preprocessing the problem is not ideal. A 2017 paper by Gidel et al. [24] made significant headway in applying the Frank-Wolfe algorithm to saddle point problems, partially answering Hammond's conjecture. Gidel et al. extend the Frank-Wolfe algorithm to solve the saddle point problem (3.13), showing convergence for their method over polyhedral sets, but under the assumption of strong convex-concavity of the saddle point objective. The assumption of strong convex-concavity is quite restrictive for our setting however, since it doesn't cover the case where the objective is the Lagrangian of a linear programming problem. It also means Hammond's conjecture is only partially solved. Very recently, Boroun et al. [9] propose a regularized Frank-Wolfe algorithm for nonconvex-concave saddle point problems. However, they make the assumption that the feasible set of the dual is strongly convex, discounting general polyhedra.

For algorithms that solve the linear programming problem without the minimization

[1]Hammond defines a strongly convex set $\mathcal{C}$ as one which for every $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}$ with $\boldsymbol{x} \neq \boldsymbol{y}$ and every $\lambda \in (0, 1)$, there exists an $r > 0$ such that $\boldsymbol{z} \in \mathcal{C}$ whenever $\|\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} - \boldsymbol{z}\|_2 < r$.

of quadratic functions, we defer to the following section.

4.0.3 First order methods for linear programming

The ubiquity of linear programming in modern industrial applications has led to the development of many commercial LP solvers that are able to consistently provide highly accurate solutions. In recent years, data science applications have given rise to increasingly large problems, such that it is not uncommon to see instances of linear programming with billions of variables. State-of-the-art LP solvers typically use variations of the simplex method or interior-point method, however, both methods require solving systems of linear equations and thus do not scale well to very large problems. Over the last few years, many researchers have focused on developing LP algorithms based on first-order methods (FOMs), which are methods whose primary computational cost comes from matrix-vector multiplication. Examples of such algorithms are the following:

- ECLIPSE [4] - a distributed solver for extreme scale problems that applies accelerated gradient descent to the dual of a perturbed version of the original LP.
- PDLP [2, 3] - a specialization of PDHG to linear programs, PDLP applies PDHG to a saddle-point formulation of the LP problem. PDLP is the best known first-order method for linear programming.

These algorithms will be discussed in detail in the next chapter.

Chapter 5

Algorithms for Linear Programming

5.1 The Simplex method

Pioneered by George Dantzig in 1947, the Simplex method was the first algorithm for solving general linear programs. Despite its development being almost 80 years ago, the Simplex method is the most widely used algorithm for linear programming today. The idea of the Simplex method is to start at a basic feasible solution and move to another adjacent BFS if the cost of the adjacent BFS will lead to a better objective value. Eventually, a BFS is reached such that no adjacent BFS leads to a reduction in the objective. In such a case, the basic feasible solution we are at is optimal, and the Simplex method stops. Adjacent, in this setting, means that both basic feasible solutions share all the same basic variables except one. This implies that at each iteration of the method, we are only ever adding one basic variable, and removing one basic variable.

We describe the Simplex method in Algorithm 5.1.

Algorithm 5.1 The Simplex method.

```
Input: A matrix $A \in \mathbb{R}^{m\times n}$ and vectors $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n$, according to (LP). An initial
    basic feasible solution $\boldsymbol{x}$ of (LP) with corresponding basic indices contained in $B$.
 1: for $k = 1, 2, \dots$ do
 2:     For each $i \notin B$, compute $\bar{c}_i = c_i - \boldsymbol{c}_B^\top A_B^{-1} A(:, i)$.
 3:     if $\bar{c}_i \geq 0$ for all $i \in [n]$ then
 4:         The current BFS $\boldsymbol{x}_B$ is optimal.
 5:         break
 6:     else choose entering variable $x_i$:
 7:         Choose some $i$ with $\bar{c}_i < 0$.
 8:     Compute $\boldsymbol{u} = A_B^{-1} A(:, i)$.
 9:     if $u_j \leq 0$ for all $j \in B$ then
10:         The problem is unbounded.
11:         break
12:     else choose leaving variable $x_\ell$:
13:         Choose some $\ell \in \operatorname{argmin}\{\frac{x_j}{u_j} : u_j > 0, j \in B\}$.
14:     Update the set of basic indices: $B = (B \setminus \ell) \cup \{i\}$.
15:     Update the basic feasible solution:
```

$$\boldsymbol{x}_j = \begin{cases} \frac{x_\ell}{u_\ell}, & \text{if } j = i; \\ x_j - \frac{\boldsymbol{x}_\ell}{u_\ell} u_j, & \text{if } j \in B \setminus i; \\ 0, & \text{if } j \notin B. \end{cases} \tag{5.1}$$

The quantity $\bar{c}_i$ is known as the *reduced cost* of the variable x_i. Lines 3-8 of Algorithm 5.1 rely on the following theorem:

Theorem 5.1.1 ([6, Theorem 3.1])**.** Suppose $\boldsymbol{x}$ is a basic feasible solution with basic variables given by the indices in B. If the corresponding vector of reduced costs $\bar{\boldsymbol{c}}$ is nonnegative, then $\boldsymbol{x}$ is optimal.

Remark. For every $i \in B$, we have

$$\bar{c}_i = c_i - \boldsymbol{c}_B^\top A_B^{-1} A(:, i) = c_i - \boldsymbol{c}_B^\top \boldsymbol{e}_i = c_i - c_i = 0,$$

so for all basic variables, their reduced cost is zero.

To derive the reduced cost, we consider moving from the current BFS $\boldsymbol{x}$ to a new BFS $\boldsymbol{x} + \theta\boldsymbol{d}$, where $\theta > 0$. By setting $d_i = 1$ for some $i \notin B$ and $d_j = 0$ for every $j \notin B, j \neq i$, we ensure we are adding one previously non-basic variable to the new basis. We are free to choose d_B however we like, but we must ensure $A(\boldsymbol{x} + \theta\boldsymbol{d}) = b$ to mainain feasibility. Since, we already have that $\boldsymbol{x}$ is feasible, we need to choose the remaining elements of $\boldsymbol{d}$ such that $\boldsymbol{d} \in \operatorname{Null} A$:

$$\begin{aligned}
\mathbf{0} = A\boldsymbol{d} &= \sum_{j=1}^{n} A(:,j)\boldsymbol{d}_j + A(:,i)\boldsymbol{d}_i && (5.2)\\
&= \sum_{j\in B} A(:,i)\boldsymbol{d}_j + A(:,i) && (5.3)\\
&= A_B\boldsymbol{d}_B + A(:,i). && (5.4)
\end{aligned}$$

So, the basic indices of $\boldsymbol{d}$ must be chosen as $\boldsymbol{d}_B = -A_B^{-1}A(:,i)$. The reduced cost is now simply given by

$$\boldsymbol{c}^\top(\boldsymbol{x} + \boldsymbol{d}) - \boldsymbol{c}^\top\boldsymbol{x} = \boldsymbol{c}^\top\boldsymbol{d} = \boldsymbol{c}^\top\boldsymbol{d}_B + \boldsymbol{c}^\top\boldsymbol{d}_i = c_i - \boldsymbol{c}_B^\top A_B^{-1}A(:,i).$$

One should also note that if $\boldsymbol{x}$ is nondegenerate, $\boldsymbol{x}_B > \mathbf{0}$ holds. Hence, the only way the nonnegativity constraints could be violated is if θ is chosen in $\boldsymbol{x}_B + \theta\boldsymbol{d}_B$ to be too large. To get around this, we need only choose a sufficiently small θ. If on the other hand $\boldsymbol{x}$ is degenerate and we have $\boldsymbol{x}_j = 0$ for some $j \in B$, then no value of θ gives a feasible step if $d_j < 0$. It follows that $\boldsymbol{d}$ is not a feasible direction. We will cover this case later.

Lines 9-14 of the algorithm deal with choosing the step-size θ (note $\boldsymbol{u} = -\boldsymbol{d}$). We know from our choice of i on line 8 that the objective strictly decreases as we move in the direction $\boldsymbol{d}$. Hence, we wish to choose θ such that it is as large as possible, while $\boldsymbol{x} + \theta\boldsymbol{d}$ remains feasible for (LP). By our choice of $\boldsymbol{d}$, we know that $A(\boldsymbol{x} + \theta\boldsymbol{d}) = b$ will always hold, so we need only worry about satisfying $\boldsymbol{x} + \theta\boldsymbol{d} \geq 0$. If $\boldsymbol{d} \geq \mathbf{0}$, then we can increase θ to infinity and remain feasible. In this case, (LP) is unbounded (cf. lines 10-12). In the case that there exists a j such that $d_j < 0$ (note, this is only possible for $j \in B$), we must satisfy $\theta \leq -x_j/d_j$, which is equivalent to x_j/u_j as seen on line 14 of Algorithm 5.1.

The final lines cover moving to the new BFS by updating $\boldsymbol{x}$ to be $\boldsymbol{x} + \theta\boldsymbol{d}$. Notably, $x_\ell + \theta d_\ell = x_\ell - \frac{x_\ell}{u_\ell}u_\ell = 0$, i.e. we remove x_ℓ from the basis. On the other hand, $x_i + \theta d_i = \theta d_i = \frac{x_\ell}{u_\ell}$, so x_i enters the basis. The following theorem states that after an iteration of the Simplex method, we are at a basic feasible solution, and by virtue of our choice of which variable to add to the basis, the objective value under the new BFS must be an improvement.

Theorem 5.1.2 ([6, Theorem 3.2])**.** After an iteration of the Simplex method, the columns of the matrix A_B are linearly independent, and the vector $\boldsymbol{x}+\theta\boldsymbol{d}$ is a basic feasible solution with corresponding basis A_B.

If the LP we are dealing with is feasible, then we can always find an initial basic feasible solution. Under the assumption of nondegeneracy, the Simplex method described in Algorithm 5.1 terminates after a finite number of iterations (since there are only finitely many BFS), returning either an optimal basic feasible solution, or a certificate of unboundedness. As mentioned, if the current BFS $\boldsymbol{x}$ is degenerate, $x_j = 0$, and $d_j < 0$, then any value of $\theta > 0$ will give $x+\theta d_j < 0$. In this case, we can continue the Simplex method by remaining at the same BFS, but still update our basis in accordance with line 15 of Algorithm 5.1. Changing basis like this can lead to the discovery of a new BFS, or it can lead back to the same BFS, beginning an indefinite cycle. Cycling can be prevented by following a *pivot rule* in the choice of the entering and leaving variables (cf. lines 8 and 14). For example, the pivot rule known as Bland's rule [7] gives rise to the following theorem.

Theorem 5.1.3 ([6, Theorem 3.3])**.** Assume that (LP) is feasible. The Simplex method applied to (LP) with Bland's rule terminates after a finite number of iterations, returning either a basic feasible solution that is optimal or a certificate of unboundedness.

In Algorithm 5.1 we assumed the existence of an initial basic feasible solution. If we are given a problem with constraints of the form $A\boldsymbol{x} \leq \boldsymbol{b}$, finding an initial BFS is easy; we simply introduce a slack variable $\boldsymbol{s} \geq \boldsymbol{0}$ such that $A\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}$. Our initial BFS is simply $\boldsymbol{s}$ such that $\boldsymbol{s} = \boldsymbol{b}$, obtained by setting $\boldsymbol{x} = \boldsymbol{0}$. If we are not in a case where a similar trick can be applied, we have to resort to using techniques such as the two-phase Simplex method, in which an auxillary linear program must be solved prior to applying the Simplex method to solve the original problem.

The naïve implementation of Algorithm 5.1 incurs a computational cost of $\mathcal{O}(m^3+mn)$ per iteration, with most of the cost coming from solving the systems of equations associated with lines 3 and 9. In practice, algorithms typically employ alternative implementations of the Simplex method such as the revised Simplex method, or the full tableau implementation. Both alternative implementations have a worst-case computational cost of $\mathcal{O}(mn)$ per iteration.

Famously, for most well-known pivot rules, it can be shown that the number of iterations taken by the Simplex method in the worst case is superpolynomial in m and n. In fact, for most pivot rules, including Bland's rule, it can be shown that the number of iterations is at worst exponential. The first example that the Simplex algorithm could run in exponential time is due to Klee and Minty [37]. They showed that if the feasible set of (LP) is chosen to be a perturbed hypercube known as a Klee-Minty cube, then the Simplex method will visit every vertex of the cube before terminating. Since an n-dimensional cube has 2^n vertices, the number of vertices visited in this example by the Simplex method will be 2^n. Since Klee and Minty's example in 1972, there have been many adaptations of the Klee-Minty cube idea to other pivot rules, including Bland's rule.

Despite the poor worst-case performance of the Simplex method, its practical performance is much more favorable, usually terminating after $\mathcal{O}(m+n)$ iterations. Theoretical average case behavior of the Simplex method was first analyzed by Borgwardt in the 1980s. As summarized in [8], on average the Simplex method takes polynomial time. Significant progress toward closing the gap between runtime in practice and in theory was made only recently with Spielman and Teng's development of smoothed analysis for the Simplex method [55]. The authors show that the Simplex method has smoothed complexity that is polynomial.

In practice, solvers can improve the performance of the Simplex method by implementing other techniques, such as applying the Simplex method to the dual linear program (DLP), or exploiting the structure of the given data in another way. It is common for solvers to default to the Simplex method for smaller problems, and then switch to an interior point method for large, sparse problems.

5.2 Interior point methods

Upon the discovery that the Simplex method was indeed not polynomial, researchers began to search for alternatives with the theoretical guarantee of polynomial runtime. The first algorithm for linear programming that became known to run in polynomial time was the Ellipsoid algorithm, initially developed in the 1970's by Shor, Yudin, and Nemirovsky for convex programs [53, 47]. Khachiyan famously applied the Ellipsoid algorithm to linear programs in 1979, showing that LPs could be solved in polynomial time with the Ellipsoid method [36, 23].

Despite its better theoretical performance in comparison to the Simplex method, the Ellipsoid method always runs in time close to its worst-case bound and thus cannot compete with the Simplex method in practice. Despite being impractical, Khachiyan's method opened the door for other polynomial time methods such as Karmarkar's algorithm [35]. While Karmarker's claims of the algorithms' excellent practical performance never materialized, his algorithm gave rise to many new methods including *primal-dual interior-point* *methods*. We will focus on such IPMs because they are the most widely used in practice due to their efficiency in comparison to the alternatives.

Recall the Simplex method iterates by taking steps from vertex to vertex of the polyhedron P until it lands on the optimal vertex. IPMs on the other hand, approach the optimal vertex from either the interior or the exterior of P in the limit. The ideas of interior point methods can be motivated by the difficulty of dealing with the nonnegativity constraint $\boldsymbol{x} \geq \mathbf{0}$ in (LP). To avoid this, IPMs convert the LP into minimizing a linear function with a log-barrier, subject to a linear constraint:

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} - \mu \sum_{i=1}^n \log x_i \\ \text{subject to} \quad & A\boldsymbol{x} = \boldsymbol{b}. \end{aligned} \tag{LP-B}$$

Here, $\mu > 0$ is a parameter and we define the objective to be infinity if $x_i \leq 0$ for any $i \in [n]$. Observe the objective remains convex, since it is a sum of convex functions, so any stationary point corresponds to a global minimum. As $x_i \to 0$, $-\log x_i$ becomes unbounded, preventing any entry of the variable $\boldsymbol{x}$ from ever reaching the boundary. Under the same reasoning, we add a log-barrier term to the dual problem (note we have added the slack variable $\boldsymbol{s} \geq \mathbf{0}$ to enforce equality of the constraint), keeping the objective concave:

$$\begin{aligned} \underset{\boldsymbol{y} \in \mathbb{R}^m}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} + \mu \sum_{i=1}^n \log s_i \\ \text{subject to} \quad & A^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}, \\ & \boldsymbol{s} \geq \mathbf{0}. \end{aligned} \tag{DLP-B}$$

The following are the KKT conditions for (LP-B) and (DLP-B) (cf. [6, Lemma 9.5]):

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{5.5}$$
$$A^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}, \tag{5.6}$$
$$XS\boldsymbol{e} = \boldsymbol{e}\mu, \tag{5.7}$$
$$\boldsymbol{x}, \boldsymbol{s} \geq \mathbf{0}, \tag{5.8}$$

where $X = \mathrm{Diag}(\boldsymbol{x})$ and $S = \mathrm{Diag}(\boldsymbol{s})$. Unfortunately, the system of equations determined by the above KKT conditions is not linear because of (5.7). This makes the system difficult to solve directly and motivates the use of Newton's method in interior point methods to iteratively solve the KKT system. Ignoring the nonnegativity constraints, consider finding a zero of the function $F : \mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m}$ given by

$$F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) = \begin{bmatrix} A\boldsymbol{x} - \boldsymbol{b} \\ A^\top \boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} \\ XS\boldsymbol{e} - \mu\boldsymbol{e} \end{bmatrix} = \boldsymbol{0}. \tag{5.9}$$

Newton's method does this iteratively, so suppose we are at iteration k with feasible solutions given by $(\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{s}_k)$, and parameter μ_k. The Newton update direction is $\boldsymbol{r} = (\boldsymbol{r}_x, \boldsymbol{r}_y, \boldsymbol{r}_s)$, given by solving the system $F(\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{s}_k) = -J(\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{s}_k)\boldsymbol{r}$, where $J(\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{s}_k)$ is the Jacobian of F at the current iterate:

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^\top & I \\ S_k & 0 & X_k \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_x \\ \boldsymbol{r}_y \\ \boldsymbol{r}_s \end{bmatrix} = - \begin{bmatrix} A\boldsymbol{x}_k - \boldsymbol{b} \\ A^\top \boldsymbol{y}_k + \boldsymbol{s}_k - \boldsymbol{c} \\ X_k S_k \boldsymbol{e} - \mu_k \boldsymbol{e} \end{bmatrix} \tag{5.10}$$

After solving (5.10) for the Newton direction $\boldsymbol{r} = (\boldsymbol{r}_x, \boldsymbol{r}_y, \boldsymbol{r}_s)$, iterates are updated as $(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}, \boldsymbol{s}_{k+1}) = (\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{s}_k) + \alpha_k \boldsymbol{r}$ by choosing the step length $\alpha_k > 0$ in such a way to enforce $\boldsymbol{x}_{k+1}, \boldsymbol{s}_{k+1} > 0$. In practice, μ_k is typically chosen to be $\mu_k = \boldsymbol{x}_k^\top \boldsymbol{s}_k / n$. Primal-dual IPMs in practice also tend to modify the system of equations (5.10) to add a *centering parameter* $\sigma_k \in [0, 1]$. When $\sigma_k = 0$, we are performing what is called an *affine scaling* step. When $\sigma_k = 1$, we are taking what is known as a *centering step*. Choosing a σ_k between 0 and 1 allows the practitioner to modulate between an affine scaling step and a centering step. In this case, the Newton direction is found by solving

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^\top & I \\ S_k & 0 & X_k \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_x \\ \boldsymbol{r}_y \\ \boldsymbol{r}_s \end{bmatrix} = - \begin{bmatrix} A\boldsymbol{x}_k - \boldsymbol{b} \\ A^\top \boldsymbol{y}_k + \boldsymbol{s}_k - \boldsymbol{c} \\ X_k S_k \boldsymbol{e} - \sigma_k \mu_k \boldsymbol{e} \end{bmatrix}. \tag{5.11}$$

Regardless, at each iteration primal-dual IPMs solve a $(2n + m) \times (2n + m)$ system of equations, incurring a computational cost of $\mathcal{O}(n^3)$ per iteration. We can set $D_k^2 = S_k^{-1} X_k$ and write (5.11) as the system of equations

$$AD_k^2 A^\top \boldsymbol{r}_y = \boldsymbol{c} - A^\top \boldsymbol{y}_k - \boldsymbol{s}_k - AD_k^2 (A\boldsymbol{x}_k - \boldsymbol{b} - X_k^{-1}(X_k S_k - \sigma_k \mu_k)\boldsymbol{e}) \tag{5.12}$$

$$\boldsymbol{r}_s = \boldsymbol{b} - A\boldsymbol{x}_k - A^\top \boldsymbol{r}_y \tag{5.13}$$

$$\boldsymbol{r}_x = -S_k^{-1}(X_k(S_k \boldsymbol{e} + \boldsymbol{r}_s) - \sigma_k \mu_k \boldsymbol{e}), \tag{5.14}$$

recalling that X_k and S_k are diagonal matrices. Writing (5.12) as $AD_k^2 A^\top \boldsymbol{r}_y = \boldsymbol{q}_k$, we can perform a Cholesky decomposition of the symmetric positive definite matrix $AD_k^2 A^\top$, obtaining a square lower-triangular matrix L_k such that $AD_k^2 A^\top = L_k L_k^\top$. Now the Newton directions are obtained by first solving $L_k \boldsymbol{v} = \boldsymbol{q}$ for $\boldsymbol{v}$ and then solving $L_k^\top \boldsymbol{r}_y = \boldsymbol{q}$ for $\boldsymbol{r}_y$.

Equations (5.13) and (5.14) follow immediately without the need for solving a system of equations. The benefit of solving the Newton equations this way is that the systems $L_k \boldsymbol{v} = \boldsymbol{q}$ and $L_k^\top \boldsymbol{r}_y = \boldsymbol{q}$ can be solved in $\mathcal{O}(n^2)$ operations, hence the whole system has computational complexity $\mathcal{O}(n^2)$. The caveat is that one still needs to form the Cholesky decomposition at a cost of $\mathcal{O}(n^3)$, however the practical performance of computing Cholesky factors L and then solving the above system is better than solving directly, particularly when A is large and sparse. For this reason, primal-dual IPMs are usually preferred over the Simplex method for solving large linear programs.

A general primal-dual interior point method is given in Algorithm 5.2 below.

Algorithm 5.2 A primal-dual interior-point method.

Input: A matrix $A \in \mathbb{R}^{m\times n}$ and vectors $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n$, according to (LP). Initial points $(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{s}_1)$ such that $\boldsymbol{x}_1, \boldsymbol{s}_1 > \boldsymbol{0}$.

```
1: for $k = 1, 2, \dots$ do
2:     Choose some $\sigma_k \in (0, 1]$ and set $\mu_k = \boldsymbol{x}_k^\top \boldsymbol{s}_k / n$.
3:     Solve the system of equations (5.11) for $\boldsymbol{r} = (\boldsymbol{r}_x, \boldsymbol{r}_y, \boldsymbol{r}_s)$:
           $\begin{bmatrix} A & 0 & 0 \\ 0 & A^\top & I \\ S_k & 0 & X_k \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_x \\ \boldsymbol{r}_y \\ \boldsymbol{r}_s \end{bmatrix} = - \begin{bmatrix} A\boldsymbol{x}_k - \boldsymbol{b} \\ A^\top \boldsymbol{y}_k + \boldsymbol{s}_k - \boldsymbol{c} \\ X_k S_k \boldsymbol{e} - \sigma_k \mu_k \boldsymbol{e} \end{bmatrix}$
4:     Take a step in the Newton direction, choosing $\alpha_k > 0$ to ensure $\boldsymbol{x}_{k+1}, \boldsymbol{s}_{k+1} > 0$:
           $(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}, \boldsymbol{s}_{k+1}) = (\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{s}_k) + \alpha_k \boldsymbol{r}.$
```

Notice that as input to Algorithm 5.2, we do not require the initial points $(\boldsymbol{x}_1, \boldsymbol{y}_1, \boldsymbol{s}_1)$ to be feasible; our only requirement is that $\boldsymbol{x}_1, \boldsymbol{s}_1 > 0$. Primal-dual IPMs started with infeasible points are known as *infeasible methods* and are used widely in practice because of their observed performance benefits, while still being convergent in theory [57, Chapter 6].

As alluded to, the runtime of Algorithm 5.2 is polynomial. If we set $\epsilon_1 := \boldsymbol{s}_1^\top \boldsymbol{x}_1$ to be the initial duality gap, it takes a primal-dual interior point method $\mathcal{O}(\sqrt{n} \log \frac{\epsilon_1}{\epsilon})$ iterations to bring the duality gap down from ϵ_1 to ϵ.

5.3 First order algorithms

5.3.1 ECLIPSE [4]

ECLIPSE, which stands for Extreme Scale Linear Program Solver, considers the linear programming problem

$$\begin{array}{ll} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \le \boldsymbol{b}, \\ & x_i \in \mathcal{C}_i, \;\; \forall i \in [I], \end{array} \tag{5.15}$$

where I will be defined later and $\mathcal{C}_i$ are simple in that projection onto them is efficient. By noting that the standard form polyhedron P defined earlier is equivalent to $\{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \le \boldsymbol{b}\}$, (LP) can be equivalently written as $\min\{\boldsymbol{c}^\top \boldsymbol{x} : A\boldsymbol{x} \le \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n\}$. Dropping the $x_i \in \mathcal{C}_i$ constraint in (5.15) gives an LP equivalent to the standard form problem (LP). We will see later that in the case where we are dealing with a standard form LP with no additional constraints, it will be necessary to introduce redundant constraints. For the remainder of the section on ECLIPSE, we will consider the problem (5.15), but the reader should keep in mind the aforestated equivalence.

ECLIPSE assumes the following structure on (5.15):

- Variable $\boldsymbol{x}$ is the concatenation of I vectors: $\boldsymbol{x} = (\boldsymbol{x}_1; \ldots; \boldsymbol{x}_I)$, where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_I \in \mathbb{R}^J$.
- $n = IJ$ ranges from 100s of millions to 10s of trillions or larger.
- A is sparse of the form $(A_1; A_2)$, where $A_1 \in \mathbb{R}^{m_1 \times n}$ contains no zeros with $m_1 = \mathcal{O}(1) \ll n$, and

  $$A_2 = \begin{bmatrix} D_{11} & \ldots & D_{1I} \\ \vdots & \ddots & \vdots \\ D_{m_2 1} & \ldots & D_{m_2 I} \end{bmatrix}$$

  where D_{ij} are $J \times J$ diagonal matrices.
- The diagonal of each D_{ij} is sparse, with a maximum of K non-zero entries such that $K < J$. It follows that the total number of non-zero entries in A is $m_1 IJ + m_2 IK$.
- $\mathcal{C}_i$ are simple in that projection onto them can be computed efficiently.

The authors claim that several important problems arising in web-applications satisfy the above assumptions. However, these assumptions are not necessary for the ECLIPSE algorithm (described below) from a theoretical standpoint.

```
Algorithm 5.3 ECLIPSE: Extreme Scale Linear Program Solver.
Input: A matrix $A \in \mathbb{R}^{m\times n}$, vectors $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n$, scalar $\gamma \in \mathbb{R}$, and constraints
    $\{\mathcal{C}_i\}_{i=1}^I$. Initial point $\boldsymbol{y}_1 \in \mathbb{R}^m$ and step-size $\eta \in (0, 1/L]$, where $L = \|A\|_2^2/\gamma$.
1: for $k = 1, 2, \ldots$ do
2:     Perform the primal update:
           $$\hat{x}_i(\boldsymbol{y}_k) = P_{\mathcal{C}_i}\left(-\frac{1}{\gamma}\left[A^\top \boldsymbol{y}_k + \boldsymbol{c}\right]_i\right), \quad \forall i \in [n] \tag{5.16}$$
3:     Compute the dual gradient:
           $$\nabla g(\boldsymbol{y}_k) := A\hat{\boldsymbol{x}}(\boldsymbol{y}_k) - \boldsymbol{b} \tag{5.17}$$
4:     Perform projected gradient ascent on the dual variable $\boldsymbol{y}$:
           $$\boldsymbol{y}_{k+1} = [\boldsymbol{y}_k + \eta\nabla g(\boldsymbol{y}_k)]_+ \tag{5.18}$$
```

The authors derive Algorithm 5.3 by perturbing (5.15) by a quadratic term:

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} + \frac{\gamma}{2}\boldsymbol{x}^\top \boldsymbol{x} \\ \text{subject to} \quad & A\boldsymbol{x} \le \boldsymbol{b}, \\ & x_i \in \mathcal{C}_i, \ \ \forall i \in [I], \end{aligned} \tag{5.19}$$

where $\gamma > 0$ is a parameter whose value determines how close the optimal objective of (5.19) is to that of the LP (5.15), and the smoothness of the dual objective. In its current stage, (5.19) cannot be solved by a first-order method. To rectify this, the authors consider the Lagrangian, bringing the constraint $A\boldsymbol{x} \le \boldsymbol{b}$ into the objective, while keeping the simple constraints $\mathcal{C}_i$:

$$\begin{aligned} g(\boldsymbol{y}) := \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} \quad & \boldsymbol{c}^\top \boldsymbol{x} + \frac{\gamma}{2}\boldsymbol{x}^\top \boldsymbol{x} + \boldsymbol{y}^\top(A\boldsymbol{x} - \boldsymbol{b}) \\ \text{subject to} \quad & x_i \in \mathcal{C}_i, \ \ \forall i \in [I]. \end{aligned} \tag{5.20}$$

For dual solution $\boldsymbol{y}$, we denote the corresponding primal variable that minimizes (5.20) by $\hat{\boldsymbol{x}}(\boldsymbol{y})$. It is not hard to see that the closed form solution for $\hat{\boldsymbol{x}}(\boldsymbol{y})$ is

$$\hat{x}_i(\boldsymbol{y}_k) = P_{\mathcal{C}_i}\left(-\frac{1}{\gamma}\left[A^\top \boldsymbol{y}_k + \boldsymbol{c}\right]_i\right), \quad \forall i \in [n].$$

The following lemma connects the parameter γ with the smoothness of the dual objective from (5.20):

Lemma 5.3.1 ([4])**.** *$g(\boldsymbol{y})$ is differentiable with $\nabla g(\boldsymbol{y}) = A\hat{x}(\boldsymbol{y}) - \boldsymbol{b}$ and $\nabla g(\boldsymbol{y})$ is Lipschitz continuous with constant $L = \|A\|_2^2/\gamma$.*

Having an L-smooth dual objective with $L > 0$ allows the authors to apply projected gradient ascent as in step 4 of Algorithm 5.3. Define $g_\gamma(\boldsymbol{y}) := \min_{\boldsymbol{x}\in\mathcal{C}}\{\boldsymbol{c}^\top\boldsymbol{x} + \frac{\gamma}{2}\boldsymbol{x}^\top\boldsymbol{x} + \boldsymbol{y}^\top(A\boldsymbol{x} - \boldsymbol{b})\}$. If we set $\gamma = 0$, $g_\gamma^* := \max_{\boldsymbol{y}\geq\boldsymbol{0}} g_\gamma(\boldsymbol{y})$ is equivalent to the dual of (5.15). Unfortunately, without the quadratic perturbation $\frac{\gamma}{2}\boldsymbol{x}^\top\boldsymbol{x}$, we would have $L = 0$ and would not be able to apply first-order methods such as projected gradient ascent. Lemma 5.3.1 suggests choosing too small a γ is detrimental to the smoothness of $\nabla g_\gamma(\boldsymbol{y})$. On the other hand, the following lemma suggests that we also don't want to choose γ too large.

Lemma 5.3.2 ([4])**.** *For all $\boldsymbol{y} \in \mathbb{R}^m$, the dual objective $g_\gamma(\boldsymbol{y})$ satisfies the uniform bound*

$$g_\gamma(\boldsymbol{y}) - \gamma\nu/2 - g_0(\boldsymbol{y}) \leq g_\gamma(\boldsymbol{y}),$$

where $\nu = \max\{\boldsymbol{x}^\top\boldsymbol{x} : \boldsymbol{x} \in \mathcal{C}\}$.

The implication of the above is that the optimal value of (5.15) lies in the interval $[g_\gamma^* - \gamma\nu/2, g_\gamma^*]$. The dependence on ν in the error bound above is problematic. It suggests that one must introduce a constraint $\mathcal{C}$ that ensures $\nu < +\infty$, and preferably also ensures ν is not too large. For solving (LP) where $\max\{\boldsymbol{x}^\top\boldsymbol{x} : \boldsymbol{x} \in \mathcal{C}\}$ would be unbounded, one would have to introduce a redundant constraint much like the $\boldsymbol{e}^\top\boldsymbol{x} \leq \xi$ constraint used in Algorithm 6.1 of Chapter 6. Importantly, such a constraint should not be too restrictive that it prevents the algorithm from finding an accurate primal solution.

Most of the computational complexity of the ECLIPSE algorithm comes from the gradient computation (5.17), which performs a matrix-vector product at each iteration. This assumes that the projection done in (5.16) is simple enough to not exceed the computational effort of the matrix-vector product. However, such low cost iterations come with the sacrifice of accuracy. ECLIPSE is only able to compute approximate dual solutions (cf. Lemma 5.3.2), from which an approximate primal solution can be constructed per (5.16). Moreover, there exists no convergence analysis of Algorithm 5.3.

5.3.2 PDLP [2, 3]

The PDLP algorithm is a specialization of the previously stated PDHG algorithm to linear programs. PDLP solves for the primal and dual solutions of the standard form problem (LP) by considering the Lagrangian[1]

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} \mathcal{L}(\boldsymbol{x},\boldsymbol{y}) := \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{y}^\top(\boldsymbol{b} - A\boldsymbol{x}), \tag{5.21}$$

where $\mathcal{X} := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \geq \boldsymbol{0}\}$ and $\mathcal{Y} := \mathbb{R}^m$. A basic version of the PDHG algorithm applied to (5.21) is the following.

Algorithm 5.4 PDHG for (LP).

```
Input: A matrix $A \in \mathbb{R}^{m\times n}$ and vectors $\boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n$, according to (LP). Initial points
    $(\boldsymbol{x}_1, \boldsymbol{y}_1) \in \mathbb{R}^n \times \mathbb{R}^m$, and step-size $\eta > 0$.
1: for $k = 1, 2, \ldots$ do
2:     Perform the primal update:
           $$\boldsymbol{x}_{k+1} = \left[\boldsymbol{x}_k - \eta(\boldsymbol{c} - A^\top \boldsymbol{y}_k)\right]_+ . \tag{5.22}$$
3:     Using the new primal value, perform the dual update:
           $$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \eta(\boldsymbol{b} - A(2\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)). \tag{5.23}$$
```

Notably, Algorithm 5.4 spends most of its computational effort on matrix-vector multiplications in both the primal and dual updates. PDHG, and thus Algorithm 5.4, is known to converge to an optimal primal-dual pair when $\|A\|_2 \leq 1/\eta$, at the aforementioned rate of $\mathcal{O}(1/k)$ [2, 12, 15].

The authors in [3] note that in practice the most widely used algorithms for solving (LP) are the Simplex method and interior-point methods. The reason being that both algorithms are able to reliably find accurate solutions in reasonable time. As seen in Sections 5.1 and 5.2, both the Simplex method and interior point methods rely on solving systems of linear equations exactly, which can at times be slow for dense systems, or for large enough problems, can result in memory usage that exceeds the capabilities of the hardware being used.

[1]We can identify (5.21) with the bilinear saddle point form of (4.10) in Chapter 4 by setting $A := -A$ and $g(\boldsymbol{y}) = -\boldsymbol{b}^\top \boldsymbol{y}$. Since $\boldsymbol{y} \mapsto \boldsymbol{b}^\top \boldsymbol{y}$ is linear, it's sign doesn't affect its convexity.

Such issues motivate the authors search in [2, 3] for a linear programming algorithm that requires only matrix-vector multiplications. In doing so, they develop PDLP, which builds upon Algorithm 5.4 by modifying the step-sizes and adding restarts. Before running the algorithm, PDLP also presolves the input problem and performs diagonal preconditioning on the constraint matrix A.

Through an ablation study in [2], the authors show that each modification added on top of PDHG results in an improvement numerically. Together, the modifications significantly reduce the number of iterations required to reach a given stopping criterion on real-world test problems compared to Algorithm 5.4. Despite this, there are no results guaranteeing that the modifications made to Algorithm 5.4 improve the algorithm in theory. The authors note in [2] that the presolve and diagonal preconditioning steps preserve the theoretical guarantees of PDHG applied to (LP), and in [3] it is shown that restarting Algorithm 5.4 also preserves its theoretical guarantees.

Restarting PDLP

While we have defined Algorithm 5.4 above using the same step-sizes for both the primal and the dual, PDLP defines a primal step-size $\tau = \eta/\omega$ and a dual step-size $\sigma = \omega\eta$, where $\eta > 0$ and $\omega > 0$. Under these definitions, PDLP still converges for all $\|A\|_2 \le 1/\eta$ [2]. The authors call ω the *primal weight* and define the following norm:

$$\|(\boldsymbol{x}, \boldsymbol{y})\|_\omega := \sqrt{\omega\|\boldsymbol{x}\|_2^2 + \frac{\|\boldsymbol{y}\|_2^2}{\omega}}. \tag{5.24}$$

In [3], the authors define the *normalized duality gap*. The normalized duality gap at restart phase t for point $(\boldsymbol{x}, \boldsymbol{y})$ and radius $r > 0$ is defined as

$$\rho_r^t(\boldsymbol{x}, \boldsymbol{y}) := \frac{1}{r} \max\left\{\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{y}}) - \mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{y}) : \|(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) - (\boldsymbol{x}, \boldsymbol{y})\|_{\omega_t} \le r, \boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}\right\}. \tag{5.25}$$

The authors note in [3] that (5.25) is always finite and is computable in linear time. This is in contrast to the standard primal-dual gap

$$\rho(\boldsymbol{x}, \boldsymbol{y}) := \max\left\{\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{y}}) - \mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{y}), \boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}\right\}, \tag{5.26}$$

which is not finite on the sets $\mathcal{X}$ and $\mathcal{Y}$. For any value of $r, \omega_t > 0$, $\rho_r^t(\boldsymbol{x}, \boldsymbol{y}) = 0$ if and only if $(\boldsymbol{x}, \boldsymbol{y})$ is an optimal solution to (5.21). Hence, (5.25) is a valid convergence metric.

Let us use the notation $(\boldsymbol{x}_{t,k}, \boldsymbol{y}_{t,k})$ to denote the iterates of the kth iteration of the tth restart phase. To choose the candidate iterate to use after the restart, denoted $(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1})$, PDLP considers the cases

$$(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) = \begin{cases} (\boldsymbol{x}_{t,k+1}, \boldsymbol{y}_{t,k+1}), & \mu_{t,k+1} < \bar{\mu}_{t,k+1}; \\ (\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1}), & \text{otherwise}; \end{cases} \tag{5.27}$$

where $\mu_{t,k+1} = \rho^t_r(\boldsymbol{x}_{t,k+1}, \boldsymbol{y}_{t,k+1})$ with

$$r = \|(\boldsymbol{x}_{t,k+1}, \boldsymbol{y}_{t,k+1}) - (\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1})\|_{\omega_t},$$

$\bar{\mu}_{t,k+1} = \rho^t_r(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1})$ with

$$r = \|(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1}) - (\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1})\|_{\omega_t},$$

and $(\bar{x}_{t,k+1}, \bar{y}_{t,k+1})$ are the averaged iterates, given by

$$(\bar{x}_{t,k+1}, \bar{y}_{t,k+1}) = \frac{1}{\sum_{i=1}^{k+1} \eta_{t,i}} \sum_{i=1}^{k+1} \eta_{t,i} z_{t,i}.$$

Below are two of the restart criteria PDLP uses related to our work. PDLP will restart if one of the following are satisfied, where $\beta_{\text{suff}} \in (0,1)$ and $\beta_{\text{nec}} \in (0, \beta_{\text{suff}})$.

- **Sufficient decay in normalized duality gap:**

$$\rho^t_{\|(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) - (\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1})\|_{\omega_t}}(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) \le \beta_{\text{suff}} \rho^t_{\|(\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1}) - (\boldsymbol{x}_{t-1,1}, \boldsymbol{y}_{t-1,1})\|_{\omega_t}}(\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1}).$$

- **Necessary decay and no local progress in normalized duality gap:**

$$\rho^t_{\|(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) - (\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1})\|_{\omega_t}}(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) \le \beta_{\text{nec}} \rho^t_{\|(\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1}) - (\boldsymbol{x}_{t-1,1}, \boldsymbol{y}_{t-1,1})\|_{\omega_t}}(\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1}),$$

  and

$$\rho^t_{\|(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) - (\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1})\|_{\omega_t}}(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) > \rho^t_{\|(\boldsymbol{x}^c_{t,k}, \boldsymbol{y}^c_{t,k}) - (\boldsymbol{x}_{t,1}, \boldsymbol{y}_{t,1})\|_{\omega_t}}(\boldsymbol{x}^c_{t,k}, \boldsymbol{y}^c_{t,k}).$$

The PDLP algorithm uses the values $\beta_{\text{suff}} = 0.9$ and $\beta_{\text{nec}} = 0.1$.

We will discuss the above restart criteria further in Section 6.5, where they will be applied to the FWLP algorithm (Algorithm 6.1).

Chapter 6

Solving Saddle Point Formulations of Linear Programs with Frank-Wolfe

Throughout this chapter, we will assume that optimal solutions exist for both (LP) and (DLP), and that for each $i \in [n]$, $A\boldsymbol{e}_i \neq \boldsymbol{0}$.

FWLP (Algorithm 6.1) is derived from iteratively applying the Frank-Wolfe algorithm to the primal and dual problems of a modified saddle point formulation of (LP):

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{y}\in\mathcal{Y}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{y}^\top (\boldsymbol{b} - A\boldsymbol{x}), \tag{6.1}$$

where we define $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{e}^\top \boldsymbol{x} \leq \xi\}$ and $\mathcal{Y} = \{\boldsymbol{y} \in \mathbb{R}^m : \|\boldsymbol{y}\|_\infty \leq \eta\}$ for parameters $\xi, \eta > 0$.

Let $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ denote the optimal solutions to (LP) and (DLP). We will assume that ξ and η are chosen large enough to ensure that $\xi > \boldsymbol{e}^\top \boldsymbol{x}^*$ and $\eta > \|\boldsymbol{y}^*\|_\infty$. That is, the parameters ξ and η describe redundant constraints for (MLP) and (MDP) below. Despite being redundant constraints, we will see that the constraints defined by $\boldsymbol{e}^\top \boldsymbol{x} \leq \xi$ and $\|\boldsymbol{y}\|_\infty \leq \eta$ are necessary for boundedness of the Frank-Wolfe subproblems in FWLP.

FWLP amounts to simultaneously solving the modified linear programs

$$\begin{array}{ll} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{minimize}} & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{e}^\top \boldsymbol{x} \leq \xi, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \tag{MLP}$$

and

$$\begin{aligned} \underset{\boldsymbol{y}\in\mathbb{R}^m}{\text{maximize}} \quad & \boldsymbol{b}^\top \boldsymbol{y} \\ \text{subject to} \quad & A^\top \boldsymbol{y} \le \boldsymbol{c}, \\ & \boldsymbol{y} \le \eta \boldsymbol{e}, \\ & \boldsymbol{y} \ge -\eta \boldsymbol{e}. \end{aligned} \tag{MDP}$$

It is important to note that (MLP) and (MDP) are not a primal-dual pair. At each iteration k, we take steps to minimize $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}_k)$ with respect to $\boldsymbol{x}$ and maximize $\mathcal{L}(\boldsymbol{x}_{k+1}, \boldsymbol{y})$ with respect to $\boldsymbol{y}$ in the hope of converging to a saddle point of $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y})$. We will argue later that a saddle point is equivalent to an optimal solution for (MLP) and (MDP), as well as (LP) and (DLP), under the assumption that the constraints described by ξ and η are redundant.

The primal Frank-Wolfe update is

$$\boldsymbol{s} := \underset{\boldsymbol{s}\in\mathcal{X}}{\operatorname{argmin}}\{\boldsymbol{c}^\top \boldsymbol{x}_k + \boldsymbol{y}_k^\top(\boldsymbol{b} - A\boldsymbol{x}_k) + \boldsymbol{s}^\top(\boldsymbol{c} - A^\top \boldsymbol{y}_k)\}$$
$$\boldsymbol{x}_{k+1} = \frac{k}{k+1}\boldsymbol{x}_k + \frac{\boldsymbol{s}}{k+1},$$

which simplifies to

$$\boldsymbol{s} := \underset{\boldsymbol{s}\in\mathcal{X}}{\operatorname{argmin}}\, \boldsymbol{s}^\top(\boldsymbol{c} - A^\top \boldsymbol{y}_k), \tag{6.2}$$
$$\boldsymbol{x}_{k+1} = \frac{k}{k+1}\boldsymbol{x}_k + \frac{\boldsymbol{s}}{k+1}. \tag{6.3}$$

Observe that the constraint $\boldsymbol{e}^\top \boldsymbol{x} \le \xi$ in the definition of $\mathcal{X}$ is necessary to prevent the problem (6.2) from being unbounded. We will dicuss how to choose ξ in Section 6.4.

Let us analyze (6.2) by considering the minimum entry of the second term in the inner product. Say $i = \operatorname{argmin}_t[\boldsymbol{c} - A^\top \boldsymbol{y}_k]_t$. If $[\boldsymbol{c} - A^\top \boldsymbol{y}_k]_i \ge 0$, we know that no matter what we choose for $\boldsymbol{s} \in \mathcal{X}$, (6.2) will be nonnegative. Hence, we should take $\boldsymbol{s} = \boldsymbol{0}$. If on the other hand $[\boldsymbol{c} - A^\top \boldsymbol{y}_k]_i < 0$, we should put all our resources towards multiplying by the most negative entry, i.e. the ith entry. Thus, we take $\boldsymbol{s} = \xi \boldsymbol{e}_i$.

Now that we have updated the primal variable, we use this new information in the dual Frank-Wolfe update:

$$\boldsymbol{u} := \underset{\boldsymbol{u}\in\mathcal{Y}}{\operatorname{argmax}}\{\boldsymbol{c}^\top \boldsymbol{x}_{k+1} + \boldsymbol{y}_k^\top(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) + \boldsymbol{u}^\top(\boldsymbol{b} - A\boldsymbol{x}_{k+1})\},$$
$$\boldsymbol{y}_{k+1} = \frac{k}{k+1}\boldsymbol{y}_k + \frac{\boldsymbol{u}}{k+1}.$$

which simplifies to

$$\boldsymbol{u} := \operatorname*{argmax}_{\boldsymbol{u}\in\mathcal{Y}} \boldsymbol{u}^\top(\boldsymbol{b} - A\boldsymbol{x}_{k+1}), \tag{6.4}$$

$$\boldsymbol{y}_{k+1} = \frac{k}{k+1}\boldsymbol{y}_k + \frac{\boldsymbol{u}}{k+1}. \tag{6.5}$$

As with the primal case, we can analyze (6.4) according to the sign pattern of $\boldsymbol{b} - A\boldsymbol{x}_{k+1}$ and determine that the inner product is maximized by choosing $\boldsymbol{u} = \eta\,\mathrm{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1})$. Like in the primal case, the constraint $\|\boldsymbol{y}_k\|_\infty \le \eta$ is important for the problem (6.4) to have a solution.

The above analysis leads to the closed form updates described in equations (6.55), (6.56), and (6.57) of Algorithm 6.1. Of note here is the low cost nature of the updates. Most of the computational cost of FWLP comes from computing two matrix-vector products: $A^\top\boldsymbol{y}_k$ and $A\boldsymbol{x}_{k+1}$.

Algorithm 6.1 FWLP: A primal-dual algorithm for (LP) based on Frank-Wolfe.

Input: Starting points $\boldsymbol{x}_1 \in \mathbb{R}^n, \boldsymbol{y}_1 \in \mathbb{R}^m$, constraint data A and $\boldsymbol{b}$, and objective $\boldsymbol{c}$.

Parameters: $\xi > 0$ such that $\boldsymbol{e}^\top\boldsymbol{x}_k \le \xi$, $\eta > 0$ such that $\|\boldsymbol{y}_k\|_\infty \le \eta$.

```
1: for $k = 1, 2, \ldots$ do
2:     Determine $i = \mathrm{argmin}_t[\boldsymbol{c} - A^\top\boldsymbol{y}_k]_t$.
3:     if $[\boldsymbol{c} - A^\top\boldsymbol{y}_k]_i \ge 0$ then
4:         Step towards zero in $\boldsymbol{x}$:
               $$\boldsymbol{x}_{k+1} = \frac{k}{k+1}\boldsymbol{x}_k. \tag{6.6}$$
5:     else
6:         Step toward $\xi$ for $x_i$, otherwise step toward zero for $x_j, j \ne i$:
               $$\boldsymbol{x}_{k+1} = \frac{k}{k+1}\boldsymbol{x}_k + \frac{\xi}{k+1}\boldsymbol{e}_i. \tag{6.7}$$
7:     Step towards $\pm\eta$ for each coordinate in $y$ according to the sign pattern of $\boldsymbol{b} - A\boldsymbol{x}_{k+1}$.
               $$\boldsymbol{y}_{k+1} = \frac{k}{k+1}\boldsymbol{y}_k + \frac{\eta}{k+1}\mathrm{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}). \tag{6.8}$$
```

6.1 Theoretical results

FWLP works by choosing the primal variable corresponding to the most infeasible dual constraint x_i, and taking a Frank-Wolfe step towards ξ in x_i. Our first result considers the case where we are dual infeasible and shows that Algorithm 6.1 cannot select the same entry of $\boldsymbol{x}$ consecutively forever. We begin with a useful Lemma:

Lemma 6.1.1. Let $A \in \mathbb{R}^{m \times n}$ and $\boldsymbol{w}, \boldsymbol{\Delta} \in \mathbb{R}^n$. Then

$$\boldsymbol{w}^\top A^\top \operatorname{sgn}(A(\boldsymbol{w}+\boldsymbol{\Delta})) = \|A(\boldsymbol{w}+\boldsymbol{\Delta})\|_1 - \boldsymbol{\Delta}^\top A^\top \operatorname{sgn}(A(\boldsymbol{w}+\boldsymbol{\Delta})).$$

Proof. We may equivalently write the left hand side as

$$\begin{aligned}\boldsymbol{w}^\top A^\top \operatorname{sgn}(A(\boldsymbol{w}+\boldsymbol{\Delta})) &= (\boldsymbol{w}+\boldsymbol{\Delta})^\top A^\top \operatorname{sgn}(A(\boldsymbol{w}+\boldsymbol{\Delta})) - \boldsymbol{\Delta}^\top A^\top \operatorname{sgn}(A(\boldsymbol{w}+\boldsymbol{\Delta}))\\ &= \|A(\boldsymbol{w}+\boldsymbol{\Delta})\|_1 - \boldsymbol{\Delta}^\top A^\top \operatorname{sgn}(A(\boldsymbol{w}+\boldsymbol{\Delta})).\end{aligned}$$

The second equality follows because $(\boldsymbol{w}+\boldsymbol{\Delta})^\top A^\top \operatorname{sgn}(A(\boldsymbol{w}+\boldsymbol{\Delta}))$ adds up the entries of $(\boldsymbol{w}+\boldsymbol{\Delta})^\top A^\top$ scaled by their signs. □

Theorem 6.1.2. In Algorithm 6.1, the number of consecutive iterations such that the same $i \in [n]$ is chosen as the solution to $\operatorname{argmin}_t[\boldsymbol{c} - A^\top \boldsymbol{y}_k]_t$, while also satisfying $[\boldsymbol{c} - A^\top \boldsymbol{y}_k]_i < 0$, is finite.

Proof. Let us assume that the same i attains the min on all iterations k_1 and after: $k_1, k_1 + 1, \ldots, k_1 + \ell$. We want to show a contradiction if ℓ is sufficiently large. We have the recurrence

$$\boldsymbol{x}_{k_1+1} = \frac{k_1}{k_1+1}\boldsymbol{x}_{k_1} + \frac{1}{k_1+1}\xi \boldsymbol{e}_i,$$

and

$$\begin{aligned}\boldsymbol{x}_{k_1+2} &= \frac{k_1+1}{k_1+2}\left(\frac{k_1}{k_1+1}\boldsymbol{x}_{k_1} + \frac{1}{k_1+1}\xi \boldsymbol{e}_i\right) + \frac{1}{k_1+2}\xi \boldsymbol{e}_i\\ &= \frac{k_1}{k_1+2}\boldsymbol{x}_{k_1} + \frac{2}{k_1+2}\xi \boldsymbol{e}_i.\end{aligned}$$

It is easy to see inductively that

$$\boldsymbol{x}_{k_1+\ell} = \frac{k_1}{k_1+\ell}\boldsymbol{x}_{k_1} + \frac{\ell}{k_1+\ell}\xi \boldsymbol{e}_i.$$

This implies that

$$\boldsymbol{b} - A\boldsymbol{x}_{k_1+\ell} = \frac{k_1}{k_1+\ell}(\boldsymbol{b} - A\boldsymbol{x}_{k_1}) - \frac{\ell}{k_1+\ell}\xi A\boldsymbol{e}_i + \frac{\ell}{k_1+\ell}\boldsymbol{b}. \tag{6.9}$$

Let $\boldsymbol{\sigma}_k$ be used to denote $\operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_k)$ (an m-vector of ± 1's). It then follows from Algorithm 6.1 that

$$\boldsymbol{y}_{k_1+\ell} = \frac{k_1}{k_1+\ell}\boldsymbol{y}_{k_1} + \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\boldsymbol{\sigma}_{k_1+j}.$$

Therefore,

$$\begin{aligned}
\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1+\ell} &= \frac{k_1}{k_1+\ell}(\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1}) - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}A^\top\boldsymbol{\sigma}_{k_1+j} + \frac{\ell}{k_1+\ell}\boldsymbol{c} \\
&= \frac{k_1}{k_1+\ell}(\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1}) - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}A^\top\operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k_1+j}) + \frac{\ell}{k_1+\ell}\boldsymbol{c} \\
&= \frac{k_1}{k_1+\ell}(\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1}) \\
&\qquad - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}A^\top\operatorname{sgn}\left(\frac{k_1}{k_1+j}(\boldsymbol{b} - A\boldsymbol{x}_{k_1}) - \frac{j}{k_1+j}\xi A\boldsymbol{e}_i + \frac{j}{k_1+j}\boldsymbol{b}\right) \\
&\qquad + \frac{\ell}{k_1+\ell}\boldsymbol{c}.
\end{aligned}$$

Let $\boldsymbol{z}_0 := \boldsymbol{x}^* - \boldsymbol{x}_{k_1}$, where $\boldsymbol{x}^*$ is the primal optimizer. Then $A\boldsymbol{z}_0 = \boldsymbol{b} - A\boldsymbol{x}_{k_1}$ so we have

$$\begin{aligned}
\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1+\ell} &= \frac{k_1}{k_1+\ell}(\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1}) \\
&\qquad - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}A^\top\operatorname{sgn}\left(\frac{k_1}{k_1+j}A\boldsymbol{z}_0 - \frac{j}{k_1+j}\xi A\boldsymbol{e}_i + \frac{j}{k_1+j}\boldsymbol{b}\right) + \frac{\ell}{k_1+\ell}\boldsymbol{c} \\
&= \frac{k_1}{k_1+\ell}(\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1}) \\
&\qquad - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}A^\top\operatorname{sgn}\left(A\left(\frac{k_1}{k_1+j}\boldsymbol{z}_0 - \frac{j}{k_1+j}\xi\boldsymbol{e}_i + \frac{j}{k_1+j}\boldsymbol{x}^*\right)\right) \\
&\qquad + \frac{\ell}{k_1+\ell}\boldsymbol{c}.
\end{aligned}$$

Note the identities $\operatorname{sgn}(-u) = -\operatorname{sgn}(u)$ and $\operatorname{sgn}(\lambda u) = \operatorname{sgn}(u)$ for any $\lambda > 0$. Therefore, we rescale the argument of sgn in the previous formula by $-(k_1 + j)/(j\xi)$ and reorder its terms to obtain

$$\begin{aligned}\boldsymbol{c} - A^\top \boldsymbol{y}_{k_1+\ell} = \frac{k_1}{k_1+\ell}(\boldsymbol{c} - A^\top \boldsymbol{y}_{k_1}) + \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell} A^\top \operatorname{sgn}\left(A\left(\boldsymbol{e}_i - \frac{k_1}{j\xi}\boldsymbol{z}_0 - \boldsymbol{x}^*/\xi\right)\right)\\ + \frac{\ell}{k_1+\ell}\boldsymbol{c}.\end{aligned}$$

Take the inner product of both sides with $\boldsymbol{e}_i$ to obtain

$$\begin{aligned}\boldsymbol{e}_i^\top(\boldsymbol{c} - A^\top \boldsymbol{y}_{k_1+\ell}) &= \frac{k_1}{k_1+\ell}\boldsymbol{e}_i^\top(\boldsymbol{c} - A^\top \boldsymbol{y}_{k_1}) + \frac{\ell}{k_1+\ell}c_i\\ &\quad + \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell} \boldsymbol{e}_i^\top A^\top \operatorname{sgn}\left(A\underbrace{\left(\boldsymbol{e}_i - \frac{k_1}{j\xi}\boldsymbol{z}_0 - \boldsymbol{x}^*/\xi\right)}_{\boldsymbol{w}+\boldsymbol{\Delta}\text{ in Lemma 6.1.1}}\right).\end{aligned}$$

We can apply Lemma 6.1.1 to the third term above, giving

$$\begin{aligned}\boldsymbol{e}_i^\top(\boldsymbol{c} - A^\top \boldsymbol{y}_{k_1+\ell}) &= \frac{k_1}{k_1+\ell}\boldsymbol{e}_i^\top(\boldsymbol{c} - A^\top \boldsymbol{y}_{k_1}) + \frac{\ell}{k_1+\ell}c_i\\ &\quad + \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell} \left\|A\left(\boldsymbol{e}_i - \frac{k_1}{j\xi}\boldsymbol{z}_0 - \boldsymbol{x}^*/\xi\right)\right\|_1\\ &\quad + \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\left(\frac{k_1}{j\xi}\boldsymbol{z}_0 + \boldsymbol{x}^*/\xi\right)^\top A^\top \operatorname{sgn}\left(A\left(\boldsymbol{e}_i - \frac{k_1}{j\xi}\boldsymbol{z}_0 - \boldsymbol{x}^*/\xi\right)\right)\\ &=: T_1 + T_2 + T_3 + T_4,\end{aligned}$$

where

$$\begin{aligned}
T_1 &= \frac{k_1}{k_1+\ell}\boldsymbol{e}_i^\top(\boldsymbol{c} - A^\top \boldsymbol{y}_{k_1}),\\
T_2 &= \frac{\ell}{k_1+\ell}c_i,\\
T_3 &= \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\left\|A\left(\boldsymbol{e}_i - \frac{k_1}{j\xi}\boldsymbol{z}_0 - \boldsymbol{x}^*/\xi\right)\right\|_1,\\
T_4 &= \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\left(\frac{k_1}{j\xi}\boldsymbol{z}_0 + \boldsymbol{x}^*/\xi\right)^\top A^\top \operatorname{sgn}\left(A\left(\boldsymbol{e}_i - \frac{k_1}{j\xi}\boldsymbol{z}_0 - \boldsymbol{x}^*/\xi\right)\right).
\end{aligned}$$

Let us lower-bound T_3:

$$T_3 \geq \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\|A\boldsymbol{e}_i\|_1 - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\left\|\frac{k_1}{j\xi}A\boldsymbol{z}_0\right\|_1 - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\|\boldsymbol{b}\|_1/\xi \tag{6.10}$$

$$\geq \frac{\eta\ell\|A\boldsymbol{e}_i\|_1}{k_1+\ell} - \frac{\eta\|A\boldsymbol{z}_0\|_1 k_1(1+\ln(\ell))}{(k_1+\ell)\xi} - \frac{\eta\ell\|\boldsymbol{b}\|_1}{(k+\ell)\xi}. \tag{6.11}$$

And lower bound T_4 by taking the sign to be the worst case:

$$T_4 \geq -\frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\left\|\frac{k_1}{j\xi}A\boldsymbol{z}_0\right\|_1 - \frac{\eta}{k_1+\ell}\sum_{j=1}^{\ell}\|\boldsymbol{b}\|_1/\xi \tag{6.12}$$

$$\geq -\frac{\eta\|A\boldsymbol{z}_0\|_1 k_1(1+\ln(\ell))}{(k_1+\ell)\xi} - \frac{\eta\ell\|\boldsymbol{b}\|_1}{(k_1+\ell)\xi}. \tag{6.13}$$

Hence, $T_3 + T_4$ has the following lower bound:

$$T_3 + T_4 \geq \frac{\eta\ell\|A\boldsymbol{e}_i\|_1}{k_1+\ell} - \frac{2\eta\|A\boldsymbol{z}_0\|_1 k_1(1+\ln(\ell))}{(k_1+\ell)\xi} - \frac{2\eta\ell\|\boldsymbol{b}\|_1}{(k_1+\ell)\xi}. \tag{6.14}$$

Here, we have upper bounded $\sum_{j=1}^{\ell}(1/j)$ by $1+\ln(\ell)$. Note that $T_1 \to 0$ as $\ell \to \infty$, and $T_2 \to c_i$. The first term in the above approaches $\eta\|A\boldsymbol{e}_i\|_1$ as $l \to \infty$, the second term approaches zero, and the final term approaches $-2\eta\|\boldsymbol{b}\|_1/\xi$. To ensure that eventually $T_3 + T_4 \geq 0$, we can choose ξ so that the first term dominates the final term. That is, we wish to find a ξ such that

$$\frac{\eta\ell\|A\boldsymbol{e}_i\|_1}{k_1+\ell} \geq \frac{2\eta\ell\|\boldsymbol{b}\|_1}{(k_1+\ell)\xi},$$

which holds for all $\xi \geq 2\|\boldsymbol{b}\|_1/\|A\boldsymbol{e}_i\|_1$. To ensure that $T_2 + T_3 + T_4 \geq 0$, we need to pick η large enough so that the difference between the first term and the final term in (6.14) is larger than T_2. In other words, we require

$$\frac{\eta\ell\|A\boldsymbol{e}_i\|_1}{k_1+\ell} - \frac{2\eta\ell\|\boldsymbol{b}\|_1}{(k_1+\ell)\xi} \geq \frac{\ell}{k_1+\ell}c_i$$

which simplifies to

$$\eta\left(\|A\boldsymbol{e}_i\|_1 - \frac{2\|\boldsymbol{b}\|_1}{\xi}\right) \geq c_i. \tag{6.15}$$

By assumption, $\eta > 0$ and $\xi \geq 2\|\boldsymbol{b}\|_1/\|A\boldsymbol{e}_i\|_1$, so the left hand side of the above must be nonnegative. Hence, in the case that $c_i < 0$, any $\eta > 0$ satisfies (6.15). If $c_i \geq 0$, then we must choose a η that satisfies

$$\eta \geq \frac{c_i}{\|A\boldsymbol{e}_i\|_1 - 2\|\boldsymbol{b}\|_1/\xi} \tag{6.16}$$

which implies that for a choice of η to be possible, we need to set $\xi > 2\|\boldsymbol{b}\|_1/\|A\boldsymbol{e}_i\|_1$. It follows that $\boldsymbol{e}_i^\top(\boldsymbol{c} - A^\top\boldsymbol{y}_{k_1+\ell})$ approaches a nonnegative value for ℓ large enough. Hence, the theorem is proved. □

Observe that the choice of η above is inversely proportional to the choice of ξ. The closer ξ is chosen to $2\|\boldsymbol{b}\|_1/\|A\boldsymbol{e}_i\|_1$, the larger η must be chosen to satisfy its own lower bound (6.16). The above is the only theoretical result that gives insight into choosing parameters η and ξ. More attention is given to the problem of choosing parameters in Section 6.4, where we investigate numerically different choices of η and ξ.

The next two lemmas will prove useful in the theorem that follows.

Lemma 6.1.3. For any $u, v \in \mathbb{R}$, $|u| - |v| \geq (u - v)\operatorname{sgn}(v)$.

Proof. The inequality to be proved is rewritten $|u|-|v| \geq u\cdot\operatorname{sgn}(v)-|v|$, i.e. $|u| \geq u\cdot\operatorname{sgn}(v)$, which is true regardless of v. □

Lemma 6.1.4. For any $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$, $\|u\|_1 - \|v\|_1 \geq (\boldsymbol{u} - \boldsymbol{v})^\top\operatorname{sgn}(\boldsymbol{v})$.

Proof. This follows by noting that the inequality to proved is equivalently written:

$$\sum_{i=1}^{n}(|u_i| - |v_i|) \geq \sum_{i=1}^{n}(u_i - v_i)\operatorname{sgn}(v_i),$$

and this inequality holds term-by-term by Lemma 6.1.3. □

Theorem 6.1.5. Consider one step of Algorithm 6.1 in the case that a dual constraint is infeasible:

$$i := \operatorname{argmin}\{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k : i \in [n]\}, \tag{6.17}$$

$$\boldsymbol{x}_{k+1} := \frac{k}{k+1}\boldsymbol{y}_k + \frac{\xi}{k+1}\boldsymbol{e}_i, \tag{6.18}$$

$$\boldsymbol{y}_{k+1} := \frac{k}{k+1}\boldsymbol{y}_k + \frac{\eta}{k+1}\operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}). \tag{6.19}$$

Then

$$U_{k+1,i} \leq \frac{k}{k+1} \cdot U_{k,i},$$

where

$$U_{k,i} = -\frac{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k}{\eta} + \frac{\|\boldsymbol{b} - A\boldsymbol{x}_k\|_1}{\xi} + \frac{\boldsymbol{c}^\top \boldsymbol{x}_k - \boldsymbol{b}^\top \boldsymbol{y}_k}{\eta\xi},$$

and i as in (6.17).

Proof. Move the term $\frac{\eta}{k+1}\operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1})$ to the LHS of (6.18), multiply both sides by $\boldsymbol{e}_i A^\top$, subtract $c_i \cdot \frac{k}{k+1}$ from both sides, and then divide by η to obtain

$$-\frac{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_{k+1}}{\eta} - \frac{1}{k+1}\left(\boldsymbol{e}_i^\top A^\top \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) - \frac{c_i}{\eta}\right) = -\frac{k}{k+1} \cdot \frac{1}{\eta} \cdot (c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k). \tag{6.20}$$

Next, move the term $\frac{\xi}{k+1}\boldsymbol{e}_i$ to the LHS of (6.19), multiply both sides by A, change signs, add $\boldsymbol{b} \cdot \frac{k}{k+1}$ to both sides, and then divide by ξ to obtain

$$\frac{1}{\xi}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) + \frac{1}{k+1}A\boldsymbol{e}_i - \frac{1}{\xi(k+1)}\boldsymbol{b} = \frac{k}{\xi(k+1)}(\boldsymbol{b} - A\boldsymbol{x}_k), \tag{6.21}$$

which, taking the 1-norm of each side, becomes

$$\left\|\frac{1}{\xi}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) + \frac{1}{k+1}A\boldsymbol{e}_i - \frac{1}{\xi(k+1)}\boldsymbol{b}\right\|_1 = \left\|\frac{k}{\xi(k+1)}(\boldsymbol{b} - A\boldsymbol{x}_k)\right\|_1. \tag{6.22}$$

By Lemma 6.1.4, taking

$$\boldsymbol{u} := \frac{1}{\xi}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) + \frac{1}{k+1}A\boldsymbol{e}_i - \frac{1}{\xi(k+1)}\boldsymbol{b},$$

$$\boldsymbol{v} := \frac{1}{\xi}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}),$$

yields

$$\left\|\frac{1}{\xi}(\boldsymbol{b}-A\boldsymbol{x}_{k+1})+\frac{1}{k+1}A\boldsymbol{e}_i-\frac{1}{\xi(k+1)}\boldsymbol{b}\right\|_1 \geq \left\|\frac{1}{\xi}(\boldsymbol{b}-A\boldsymbol{x}_{k+1})\right\|_1 + \left(\frac{1}{k+1}A\boldsymbol{e}_i-\frac{1}{\xi(k+1)}\boldsymbol{b}\right)^\top \operatorname{sgn}(\boldsymbol{b}-A\boldsymbol{x}_{k+1}). \tag{6.23}$$

Combining (6.22) and (6.23) yields

$$\left\|\frac{1}{\xi}(\boldsymbol{b}-A\boldsymbol{x}_{k+1})\right\|_1 + \left(\frac{1}{k+1}A\boldsymbol{e}_i-\frac{1}{\xi(k+1)}\boldsymbol{b}\right)^\top \operatorname{sgn}(\boldsymbol{b}-A\boldsymbol{x}_{k+1}) \leq \left\|\frac{k}{\xi(k+1)}(\boldsymbol{b}-A\boldsymbol{x}_k)\right\|_1. \tag{6.24}$$

Next, multiply (6.18) by $\frac{1}{\eta\xi}\boldsymbol{c}^\top$ and rearrange to obtain

$$\frac{\boldsymbol{c}^\top\boldsymbol{x}_{k+1}}{\eta\xi}-\frac{c_i}{\eta(k+1)}=\frac{k}{k+1}\cdot\frac{\boldsymbol{c}^\top\boldsymbol{x}_k}{\eta\xi}. \tag{6.25}$$

Next, multiply (6.18) by $-\frac{1}{\eta\xi}\boldsymbol{b}^\top$ and rearrange to obtain

$$-\frac{\boldsymbol{b}^\top\boldsymbol{y}_{k+1}}{\eta\xi}+\frac{\boldsymbol{b}^\top\operatorname{sgn}(\boldsymbol{b}-A\boldsymbol{x}_{k+1})}{\xi(k+1)}=\frac{k}{k+1}\cdot\frac{\boldsymbol{b}^\top\boldsymbol{y}_k}{\eta\xi}. \tag{6.26}$$

Finally, the theorem is proved by adding (6.20), (6.24), (6.25) and (6.26) and noticing that many terms cancel out. □

Theorem 6.1.5 again considers the case where the dual is infeasible. It introduces the potential function $U_{k,i}$ where k is the current iterate and i corresponds to the most infeasible dual constraint. The quantity $U_{k,i}$ has three terms; the first measures dual infeasibility (larger means more infeasible), the second measures primal infeasibility, and the third measures the duality gap. The theorem says that the potential function, which simultaneously measures both infeasibilities and the duality gap, goes down by a constant factor on each iteration.

On its own, Theorem 6.1.5 is not strong enough to establish convergence of the algorithm. The theorem only covers the case of an iteration when the dual is infeasible. It still remains to show what happens in the case that the dual is feasible. Also, i given by (6.17) on iteration k is not necessarily the same i given by (6.17) on iteration $k+1$. Hence this bound does not fully control dual infeasibility.

We consider the case where the dual is feasible in the next theorem.

Theorem 6.1.6. Consider one step of Algorithm 6.1 in the case that all dual constraints are feasible:

$$\boldsymbol{x}_{k+1} := \frac{k}{k+1}\boldsymbol{x}_k, \tag{6.27}$$

$$\boldsymbol{y}_{k+1} := \frac{k}{k+1}\boldsymbol{y}_k + \frac{\eta}{k+1}\operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}). \tag{6.28}$$

Then

$$U_{k+1} \le \frac{k}{k+1} \cdot U_k,$$

where

$$U_k = \frac{\|\boldsymbol{b} - A\boldsymbol{x}_k\|_1}{\xi} + \frac{\boldsymbol{c}^\top \boldsymbol{x}_k - \boldsymbol{b}^\top \boldsymbol{y}_k}{\eta\xi}.$$

Proof. Using (6.27) and (6.28), we can expand the duality gap as

$$\boldsymbol{c}^\top \boldsymbol{x}_{k+1} - \boldsymbol{b}^\top \boldsymbol{y}_{k+1} = \frac{k}{k+1}(\boldsymbol{c}^\top \boldsymbol{x}_k - \boldsymbol{b}^\top \boldsymbol{y}_k) - \frac{\eta}{k+1}\boldsymbol{b}^\top \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}), \tag{6.29}$$

and the primal infeasibility as

$$\begin{align}
\|\boldsymbol{b} - A\boldsymbol{x}_{k+1}\|_1 &= (\boldsymbol{b} - A\boldsymbol{x}_{k+1})^\top \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) \tag{6.30}\\
&= \frac{1}{k+1}\boldsymbol{b}^\top \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) + \frac{k}{k+1}(\boldsymbol{b} - A\boldsymbol{x}_k)^\top \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) \tag{6.31}\\
&\le \frac{1}{k+1}\boldsymbol{b}^\top \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+1}) + \frac{k}{k+1}\|\boldsymbol{b} - A\boldsymbol{x}_k\|_1. \tag{6.32}
\end{align}$$

Multiplying (6.29) by $\frac{1}{\eta}$ and adding this to (6.32) gives the desired inequality if we multiply again by $\frac{1}{\xi}$:

$$\frac{\boldsymbol{c}^\top \boldsymbol{x}_{k+1} - \boldsymbol{b}^\top \boldsymbol{y}_{k+1}}{\eta} + \|\boldsymbol{b} - A\boldsymbol{x}_{k+1}\|_1 \le \frac{k}{k+1}\left(\frac{\boldsymbol{c}^\top \boldsymbol{x}_k - \boldsymbol{b}^\top \boldsymbol{y}_k}{\eta} + \|\boldsymbol{b} - A\boldsymbol{x}_k\|_1\right).$$

□

With Theorems 6.1.5 and 6.1.6, we have potential functions that decrease by a constant factor on each iteration in both the dual infeasible and feasible cases.

The following theorem follows the same idea as the previous two, except instead of making assumptions on the dual it considers the sign pattern of the primal constraint: if the sign of each entry in $\boldsymbol{b} - A\boldsymbol{x}_k$ remains the same for all iterations from k to k', then we can construct a new potential function that decreases over those iterations.

Theorem 6.1.7. Let $\ell = k' - k$ and consider iterations k through k' in the case that $\boldsymbol{\sigma} := \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_t)$ remains constant for all $k \le t \le k'$. Then,

$$D_{k+\ell} \le \frac{k}{k+\ell} \cdot D_k, \tag{6.33}$$

where

$$D_k := -\frac{\left(\sum_{j=1}^{\ell} \boldsymbol{e}_{i(k+j)}\right)^T (\boldsymbol{c} - A^T y)}{\ell\eta} + \frac{\|\boldsymbol{b} - A\boldsymbol{x}_k\|_1}{\xi} + \frac{\boldsymbol{c}^T\boldsymbol{x}_k - \boldsymbol{b}^T y_k}{\eta\xi},$$

and $i(k+j) = \operatorname{argmin}\{c_i = \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_{k+j} : i \in [n]\}$.

Proof. From the $\boldsymbol{x}$ and $\boldsymbol{y}$ updates, we have

$$\boldsymbol{x}_{k+\ell} = \frac{k}{k+\ell}\boldsymbol{x}_k + \frac{\xi}{k+\ell}\sum_{j=1}^{\ell} \boldsymbol{e}_{i(k+j)} \tag{6.34}$$

$$\boldsymbol{y}_{k+\ell} = \frac{k}{k+\ell}\boldsymbol{y}_k + \frac{\eta}{k+\ell}\sum_{j=1}^{\ell} \boldsymbol{\sigma} = \frac{k}{k+\ell}\boldsymbol{y}_k + \frac{\ell\eta}{k+\ell}\boldsymbol{\sigma}. \tag{6.35}$$

From (6.35),

$$\boldsymbol{c} - A^T\boldsymbol{y}_{k+\ell} = \frac{k}{k+\ell}(\boldsymbol{c} - A^T\boldsymbol{y}_k) - \frac{\ell\eta}{k+\ell}A^T\boldsymbol{\sigma} + \frac{\ell}{k+\ell}\boldsymbol{c}.$$

Multiplying the above by $\left(\sum_{j=1}^{\ell} \boldsymbol{e}_{i(k+j)}\right)^T / (\ell\eta)$ gives

$$\begin{aligned}\frac{\left(\sum_{j=1}^{\ell} \boldsymbol{e}_{i(k+j)}\right)^T (\boldsymbol{c} - A^T\boldsymbol{y}_{k+\ell})}{\ell\eta} = {} & \frac{\left(\sum_{j=1}^{\ell} \boldsymbol{e}_{i(k+j)}\right)^T (\boldsymbol{c} - A^T\boldsymbol{y}_k)}{\ell\eta} \\ & - \frac{\left(\sum_{j=1}^{\ell} \boldsymbol{e}_{i(k+j)}\right)^T A^T\boldsymbol{\sigma}}{k+\ell} \\ & + \frac{1}{(k+\ell)\eta}\left(\sum_{j=1}^{\ell} \boldsymbol{e}_{i(k+j)}\right)^T \boldsymbol{c}.\end{aligned} \tag{6.36}$$

From (6.34),

$$\frac{\boldsymbol{b} - A\boldsymbol{x}_{k+\ell}}{\xi} = \frac{k}{k+\ell}\left(\frac{\boldsymbol{b} - A\boldsymbol{x}_k}{\xi}\right) - \frac{1}{k+\ell}\sum_{j=1}^{\ell} A\boldsymbol{e}_{i(k+j)} + \frac{\ell}{k+\ell}\frac{\boldsymbol{b}}{\xi}$$

which allows us to write

$$\left\| \frac{\boldsymbol{b} - A\boldsymbol{x}_{k+\ell}}{\xi} + \frac{1}{k+\ell}\sum_{j=1}^{\ell} A\boldsymbol{e}_{i(k+j)} - \frac{\ell}{k+\ell}\frac{\boldsymbol{b}}{\xi} \right\|_1 = \frac{k}{\xi(k+\ell)} \left\|\boldsymbol{b} - A\boldsymbol{x}_k\right\|_1. \tag{6.37}$$

If we set $\boldsymbol{u} := \frac{\boldsymbol{b}-A\boldsymbol{x}_{k+\ell}}{\xi} + \frac{1}{k+\ell}\sum_{j=1}^{\ell} A\boldsymbol{e}_{i(k+j)} - \frac{\ell}{k+\ell}\frac{\boldsymbol{b}}{\xi}$ and $\boldsymbol{v} := \frac{\boldsymbol{b}-A\boldsymbol{x}_{k+\ell}}{\xi}$, we can apply Lemma 6.1.4 to the left hand side of the above equation and rearrange to obtain the following inequality:

$$\left\| \frac{\boldsymbol{b} - A\boldsymbol{x}_{k+\ell}}{\xi} + \frac{1}{k+\ell}\sum_{j=1}^{\ell} A\boldsymbol{e}_{i(k+j)} - \frac{\ell}{k+\ell}\frac{\boldsymbol{b}}{\xi} \right\|_1 \geq \left(\frac{1}{k+\ell}\sum_{j=1}^{\ell} A\boldsymbol{e}_{i(k+j)} - \frac{\ell}{k+\ell}\frac{\boldsymbol{b}}{\xi} \right)^T \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+\ell}) + \frac{1}{\xi}\|\boldsymbol{b} - A\boldsymbol{x}_{k+\ell}\|_1.$$

The above can be rewritten as follows by applying (6.37):

$$\frac{k}{\xi(k+\ell)} \left\|\boldsymbol{b} - A\boldsymbol{x}_k\right\|_1 \geq \left(\frac{1}{k+\ell}\sum_{j=1}^{\ell} A\boldsymbol{e}_{i(k+j)} - \frac{\ell}{k+\ell}\frac{\boldsymbol{b}}{\xi} \right)^T \operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_{k+\ell}) + \frac{1}{\xi}\|\boldsymbol{b} - A\boldsymbol{x}_{k+\ell}\|_1. \tag{6.38}$$

Now, multiply (6.34) by $\frac{1}{\eta\xi}\boldsymbol{c}^T$ and rearrange to obtain

$$\frac{\boldsymbol{c}^T\boldsymbol{x}_{k+\ell}}{\eta\xi} - \frac{\sum_{j=1}^{\ell} \boldsymbol{c}^T\boldsymbol{e}_{i(k+j)}}{\eta(k+\ell)} = \frac{k}{k+\ell} \cdot \frac{\boldsymbol{c}^T\boldsymbol{x}_k}{\eta\xi}. \tag{6.39}$$

Similarly, multiply (6.35) by $-\frac{1}{\eta\xi}\boldsymbol{b}^T$ and rearrange to obtain

$$\frac{-\boldsymbol{b}^T\boldsymbol{y}_{k+\ell}}{\eta\xi} + \frac{\sum_{j=1}^{\ell} \boldsymbol{b}^T\boldsymbol{\sigma}}{\xi(k+\ell)} = -\frac{k}{k+\ell} \cdot \frac{\boldsymbol{b}^T\boldsymbol{y}_k}{\eta\xi}. \tag{6.40}$$

Finally, we achieve the desired inequality (6.33) by adding (6.38), (6.39) and (6.40), and subtracting (6.36). □

6.1.1 Relation to the primal-dual gap for saddle point problems

Recall the primal-dual gap defined in (5.26). Unlike in PDLP where (5.26) may be infinite, our definitions of $\mathcal{X}$ and $\mathcal{Y}$ for the problem (6.1) force the primal-dual gap to be finite.

We will also see that in the setting of FWLP, the primal-dual gap is easy to compute, especially in comparison to the normalized duality gap (5.25) used in PDLP. Observe that the primal-dual gap function $\rho(\boldsymbol{x}, \boldsymbol{y})$ can be rewritten as

$$\rho(\boldsymbol{x}, \boldsymbol{y}) = \max_{\hat{\boldsymbol{y}} \in \mathcal{Y}} \mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{y}}) - \min_{\hat{\boldsymbol{x}} \in \mathcal{X}} \mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{y}), \tag{6.41}$$

where we may view the first term $\max\{\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{y}}) : \hat{\boldsymbol{y}} \in \mathcal{Y}\}$ as the primal objective function, and the second term $\min\{\mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{y}) : \hat{\boldsymbol{x}} \in \mathcal{X}\}$ as the dual objective function. From the form in (6.41), it is easy to see that $\rho(\boldsymbol{x}, \boldsymbol{y}) \geq 0$ for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$.

There is a connection between the potential functions $U_{k,i}$ and U_k from Theorems 6.1.5 and 6.1.6 and the function $\rho(\boldsymbol{x}, \boldsymbol{y})$. Suppose the dual problem of (6.1) is infeasible with $i = \text{argmin}\{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k : i \in [n]\}$. Then the primal objective evaluated at the current iterate $\boldsymbol{x}_k$ is

$$\begin{aligned} \max_{\hat{\boldsymbol{y}} \in \mathcal{Y}} \mathcal{L}(\boldsymbol{x}_k, \hat{\boldsymbol{y}}) &= \max_{\hat{\boldsymbol{y}} \in \mathcal{Y}} \{\boldsymbol{c}^\top \boldsymbol{x}_k + \hat{\boldsymbol{y}}^\top (\boldsymbol{b} - A\boldsymbol{x}_k)\} && (6.42)\\ &= \boldsymbol{c}^\top \boldsymbol{x}_k + \eta \|\boldsymbol{b} - A\boldsymbol{x}_k\|_1, && (6.43) \end{aligned}$$

and the dual objective evaluated at the current iterate $\boldsymbol{y}_k$ is

$$\begin{aligned} \min_{\hat{\boldsymbol{x}} \in \mathcal{X}} \mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{y}_k) &= \min_{\hat{\boldsymbol{x}} \in \mathcal{X}} \{(\boldsymbol{c} - A^\top \boldsymbol{y}_k)^\top \hat{\boldsymbol{x}}\} + b^\top \boldsymbol{y}_k && (6.44)\\ &= c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k + \boldsymbol{b}^\top \boldsymbol{y}_k. && (6.45) \end{aligned}$$

It follows that

$$\begin{aligned} \rho(\boldsymbol{x}_k, \boldsymbol{y}_k) &= \max_{\hat{\boldsymbol{y}} \in \mathcal{Y}} \mathcal{L}(\boldsymbol{x}_k, \hat{\boldsymbol{y}}) - \min_{\hat{\boldsymbol{x}} \in \mathcal{X}} \mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{y}_k) && (6.46)\\ &= -\xi(c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k) + \eta \|\boldsymbol{b} - A\boldsymbol{x}_k\|_1 + \boldsymbol{c}^\top \boldsymbol{x}_k - \boldsymbol{b}^\top \boldsymbol{y}_k && (6.47)\\ &= \eta\xi \cdot U_{k,i}. && (6.48) \end{aligned}$$

On the other hand, if the dual problem of (6.1) is feasible, then the primal objective evaluated at $\boldsymbol{x}_k$ remains the same as in (6.43), but the dual objective at $\boldsymbol{y}_k$ becomes

$$\begin{aligned} \min_{\hat{\boldsymbol{x}} \in \mathcal{X}} \mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{y}_k) &= \min_{\hat{\boldsymbol{x}} \in \mathcal{X}} \{(\boldsymbol{c} - A^\top \boldsymbol{y}_k)^\top \hat{\boldsymbol{x}}\} + b^\top \boldsymbol{y}_k && (6.49)\\ &= \boldsymbol{b}^\top \boldsymbol{y}_k, && (6.50) \end{aligned}$$

since the optimal choice of $\hat{\boldsymbol{x}}$ is zero in the case where all entries of $\boldsymbol{c} - A^\top \boldsymbol{y}_k$ are nonnegative. Now we see that

$$\begin{aligned} \rho(\boldsymbol{x}_k, \boldsymbol{y}_k) &= \eta \|\boldsymbol{b} - A\boldsymbol{x}_k\|_1 + \boldsymbol{c}^\top \boldsymbol{x}_k - \boldsymbol{b}^\top \boldsymbol{y}_k && (6.51)\\ &= \eta\xi \cdot U_k. && (6.52) \end{aligned}$$

We may now write the primal-dual gap function another way:

$$\rho(\boldsymbol{x}_k, \boldsymbol{y}_k) = \begin{cases} U_k, & \boldsymbol{c} - A^\top \boldsymbol{y}_k \geq \boldsymbol{0}; \\ U_{k,i}, & i = \operatorname{argmin}\{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k : c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}_k < 0, i \in [n]\}. \end{cases}$$

Using the above equivalence, we state the below theorem. Along with the fact that $\rho(\boldsymbol{x}, \boldsymbol{y}) \geq 0$, Theorem 6.1.8 suggests that $\rho(\boldsymbol{x}, \boldsymbol{y})$ can be used to measure the optimality of the FWLP problem.

Theorem 6.1.8. Suppose $\boldsymbol{x}$ is feasible for $\mathcal{X}$ but $\boldsymbol{e}^\top \boldsymbol{x} < \xi$. Likewise, suppose $\boldsymbol{y}$ is feasible for $\mathcal{Y}$, but $-\eta \boldsymbol{e} < \boldsymbol{y} < \eta \boldsymbol{e}$. Then $\rho(\boldsymbol{x}, \boldsymbol{y}) = 0$ if and only if $(\boldsymbol{x}, \boldsymbol{y})$ are optimal for (MLP) and (MDP).

Proof. First, suppose $\rho(\boldsymbol{x}, \boldsymbol{y}) = 0$ and let $\boldsymbol{y}$ satisfy the constraint $A^\top \boldsymbol{y} \leq \boldsymbol{c}$. Then

$$0 = \rho(\boldsymbol{x}, \boldsymbol{y}) = \frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_1}{\xi} + \frac{\boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{y}}{\eta \xi}.$$

If $A\boldsymbol{x} = \boldsymbol{b}$, we have from the above that $\boldsymbol{c}^\top \boldsymbol{x} = \boldsymbol{b}^\top \boldsymbol{y}$. By weak duality, we have that $(\boldsymbol{x}, \boldsymbol{y})$ are optimal for the primal-dual pair of LPs (LP) and (DLP). If we add the constraints $\boldsymbol{e}^\top \boldsymbol{x} \leq \xi$ and $\|\boldsymbol{y}\|_\infty \leq \eta$, we are minimizing and maximizing over smaller sets and thus $\boldsymbol{c}^\top \boldsymbol{x}$ is a lower bound on (MLP) and $\boldsymbol{b}^\top \boldsymbol{y}$ is an upper bound on (MDP). However, both bounds are obtained for feasible $\boldsymbol{x}$ and $\boldsymbol{y}$, so $(\boldsymbol{x}, \boldsymbol{y})$ must be optimal for the modified LPs.

If $A\boldsymbol{x} \neq \boldsymbol{b}$, $\|\boldsymbol{b} - A\boldsymbol{x}\|_1 > 0$, and from the constraint $A^\top \boldsymbol{y} \leq \boldsymbol{c}$, we obtain the inequality

$$\boldsymbol{c}^\top \boldsymbol{x} \geq (A^\top \boldsymbol{y})^\top \boldsymbol{x} = \boldsymbol{y}^\top A \boldsymbol{x},$$

so

$$\rho(\boldsymbol{x}, \boldsymbol{y}) \geq \frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_1}{\xi} + \frac{\boldsymbol{y}^\top (A\boldsymbol{x} - \boldsymbol{b})}{\eta \xi}.$$

Using $-\boldsymbol{y} > -\eta \boldsymbol{e}$ on the above, we obtain the inequality

$$\begin{aligned} \rho(\boldsymbol{x}, \boldsymbol{y}) &\geq \frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_1}{\xi} + \frac{\boldsymbol{y}^\top (A\boldsymbol{x} - \boldsymbol{b})}{\eta \xi} \\ &> \frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_1}{\xi} - \frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_1}{\xi} \\ &= 0. \end{aligned}$$

It follows that $\rho(\boldsymbol{x}, \boldsymbol{y}) \neq 0$, contradicting the choice of $A\boldsymbol{x} \neq \boldsymbol{b}$.

Now we consider the case where $\boldsymbol{y}$ does not satisfy $A^\top \boldsymbol{y} \leq \boldsymbol{c}$ and $i = \operatorname{argmin}\{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}\}$. In this case,

$$\rho(\boldsymbol{x}, \boldsymbol{y}) = -\frac{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}}{\eta} + \frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_1}{\xi} + \frac{\boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{y}}{\eta\xi}.$$

We may write

$$\begin{aligned} \boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{y} &= \boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{y}^\top A\boldsymbol{x} - \boldsymbol{y}^\top (\boldsymbol{b} - A\boldsymbol{x}) \\ &> \boldsymbol{x}^\top (\boldsymbol{c} - A^\top \boldsymbol{y}) - \eta\|\boldsymbol{b} - A\boldsymbol{x}\|_1 \\ &> \xi(c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}) - \eta\|\boldsymbol{b} - A\boldsymbol{x}\|_1, \end{aligned}$$

which if we divide by $\eta\xi$ gives the inequality

$$-\frac{c_i - \boldsymbol{e}_i^\top A^\top \boldsymbol{y}}{\eta} + \frac{\|\boldsymbol{b} - A\boldsymbol{x}\|_1}{\xi} + \frac{\boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{y}}{\eta\xi} > 0.$$

Therefore, $\rho(\boldsymbol{x}, \boldsymbol{y}) \neq 0$, so $\boldsymbol{y}$ satisfying $A^\top \boldsymbol{y} \leq \boldsymbol{c}$ is a necessary condition for $\rho(\boldsymbol{x}, \boldsymbol{y}) = 0$.

Now suppose $\boldsymbol{x}$ is an optimal solution to (MLP) and $\boldsymbol{y}$ is an optimal solution to (MDP). Since $\boldsymbol{x}$ and $\boldsymbol{y}$ are feasible, we have that

$$\rho(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{y}}{\eta\xi}.$$

Since $\boldsymbol{x}$ and $\boldsymbol{y}$ are optimal for (MLP) and (MDP) such that the constraints described by ξ and η are redundant, $\boldsymbol{x}$ and $\boldsymbol{y}$ must be optimal for (LP) and (DLP). It follows from strong duality that $\rho(\boldsymbol{x}, \boldsymbol{y}) = 0$. □

Remark. It is clear from the above proof that under the same assumptions as Theorem 6.1.8, $\rho(\boldsymbol{x}, \boldsymbol{y}) = 0$ if and only if $(\boldsymbol{x}, \boldsymbol{y})$ are optimal for (LP) and (DLP).

6.2 Connection to Hammond's Generalized Fictitious Play Algorithm

In her thesis [30], Hammond describes an algorithm for solving the VIP (4.1). Given iterate $\boldsymbol{z}_k$, the algorithm titled *Generalized Fictitious Play Algorithm*, finds $\boldsymbol{z}_{k+1}$ by first solving the linear optimization problem

$$\boldsymbol{s} := \operatorname*{argmin}_{\boldsymbol{s} \in \mathcal{Z}} \boldsymbol{s}^\top F(\boldsymbol{z}_k), \tag{6.53}$$

and then updating as follows:

$$\boldsymbol{z}_{k+1} = \frac{k}{k+1}\boldsymbol{z}_k + \frac{1}{k+1}\boldsymbol{s}. \tag{6.54}$$

From Section (4.0.1), we know that the problem of finding a saddle point of (6.1) is equivalent to the VIP (4.1) with $\boldsymbol{z} := (\boldsymbol{x}, \boldsymbol{y})$, $F(\boldsymbol{z}) := (\boldsymbol{c} - A^\top \boldsymbol{y}, A\boldsymbol{x} - \boldsymbol{b})$, and $\mathcal{Z} := \mathcal{X} \times \mathcal{Y}$. Applying Hammond's Generalized Fictitious Play algorithm to this case yields a very similar algorithm to FWLP, the only difference being that $\boldsymbol{y}_k$ is updated based on $\boldsymbol{x}_k$ and not the newly found $\boldsymbol{x}_{k+1}$:

Algorithm 6.2 Generalized Fictitious Play applied to (6.1).

```
Input: Starting points $\boldsymbol{x}_1 \in \mathbb{R}^n, \boldsymbol{y}_1 \in \mathbb{R}^m$, constraint data $A$ and $\boldsymbol{b}$, and objective $\boldsymbol{c}$.
    Parameters: $\xi > 0$ such that $\boldsymbol{e}^\top \boldsymbol{x}_k \leq \xi$, $\eta > 0$ such that $\|\boldsymbol{y}_k\|_\infty \leq \eta$.
1: for $k = 1, 2, \ldots$ do
2:     Determine $i = \text{argmin}_t [\boldsymbol{c} - A^\top \boldsymbol{y}_k]_t$.
3:     if $[\boldsymbol{c} - A^\top \boldsymbol{y}_k]_i \geq 0$ then
4:         Step towards zero in $\boldsymbol{x}$:
               $$\boldsymbol{x}_{k+1} = \frac{k}{k+1}\boldsymbol{x}_k. \tag{6.55}$$
5:     else
6:         Step toward $\xi$ for $x_i$, otherwise step toward zero for $x_j, j \neq i$:
               $$\boldsymbol{x}_{k+1} = \frac{k}{k+1}\boldsymbol{x}_k + \frac{\xi}{k+1}\boldsymbol{e}_i. \tag{6.56}$$
7:     Step towards $\pm\eta$ for each coordinate in $y$ according to the sign pattern of $\boldsymbol{b} - A\boldsymbol{x}_k$.
               $$\boldsymbol{y}_{k+1} = \frac{k}{k+1}\boldsymbol{y}_k + \frac{\eta}{k+1}\,\text{sgn}(\boldsymbol{b} - A\boldsymbol{x}_k). \tag{6.57}$$
```

Hammond's conjecture states that if F is strongly monotone and $\mathcal{Z}$ is a bounded polyhedron, Generalized Fictitious Play will solve (4.1). However, the F defined above does not satisfy strong monotonicity and as such is not covered under Hammond's conjecture.

6.3 Advantages of FWLP

FWLP is a first-order method in that at each iteration its most expensive operation is a matrix-vector product. As mentioned, this is beneficial for solving large linear programming problems because of the decreased operation cost, as well as the decreased memory usage. A benefit of FWLP is that we only need to store $A\boldsymbol{x}_k$ and $\boldsymbol{y}_k$, which we then update at each iteration. For example, suppose we have stored $A\boldsymbol{x}_1$ and $A\boldsymbol{e}_i$ for each $i \in [n]$. Then to update $A\boldsymbol{x}_1$ we need only perform one of two operations:

$$A\boldsymbol{x}_2 = \frac{k}{2}A\boldsymbol{x}_1, \quad \text{or} \quad A\boldsymbol{x}_2 = \frac{k}{2}A\boldsymbol{x}_1 + \frac{\xi}{2}A\boldsymbol{e}_i, \text{ for some } i \in [n].$$

Such an update runs in $\mathcal{O}(m)$ operations.

The update for $\boldsymbol{y}_1$ is

$$\boldsymbol{y}_2 = \frac{k}{2}\boldsymbol{y}_1 + \frac{\eta}{2}\operatorname{sgn}(\boldsymbol{b} - A\boldsymbol{x}_2),$$

which also runs in $\mathcal{O}(m)$ since $A\boldsymbol{x}_2$ has already been computed. The computation of $A^\top y$ can be improved if we note that in Algorithm 6.1 we only need to compute $(A\boldsymbol{e}_i)^\top \boldsymbol{y}_k$ for the indices i such that there is a possibility that $c_i - (A\boldsymbol{e}_i)^\top \boldsymbol{y}_k$ could become most infeasible at the next iteration. This would require a data structure to store the indices $[n]$ in some order so that only the possibly most infeasible indices need to be considered at each iteration. For example, if the majority of iterates $\boldsymbol{y}_k$ satisfy the inequality

$$c_t - (A\boldsymbol{e}_t)^\top \boldsymbol{y}_k \geq c_i - (A\boldsymbol{e}_i)^\top \boldsymbol{y}_k + D,$$

where t belongs to a subset of indices $\mathcal{S} \subseteq [n]$, i is the index of the most violated constraint, and $D > 0$ is large enough so that t doesn't become the most violated constraint on a number of consecutive iterations. Then for the number of iterations that we don't need to consider the indices in $\mathcal{S}$, the computation of $A^\top \boldsymbol{y}_k$ costs $\mathcal{O}(|[n] \setminus \mathcal{S}|m)$ plus the cost of updating the data structure storing the indices.

One might not even be interested in the value of $\boldsymbol{x}^*$. In this case, we don't need to store the iterates $\boldsymbol{x}_k$ at each iteration, only $A\boldsymbol{x}_k$.

6.4 Choosing ξ and η

Although we lack theoretical results regarding the choice of parameters η and ξ, the proof of Theorem 6.1.2 gives some insight into lower bounds. Suppose i is the index corresponding to the most violated constraint in the dual. We know from the proof of Theorem 6.1.2 that the lower bound on η (6.16) must be satisfied, along with the lower bound $\xi > 2\|\boldsymbol{b}\|_1/\|A\boldsymbol{e}_i\|_1$. Observe that

$$\frac{c_i}{\|A\boldsymbol{e}_i\|_1 - 2\|\boldsymbol{b}\|_1/\xi} \leq \frac{\max\{c_t : t \in [n]\}}{\min\{\|A\boldsymbol{e}_t\|_1 : t \in [n]\} - 2\|\boldsymbol{b}\|_1/\xi},$$

and

$$\frac{2\|\boldsymbol{b}\|_1}{\|A\boldsymbol{e}_i\|_1} \leq \frac{2\|\boldsymbol{b}\|_1}{\min\{\|A\boldsymbol{e}_t\|_1 : t \in [n]\}}.$$

It follows that to satisfy the above lower bounds on η and ξ in Algorithm 6.1, we may first choose some ξ satisfying

$$\xi > \frac{2\|\boldsymbol{b}\|_1}{\min\{\|A\boldsymbol{e}_t\|_1 : t \in [n]\}}$$

and then define η as

$$\eta := \frac{\max\{c_t : t \in [n]\}}{\min\{\|A\boldsymbol{e}_t\|_1 : t \in [n]\} - 2\|\boldsymbol{b}\|_1/\xi}.$$

Recall that the closer we choose ξ to its lower bound, the larger η becomes.

To investigate which parameter values tend to work best, we performed numerical experiments on the five randomly generated linear programming problems described in Table 6.1, where κ is the condition number of the optimal basis A_{B^*} for each problem:

$$\kappa = \frac{\sigma_{\max}(A_{B^*})}{\sigma_{\min}(A_{B^*})}.$$

Problem No.	m	n	κ
1	2	5	9.0946
2	5	8	9.8995
3	6	10	105.28
4	6	10	60.396
5	7	11	12.463

Table 6.1. The five random problems used for numerical experiments.

For each problem, we ran FWLP for 100 million iterations, 100 times, varying the parameters η and ξ each time. Define

$$\xi_{\min} := \left(1 + 10^{-2}\right) \frac{2\|\boldsymbol{b}\|_1}{\min\{\|A\boldsymbol{e}_t\|_1 : t \in [n]\}},$$

and

$$\eta_{\min} := \frac{\max\{c_t : t \in [n]\}}{\min\{\|A\boldsymbol{e}_t\|_1 : t \in [n]\} - 2\|\boldsymbol{b}\|_1/\xi}.$$

For a given problem p, we performed the following algorithm, saving the KKT error of the averaged iterates at each iteration. We denote the averaged iterates by $(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{y}}_k)$.

Algorithm 6.3 Numerical experiments comparing combinations of parameters η and ξ.

```
Input: Problem data corresponding to $p$, $\xi_{\min}$, and $\eta_{\min}$.
1: for $i = 1, 2, \ldots, 10$ do
2:     for $j = 1, 2, \ldots, 10$ do
3:         Run FWLP for 100 million iterations on problem $p$ with $\xi = i \cdot \xi_{\min}, \eta = j \cdot \eta_{\min}$.
4:         Save KKT error for $(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{y}}_k)$ for this value of $i$ and $j$.
```

The KKT error[1] measures how far away the primal-dual pair $(\boldsymbol{x}_k, \boldsymbol{y}_k)$ is from satisfying the KKT conditions of (LP). It is defined as $\|[\boldsymbol{h} - K\boldsymbol{z}]_+\|_2$, where $\boldsymbol{z} = (\boldsymbol{x}_k, \boldsymbol{y}_k)$ and

$$K := \begin{bmatrix} I_{n\times n} & 0_{n\times m} \\ -A & 0_{m\times m} \\ A & 0_{m\times m} \\ 0_{n\times n} & -A^\top \\ -\boldsymbol{c}^\top & \boldsymbol{b}^\top \end{bmatrix}, \quad h := \begin{bmatrix} \boldsymbol{0} \\ -\boldsymbol{b} \\ \boldsymbol{b} \\ -\boldsymbol{c} \\ 0 \end{bmatrix}.$$

Figure 6.1 presents the results of the experiments in the form of a heatmap, where the values along the x-axis represent the amount η is scaled (i.e. j in Algorithm 6.3), and the values along the y-axis represent the amount ξ is scaled (i.e. i in Algorithm 6.3). The color values of the heatmap represent the inverse of the KKT error corresponding to the chosen parameters, normalized by the best inverse KKT error for that problem. In other words, for a given problem p, let E^p_{best} denote the best KKT error of the averaged iterates after 100 million iterations for any η and ξ. Let $E^p(\eta, \xi)$ denote the KKT error of the averaged iterates after 100 million iterations for that specific combination of η and ξ. Then,

$$\text{Heatmap color value for combination } (p, \eta, \xi) = \begin{cases} 1 & E^p(\eta, \xi) = 0, \\ \frac{1/E^p(\eta,\xi)}{1/E^p_{\text{best}}} & \text{otherwise.} \end{cases}$$

This implies that a more red square in Figure 6.1 corresponds to a combination (η, ξ) with a smaller KKT error, while a more blue square corresponds to a combination with a higher KKT error.

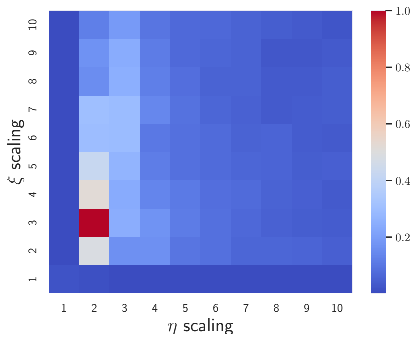
[1]The KKT error is used in the PDLP paper [3] and is commonly used as a metric for the termination of LP solvers [1].

In addition to the heatmaps, Tables 6.2-6.6 show the smallest five KKT error values and their corresponding combination (η, ξ) for the averaged iterates after 100 million iterations.

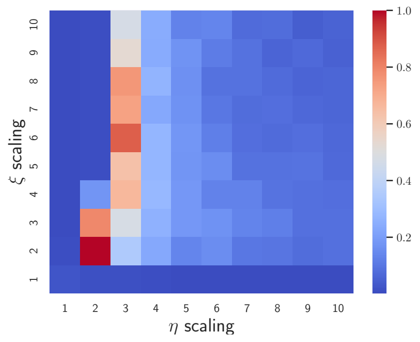
Analyzing the results, it appears that for most of the problems tested, values of η around $2\eta_{\min}$ seem to give the smallest KKT errors. There is some variation from this for Problems 4 and 5. Looking at the heatmaps, there does appear to be some positive correlation between the size of ξ and the size of η, particularly in the heatmaps corresponding to Problem 4 and Problem 5, and to a lesser extent in Problem 2. Values of ξ that perform well on our test problems appear to be concentrated between $2\xi_{\min}$ and $4\xi_{\min}$. However, Problems 2 and 5 suggest that choices of ξ between $2\xi_{\min}$ and $10\xi_{\min}$ perform well, so long as a large enough η is selected. All heatmaps show that selecting η and ξ to be their minimum values leads to poor performance in comparison to other choices.

Analyzing the values of ξ and η in Tables 6.2-6.6, we see that the best performing values of η for the given test problems seem to be in the range of 1.5 to 4, while values for ξ in the range $[280, 480]$ perform well in our experiments.

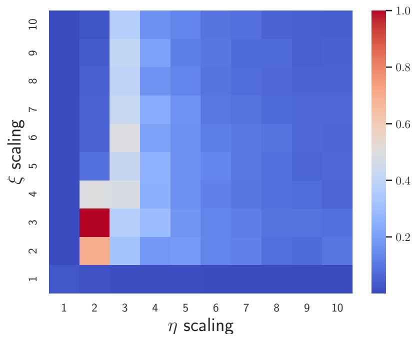
In the numerical results, there seems to be no observable correlation between the size of ξ and η and the problem size. However, the size of Problems 1-5 may not vary enough to show such a correlation.



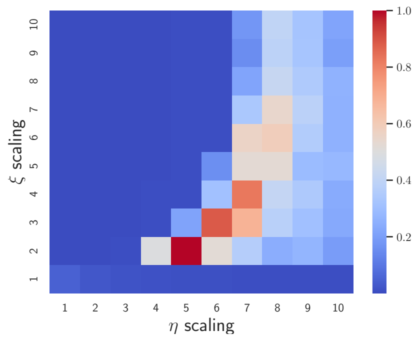
(a) Problem 1



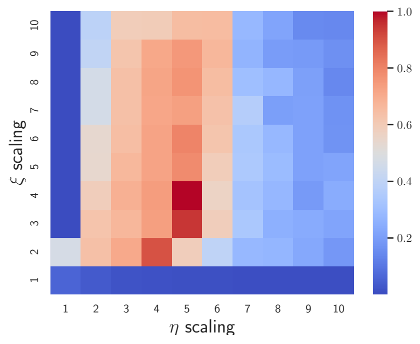
(b) Problem 2



(c) Problem 3



(d) Problem 4



(e) Problem 5

Figure 6.1. Parameter heatmaps comparing the interaction between various ξ and η values.

ξ	ξ Scaling	η	η Scaling	KKT Error
436.86	3	4.2370	2	0.003568
582.48	4	3.7725	2	0.006909
291.24	2	5.6217	2	0.007320
728.10	5	3.5396	2	0.008396
1019.30	7	3.3063	2	0.011816

Table 6.2. Problem 1: KKT error of the averaged iterates for the top 5 combinations of η and ξ.

ξ	ξ Scaling	η	η Scaling	Average KKT Error
425.60	2	3.3718	2	0.027250
1276.80	6	3.0586	3	0.031112
638.41	3	2.5413	2	0.034511
1702.40	8	2.9146	3	0.035997
1489.60	7	2.9746	3	0.037376

Table 6.3. Problem 2: KKT error of the averaged iterates for the top 5 combinations of η and ξ.

ξ	ξ Scaling	η	η Scaling	Average KKT Error
481.17	3	1.5514	2	0.018702
320.78	2	2.0584	2	0.026696
641.56	4	1.3813	2	0.037801
962.34	6	1.8672	3	0.038138
641.56	4	2.0719	3	0.039077

Table 6.4. Problem 3: KKT error of the averaged iterates for the top 5 combinations of η and ξ.

ξ	ξ Scaling	η	η Scaling	Average KKT Error
287.77	2	2.5795	5	0.017490
431.66	3	2.3330	6	0.019754
575.54	4	2.4234	7	0.021200
431.66	3	2.7218	7	0.025605
863.32	6	2.4959	8	0.029688

Table 6.5. Problem 4: KKT error of the averaged iterates for the top 5 combinations of η and ξ.

ξ	ξ Scaling	η	η Scaling	Average KKT Error
407.71	4	2.4353	5	0.022427
305.79	3	2.7352	5	0.023667
203.86	2	2.9033	4	0.024882
611.57	6	2.1947	5	0.028071
509.64	5	2.2850	5	0.028567

Table 6.6. Problem 5: KKT error of the averaged iterates for the top 5 combinations of η and ξ.

6.5 Restarting FWLP

For Problems 1-5, we ran FWLP for 100 million iterations, for each of the following restart schemes:

1. **Constant restarts:** Restart every τ iterations. We use $\tau = 10^6$ and $\tau = 2 \cdot 10^6$.

2. **Sufficient decay in primal-dual gap:** Inspired by the similarly named restart scheme used in PDLP (cf. Section 5.3.2), we restart when

$$\rho(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1}) \leq \beta_{\text{suff}} \rho(\bar{\boldsymbol{x}}_{t,1}, \bar{\boldsymbol{y}}_{t,1}).$$

   This restart scheme is named 'Adaptive 1' in the below plots.

3. **Necessary decay and no local progress in primal-dual gap:** Also inspired by the similarly named restart scheme used in PDLP and discussed in Section 5.3.2, we restart when

$$\rho(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1}) \leq \beta_{\text{nec}} \rho(\bar{\boldsymbol{x}}_{t,1}, \bar{\boldsymbol{y}}_{t,1}),$$

   and

$$\rho(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1}) > \rho(\bar{\boldsymbol{x}}_{t,k}, \bar{\boldsymbol{y}}_{t,k}).$$

   This restart scheme is named 'Adaptive 2' in the below plots.

Algorithm 6.4 FWLP (restarted): FWLP with restarts.

```
Input: Starting points $\boldsymbol{x}_1 \in \mathbb{R}^n, \boldsymbol{y}_1 \in \mathbb{R}^m$, $(\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{y}}_1) = (\boldsymbol{x}_1, \boldsymbol{y}_1)$, constraint data $A$ and $\boldsymbol{b}$,
    and objective $\boldsymbol{c}$.
    Parameters: $\xi > 0$ such that $\boldsymbol{e}^\top \boldsymbol{x}_k \leq \xi$, $\eta > 0$ such that $\|\boldsymbol{y}_k\|_\infty \leq \eta$.
1: for $k = 1, 2, \dots$ do
2:     Perform FWLP step according to Algorithm 6.1 and store averaged iterates:
           $(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}) = \mathrm{FWLP}(k, \boldsymbol{x}_k, \boldsymbol{y}_k, A, \boldsymbol{b}, \boldsymbol{c}, \xi, \eta),$
           $(\bar{\boldsymbol{x}}_{k+1}, \bar{\boldsymbol{y}}_{k+1}) = \frac{k}{k+1}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{y}}_k) + \frac{1}{k+1}(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}).$
3:     if restart criteria are satisfied then
           $(\boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}) = (\bar{\boldsymbol{x}}_{k+1}, \bar{\boldsymbol{y}}_{k+1}).$
```

Notably, we are using the primal-dual gap (5.26) instead of the normalized duality gap (5.25). This is because the motivation for using the normalized duality gap in PDLP is that (5.26) could be unbounded. In the setting of FWLP however, (5.26) is always bounded, so the use of the normalized duality gap is unnecessary.

In our initial experiments comparing the above restart schemes, we chose the restart candidate similar to (5.27):

$$(\boldsymbol{x}^c_{t,k+1}, \boldsymbol{y}^c_{t,k+1}) = \begin{cases} (\boldsymbol{x}_{t,k+1}, \boldsymbol{y}_{t,k+1}), & \rho(\boldsymbol{x}_{t,k+1}, \boldsymbol{y}_{t,k+1}) < \rho(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1}); \\ (\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1}), & \text{otherwise.} \end{cases} \tag{6.58}$$

However, we noted that $\rho(\boldsymbol{x}_{t,k+1}, \boldsymbol{y}_{t,k+1}) < \rho(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1})$ was never not satisfied numerically for Problems 1-5. Hence, every restart candidate was chosen to be $(\boldsymbol{x}_{t,k+1}, \boldsymbol{y}_{t,k+1})$. Since we do not update the step-sizes when restarting in FWLP (cf. Algorithm 6.4), this made restarting redundant. For the numerical experiments presented here, the restart candidate is always chosen to be $(\bar{\boldsymbol{x}}_{t,k+1}, \bar{\boldsymbol{y}}_{t,k+1})$. This is reflected in the restart schemes listed above.

In our numerical experiments we chose $\beta_{\text{suff}} = 0.5$ and $\beta_{\text{nec}} = 0.1$.

Figures 6.2 through 6.6 show the KKT error of the averaged iterates $(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{y}}_k)$ over 100 million iterations with different restarting schemes. The results show that for each problem, no restarts and the 'Adaptive 2' restart scheme perform the best. This suggests that restarting may not provide any benefit to FWLP.

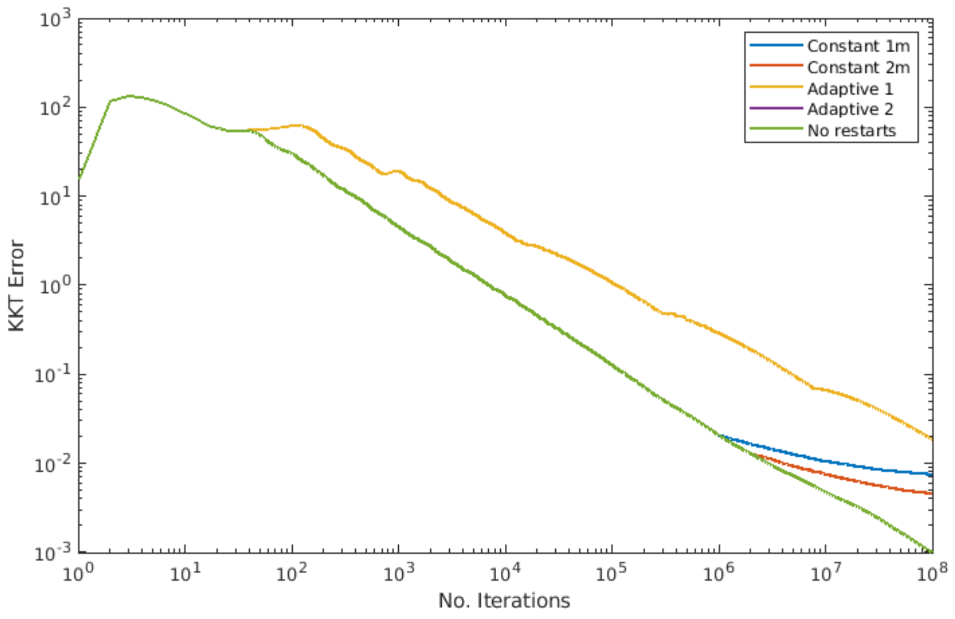


Figure 6.2. Problem 1: KKT error of averaged iterates for different restarting schemes.

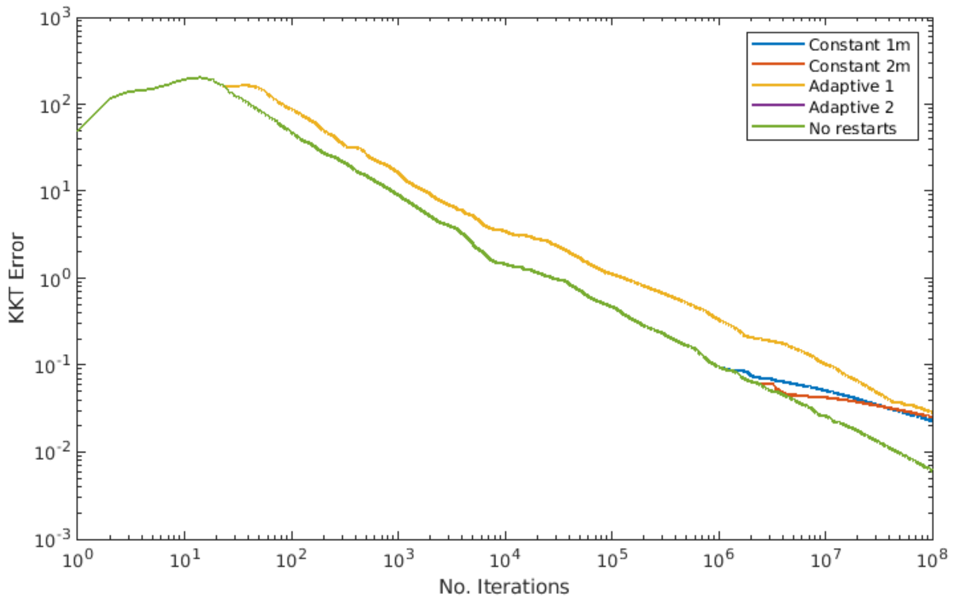


Figure 6.3. Problem 2: KKT error of averaged iterates for different restarting schemes.

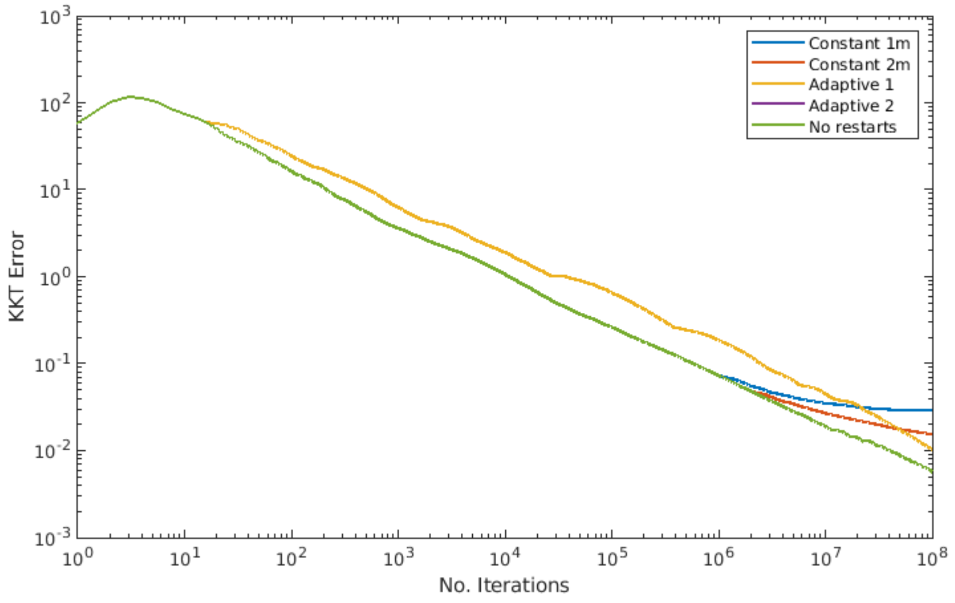


Figure 6.4. Problem 3: KKT error of averaged iterates for different restarting schemes.

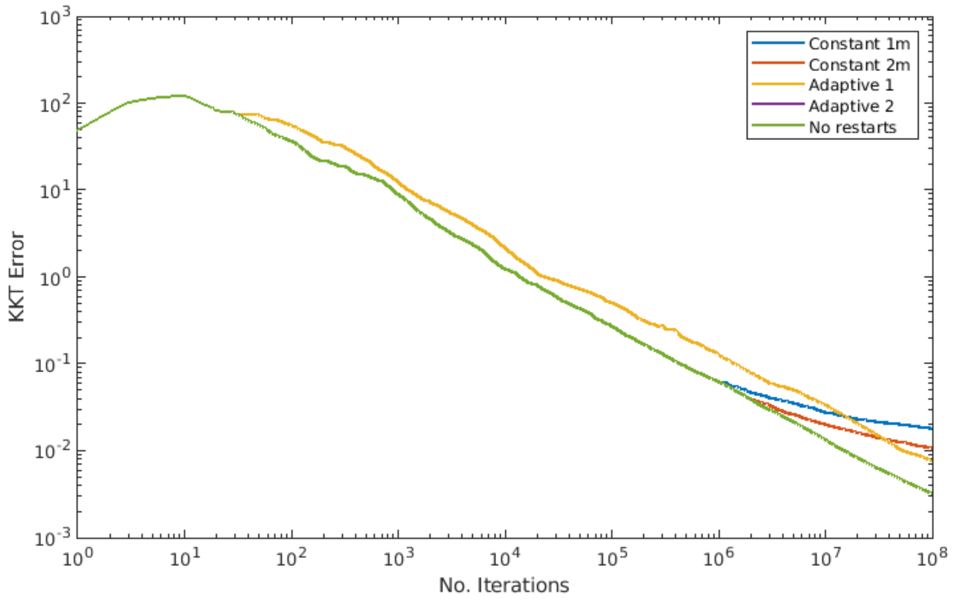


Figure 6.5. Problem 4: KKT error of averaged iterates for different restarting schemes.

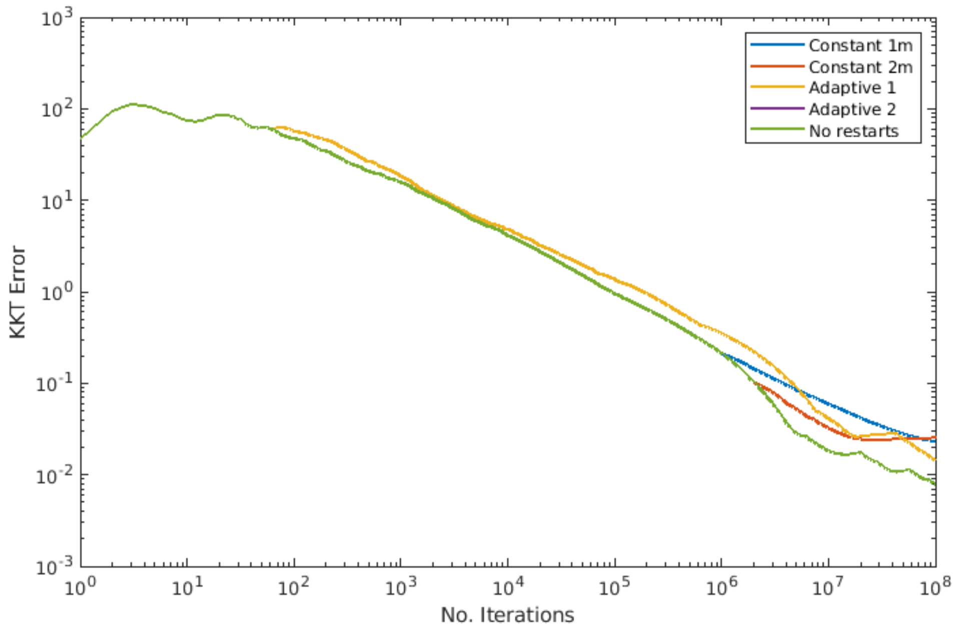


Figure 6.6. Problem 5: KKT error of averaged iterates for different restarting schemes.

6.6 Experimental analysis of convergence rate

In order to gain some insight into the convergence rate of FWLP, we perform numerical experiments comparing the KKT error of $(\boldsymbol{x}_k, \boldsymbol{y}_k)$, $(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{y}}_k)$, and the lines described by $k \mapsto C/\sqrt{k}$ and $k \mapsto C/k$, where k ranges from 1 to 100 million. C is chosen to be the KKT error of $(\boldsymbol{x}_1, \boldsymbol{y}_1)$. We use 'KKT error (avg)' in the plots below to refer to the KKT error of the averaged iterates.

The below plots suggest a worst-case convergence rate of $\mathcal{O}(1/\sqrt{k})$ for Algorithm 6.1.

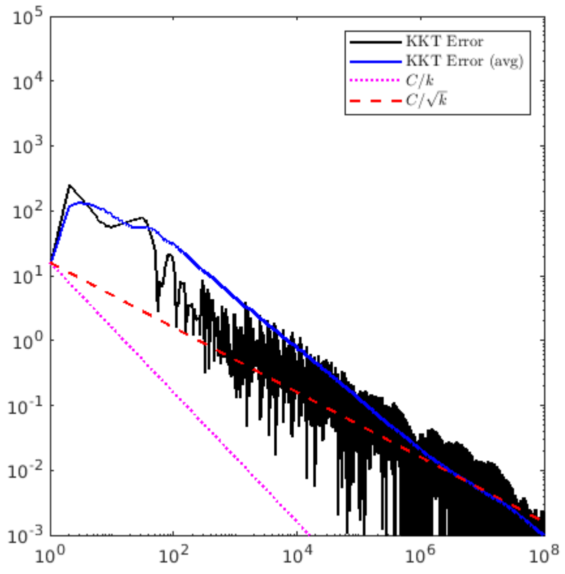


Figure 6.7. Problem 1: KKT error of last and averaged iterates compared to $C/\sqrt{k}$ and C/k.

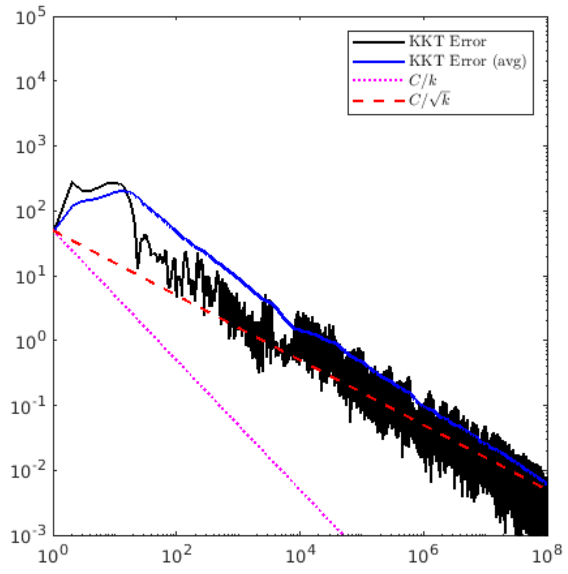


Figure 6.8. Problem 2: KKT error of last and averaged iterates compared to $C/\sqrt{k}$ and C/k.

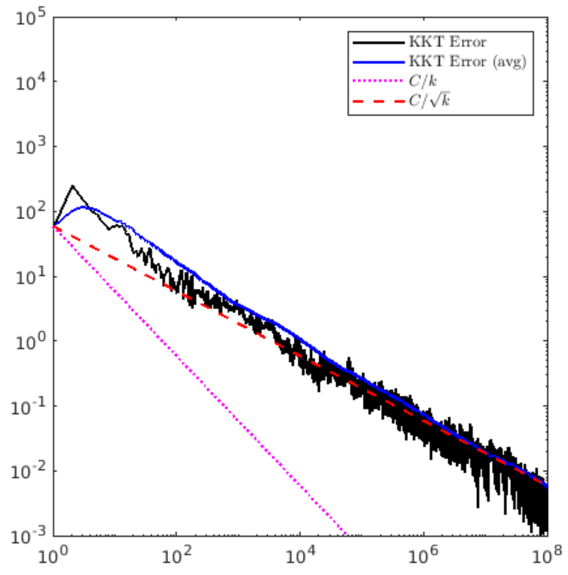


Figure 6.9. Problem 3: KKT error of last and averaged iterates compared to $C/\sqrt{k}$ and C/k.

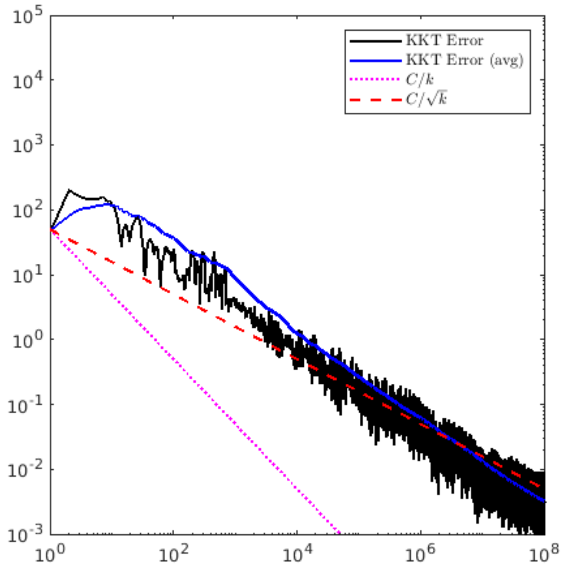


Figure 6.10. Problem 4: KKT error of last and averaged iterates compared to $C/\sqrt{k}$ *and* C/k.

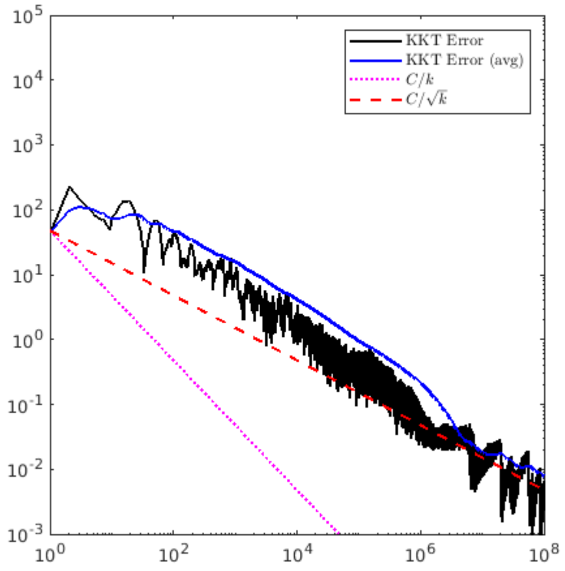


Figure 6.11. Problem 5: KKT error of last and averaged iterates compared to $C/\sqrt{k}$ *and* C/k.

Chapter 7

Conclusions and Future Work

Classical algorithms for solving linear programs require the solution of a system of linear equations at each iteration and thus do not scale well when it comes to large linear programs with billions of variables that are common in modern day data science applications. We proposed FWLP, a first-order algorithm for linear programming inspired by the Frank-Wolfe algorithm [20]. Potential functions that measure the primal feasibility, dual feasibility, and duality gap of a modified version of (LP) are shown to decrease by a constant factor at each iteration for FWLP. Moreover, these potential functions are shown to be equivalent to a constant multiple of the primal-dual gap (5.26) commonly used in the theory of saddle-point problems. While our numerical results suggest a worst-case convergence rate of $\mathcal{O}(1/\sqrt{k})$, slower than that of the best known first-order LP solver PDLP [2, 3], FWLP has the advantage that it only needs part of the matrix A at each iteration if one uses an appropriate data structure. We believe that this is a practical advantage of FWLP for very large problems. In an attempt to speed up FWLP further in practice, we tried different restart schemes similar to PDLP. However, numerical experiments seem to suggest that no speed up can be obtained by restarting the algorithm.

While numerical results suggest a worst-case convergence rate for FWLP, this thesis provides no convergence proof for the algorithm, nor does it provide an efficient implementation of FWLP based on the description of the algorithm's numerical advantages. FWLP would benefit from future research in both of these directions. Moreover, an efficient implementation of FWLP would allow for numerical experiments to be performed on larger problems, a topic that has not been investigated in this thesis.

References

[1] E.D. Andersen and K.D. Andersen. *The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm*, pages 197–232. Springer US, Boston, MA, 2000.

[2] D. Applegate, M. Díaz, O. Hinder, H. Lu, M. Lubin, B. O'Donoghue, and W. Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 20243–20257. Curran Associates, Inc., 2021.

[3] D. Applegate, O. Hinder, H. Lu, and M. Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1):133–184, Sep 2023.

[4] K. Basu, A. Ghoting, R. Mazumder, and Y. Pan. ECLIPSE: An extreme-scale linear program solver for web-applications. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 704–714. PMLR, 13–18 Jul 2020.

[5] H.H. Bauschke and P.L. Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, second edition, 2017.

[6] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[7] R.G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

[8] K.H. Borgwardt. *The Simplex Method: A Probabilistic Analysis*. Springer, 1987.

[9] M. Boroun, E.Y. Hamedani, and A. Jalilzadeh. Projection-free methods for solving nonconvex-concave saddle point problems, 2023. arXiv:2306.11944 [math.OC].

[10] Y. Cai, A. Oikonomou, and W. Zheng. Tight last-iterate convergence of the extragradient and the optimistic gradient descent-ascent algorithm for constrained monotone variational inequalities, 2022. arXiv:2204.09228 [math.OC].

[11] M.D. Canon and C.D. Cullum. A tight upper bound on the rate of convergence of the Frank-Wolfe algorithm. *SIAM Journal on Control*, 6(4):509–516, 1968.

[12] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, May 2011.

[13] A. Chambolle and T. Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1):253–287, Sep 2016.

[14] P.L. Combettes. Solving monotone inclusions via compositions of nonexpansive averaged operators. *Optimization*, 53(5-6):475–504, 2004.

[15] L. Condat. A primal–dual splitting method for convex optimization involving lipschitzian, proximable and linear composite terms. *Journal of Optimization Theory and Applications*, 158(2):460–479, Aug 2013.

[16] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, 82(2):421–439, 1956.

[17] J.C. Dunn. Rates of convergence for conditional gradient algorithms near singular and nonsingular extremals. *SIAM Journal on Control and Optimization*, 17(2):187–211, 1979.

[18] J.C. Dunn and S. Harshbarger. Conditional gradient algorithms with open loop step size rules. *Journal of Mathematical Analysis and Applications*, 62(2):432–444, 1978.

[19] J. Eckstein and D.P. Bertsekas. On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1):293–318, Apr 1992.

[20] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[21] Robert M. Freund and Paul Grigas. New analysis and results for the Frank–Wolfe method. *Mathematical Programming*, 155(1):199–230, Jan 2016.

[22] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976.

[23] P. Gács and L. Lovász. *Khachiyan's algorithm for linear programming*, pages 61–68. Springer Berlin Heidelberg, Berlin, Heidelberg, 1981.

[24] G. Gidel, T. Jebara, and S. Lacoste-Julien. Frank-Wolfe algorithms for saddle point problems. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[25] R. Glowinski and A. Marrocco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires. *Rev. Française Automat. Informat. Recherche Opérationnelle Sér. Rouge Anal. Numér.*, 9(R-2):41–76, 1975.

[26] A.A. Goldstein. Convex programming in Hilbert space. *Bulletin of the American Mathematical Society*, 70:709–710, 1964.

[27] E. Gorbunov, N. Loizou, and G. Gidel. Extragradient method: O(1/k) last-iterate convergence for monotone variational inequalities and connections with cocoercivity. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 366–402. PMLR, 28–30 Mar 2022.

[28] J. GuéLat and P. Marcotte. Some comments on Wolfe's 'away step'. *Mathematical Programming*, 35(1):110–119, May 1986.

[29] O. Güler. On the convergence of the proximal point algorithm for convex minimization. *SIAM Journal on Control and Optimization*, 29(2):403–419, 1991.

[30] J.H. Hammond. *Solving Asymmetric Variational Inequality Problems and Systems of Equations with Generalized Nonlinear Programming Algorithms.* PhD thesis, Massachussetts Institute of Technology, 1984.

[31] P. Hartman and G. Stampacchia. On some non-linear elliptic differential-functional equations. *Acta Mathematica*, 115(1):271–310, 1966.

[32] B. He and X. Yuan. On the $\mathcal{O}(1/n)$ convergence rate of the Douglas–Rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

[33] C.A. Holloway. An extension of the Frank and Wolfe method of feasible directions. *Mathematical Programming*, 6(1):14–27, December 1974.

[34] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 427–435, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[35] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[36] L.G. Khachiyan. A polynomial algorithm for linear programming. *Doklady Akademii Nauk SSSR*, 244, 1979.

[37] V. Klee and G.J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. 1972.

[38] G.M. Korpelevich. The extragradient method for finding saddle points and other problems. *Matecon*, 12:747–756, 1976.

[39] G. Lan. The complexity of large-scale convex programming under a linear optimization oracle, 2014. arXiv:1309.5550 [math.OC].

[40] E.S. Levitin and B.T. Polyak. Constrained minimization methods. *USSR Computational Mathematics and Mathematical Physics*, 6(5):1–50, 1966.

[41] X. Li, D. Sun, and KC Toh. An asymptotically superlinearly convergent semismooth newton augmented lagrangian method for linear programming. *SIAM Journal on Optimization*, 30(3):2410–2440, 2020.

[42] T. Lin, S. Ma, Y. Ye, and S. Zhang. An ADMM-based interior-point method for large-scale linear programming. *Optimization Methods and Software*, 36(2-3):389–424, 2021.

[43] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

[44] H. Lu and J. Yang. On a unified and simplified proof for the ergodic convergence rates of PPM, PDHG and ADMM, 2023. arXiv:2305.02165 [math.OC].

[45] B. Martinet. Brève communication. Régularisation d'inéquations variationnelles par approximations successives. *Revue Française d'Informatique et de Recherche Opérationnelle. Série Rouge*, 4(R3):154–158, 1970.

[46] A.S. Nemirovski. Prox-method with rate of convergence $\mathcal{O}(1/t)$ for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[47] A.S. Nemirovsky and D.B. Yudin. Effective methods for solving convex programming problems of large size. *Ékonomika i Matematicčeskie Metody*, 15, 1979.

[48] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[49] D. O'Connor and L. Vandenberghe. On the equivalence of the primal-dual hybrid gradient method and Douglas–Rachford splitting. *Mathematical Programming*, 179(1):85–108, Jan 2020.

[50] B.T. Polyak. *Introduction to Optimization*. Optimization Software, Inc., 1987.

[51] R.T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, 1970.

[52] R.T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976.

[53] N.Z. Shor. Convergence rate of the gradient descent method with dilatation of the space. *Kibernetika*, 2, 1970.

[54] M.V. Solodov. *Constraint Qualifications*. John Wiley & Sons, Ltd, 2011.

[55] D.A. Spielman and SH Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, May 2004.

[56] P. Wolfe. Convergence theory in nonlinear programming. In *Integer and Nonlinear Programming*, pages 1–36. North-Holland, 1970.

[57] S.J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, 1997.