A QUANTUM ACCELERATED APPROACH FOR THE CENTRAL PATH METHOD IN LINEAR PROGRAMMING

VIJAY ADONI
Bachelor of Technology, GITAM University, 2017
Master of Science, IIIT Hyderabad, 2019

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

A QUANTUM ACCELERATED APPROACH FOR THE CENTRAL PATH METHOD IN LINEAR PROGRAMMING

VIJAY ADONI

Date of Defence: 2023-04-05

Dr. Daya Gaur Thesis Supervisor	Professor	Ph.D.
Dr. Shahadat Hossain Thesis Examination Committee Member	Professor	Ph.D.
Dr. Robert Benkoczi Thesis Examination Committee Member	Professor	Ph.D.
Dr. John Zhang Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

To my mom.

Abstract

The central path method is a crucial technique used in the optimization of linear programs. The method relies on classical computation which hits its limit for large instances, generally used in practice, in terms of efficiency. In this thesis, a proposal is made to explore the use of quantum algorithms to enhance the central path method's performance when solving linear programs. We will go through the potential benefits and limitations of replacing the iterative equation-solving step with the HHL quantum algorithm, the Newton's step for solving a set of nonlinear equations, and converting the nonlinear set of equations to bilinear equations with the help of McCormick relaxations. The aim of this thesis is to perform extensive experimentation on several types of efficient instances using each of the proposed algorithms and to evaluate their effectiveness through numerical simulations to find a promising approach for the central path method.

Acknowledgments

I would like to extend my heartfelt thanks to Dr. Daya Gaur for his invaluable support and guidance throughout my research project. As my supervisor, Dr. Gaur has been exceptional in pushing me to explore new avenues of research and showing me the excitement of stepping out of my comfort zone. His patience is truly remarkable, and it's a quality that I aspire to emulate in my future endeavors.

I extend my sincere appreciation to Dr. Robert Benkoczi and Dr. Shahadat Hossain, my committee members, for their invaluable feedback during our on-campus discussions and conferences. Their insightful comments have been instrumental in shaping my research project, and I am grateful for their guidance and support.

In addition, I would like to thank Leila Karimi and Sajad F. Hafshejan, who have been not only friends but also mentors and colleagues during my two years at the university. Their support and camaraderie have made my academic journey eventful and memorable, and I am grateful for their contributions to my personal and professional growth.

Contents

List of Tables

List of Figures

List of Symbols

x	Vector
X	Diagonal matrix of x
$f()$	function
δ	Small constant, typically between 0 and 1
Δx	Step direction for variable x
x	Primal variable
y	Dual variable
d	Primal slack variable
q	Dual slack variable
e	Vector of 1's
H	Hessian matrix
$\nabla f()$	Gradient of function f
x^*	Optimal value of variable x
$L()$	Lagrangian function
γ	Duality gap
$\frac{\partial f()}{\partial y}$	Partial derivative of a function f w.r.t y
θ_1	Step size for primal variables
θ_2	Step size for dual variables
ε	Accuracy (used interchangeably in HHL, Newton's, and Central Path methods)
$\mathbb{R}^{nxmx...}$	Vector space of n x m x ... matrices, with real entries
$F()$	Matrix of functions
$F'()$	Jacobian matrix
x^L	Lower bound of variable x
x^U	Upper bound of variable x
$\langle \vert$	Bra notation
$\vert \rangle$	Ket notation
λ	Lagrange multiplier
λ_i	Eigenvalue indexed by an integer i
$RY_\theta()$	RY Gate
$\otimes$	Tensor product
U	Unitary matrix
U^n	$U \times U \times U ...$ n times
$e^{i\theta}$	$\cos\theta + i\sin\theta$

Chapter 1

Introduction

1.1 Introduction

Problems of the type where it is crucial to increase a given quantity given a set of restrictions are a natural occurrence. For example, resources available during wartime are often scarce, and it becomes necessary to figure out how to efficiently use the available resources, given that these resources deplete pretty rapidly. During the 1940s, similar demands of resource efficiency led to the first known modeling of a class of programs called linear programs (LPs).

An example is the production planning problem for a manufacturing company. The company wants to determine the optimal production levels of multiple products to minimize the total cost of production, subject to constraints such as available raw materials, production capacity, and demand for the products. The example given previously on a wartime scenario can be classified as a maximization problem, whereas the example on production planning is classified as the minimization problem.

Modeling of linear programs can be extended to several domains. In the healthcare domain, the LP can be used in health to model and solve problems related to diet planning, resource allocation in healthcare, and disease control. In the energy domain, an LP can be used to model and solve problems related to energy planning and management problems, such as determining the optimal generation levels of multiple power plants to meet energy demand while minimizing the cost of generation. Agriculture, finance, and many other fields have critical applications, and hence, it becomes necessary to find quick solutions.

Various techniques are available for solving LP problems, ranging from graphical to numerical optimization methods. The choice of method depends on the size and complexity of the problem, as well as the desired level of accuracy and computational efficiency. Regardless of the method used, the key to solving LP problems is to find the feasible solution space and then optimize the objective function within that space. This method requires a systematic approach and a clear understanding of the constraints and objective function.

Finding an exact solution to a Linear Programming problem can often be time-consuming. The reason for the problem being time intensive is because LP problems can be very complex and involve a large number of variables and constraints, which must then be carefully analyzed and optimized in order to find the best solution. One example of why taking a long time to find a solution to an LP problem can be bad is in the case of resource assignment in device-to-device communications underlay 5g networks. There is a lot of interference that occurs when a cellular device and a d2d pair share a spectrum. The LP problem of allocating radio resources such that the total interference is minimized seeks to mitigate our problem of interference. Finding an exact solution that may be time-consuming is not a good idea since these networks are often dynamic and optimal allocations will need to be re-calibrated much faster than the time taken to solve our LP. In such cases, we seek other ways to solve our LP.

One such approach is to find solutions that are good enough. These solutions are classified under n-approximation algorithms; where n represents how close the solution is to the optimal. In the context of LP, approximation algorithms are used to solve LP problems when finding an exact solution is not feasible or too time-consuming. This methodology for solving LPs is beneficial for NP hard problems as the approximations can be solved in polynomial time. For example, the Traveling Salesman Problem (TSP) is an NP hard problem that involves finding the shortest possible route that visits a set of cities and returns to the starting city. An approximation algorithm for the TSP, such as the Christofides algorithm, can be used to find a solution that is within a constant factor of the optimal solution.

There are several types of approximation algorithms that can be used to solve LP problems, including greedy algorithms, randomized algorithms, and heuristics. A popular method used to solve a special type of LP, called the ILP (Integer Linear Program), is the branch and bound algorithm. This algorithm divides the problem into smaller subproblems and iteratively refines the solution by adding constraints to the subproblems until the optimal solution is found.

The next approach is to try and speed up the time taken to solve an LP. Here, we aim to replace classical computations, which take a long time to solve, with quantum computations. This will show an exponential speed-up in time.

Quantum computing is an innovative approach to computing that harnesses the principles of quantum physics to process information. Unlike classical computing, which uses bits to represent data and relies on binary logic, quantum computing uses qubits, which can be in a superposition of states and entanglement to perform operations. In November 2022, IBM built the largest quantum computer (Dubbed Osprey). It has 433 qubits which are capable of holding 2^{433} states at once. The ever-growing research makes quantum computing feasible for certain types of problems, such as optimization and simulation, that are difficult or impossible to solve with classical methods. As the field continues to mature and advances are made in hardware and software, quantum methods are becoming increasingly practical for a wide range of applications.

The efficiency of solving linear systems of equations is crucial in many applications, from finance to scientific research. In classical computing, algorithms such as Gaussian elimination have a time complexity of $O(N^3)$, where N is the dimension of the system, making the process computationally expensive for large systems. However, quantum computing offers a solution to this problem with the HHL algorithm [17], created by Harrow, Hassidim, and Lloyd., which achieves an exponential speedup compared to classical algorithms by solving linear systems of equations in $O(\log N)$ time. It achieves this by utilizing quantum parallelism during the first stage of the algorithm; which will be studied in Chapter

5.

The HHL algorithm is particularly useful for solving sparse systems of linear equations, where the number of non-zero entries in the matrix is much smaller than the total number of entries. The HHL algorithm is based on the principles of quantum mechanics and leverages the power of quantum computers to provide efficient solutions to linear systems of equations.

In the following section, we will review related work in this area, examining the evolution of the methods used to solve LPs and a few quantum computing techniques for solving these problems. This review will provide a foundation for our proposed approach and help to establish the significance of our work in this field.

1.2 Related Work

Over the years, the complexity of solving linear programs has undergone substantial improvement as a result of advancements in both algorithm design and computational hardware. The key developments that have played a crucial role in this improvement will be discussed in this section.

Algorithms used for solving linear optimization programs date back to 1939 with the introduction of the graphical method. A more prominent method used frequently is the simplex method [14] introduced by George B. Dantzig. The simplex method is an algorithm for solving LP problems and finding their optimal solution. This algorithm visits the vertices of the polytope, formed by the linear problem, as it moves toward the optimal solution.

A few years forward to 2004, Deza et al. [15] state a few downfalls of applying the simplex algorithm as a means to find the optimal solution to an LP. Figure 1, also known as the Klee-Minty cube, shows the worst-case scenario with linear programming problems solved using the simplex method. The number of vertices visited is 2n, which is large when compared to the best possible case of finding the optimal solution in one vertex.

Motivated by the general limitations of the simplex method, which can be slow and

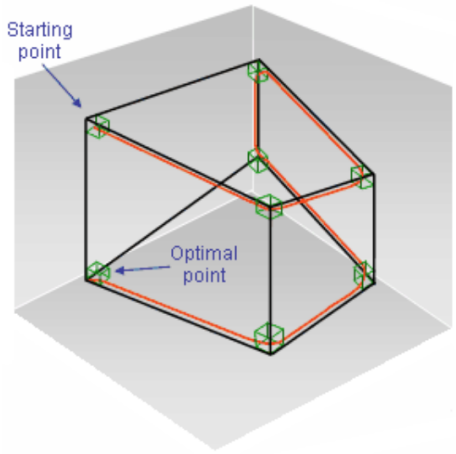


Figure 1.1: Disadvantage of Simplex. From, Antoine Deza, Eissa Nematollahi, M. Reza Peyghami, and Tamas Terlaky. The central path visits all the vertices of the klee–minty cube. Optimization Methods & Software - OPTIM METHOD SOFTW, 21, 09 2004

inefficient for large-scale linear programs. Narendra Karmarkar proposed a set of methods known as interior point methods (IPMs) [19]. Karmarkar's idea was to use a different approach to solve linear programs, one that starts from the interior of the feasible region and iteratively moves toward the optimal solution on the boundary. He called this approach the interior point method and showed that it could solve linear programs much faster than the simplex method, especially for significant problems. The algorithm shown in [19] was one of the first interior point algorithms presented in 1984. This sparked a wave of research on interior point methods, and many other algorithms have been developed since then.

The existence of an interior path, known as the central path, that converges to an optimal solution paved the way for an efficient algorithm that finds the optimal solution quicker than the simplex method. There have been several successful attempts at traversing this central path. Methods that iteratively move from the interior of the feasible region to the optimal solution on the boundary while maintaining feasibility at each step are known as feasible interior point methods. Roos et al. [29] showed that a feasible interior solution can be optimally constructed. Methods that converge to an optimal when the initial point is outside the feasible region, but a strictly positive solution, are known as infeasible IPMs. One of

the seminal papers on this method is given by Wright [33], published in 1993. In this paper, the author states that a subquadratic complexity can be achieved given that the starting point is a positive infeasible solution and the problem has a strict complementarity solution. Comparisons have been made between feasible and infeasible interior point methods. In [34], Wright states that although feasible IPMs are theoretically faster than infeasible IPMs by a factor of n, where n is the size of the input, in practice, he demonstrates that both methods can be highly effective for solving linear programs.

IPMs have been implemented with great success in recent years. In fact, major optimization software such as CPLEX and Gurobi provides the ability to use IPM to solve LPs. For the purposes of this thesis, we stick with the feasible interior point method given in [32] and discuss it in further detail in Section 3.

The first experimental realizations of quantum computing were in the late 1990s, and the first scalable quantum computer was developed in the early 21st century. The first use of quantum computing in the simulation of molecular and chemical systems was a proof-of-concept that demonstrated the potential of quantum computing in solving complex problems. Quantum computing has also been applied to other areas, such as cryptography, optimization, and machine learning. The ability of quantum computing algorithms to solve complex problems has made it a promising technology for a variety of industries, including finance, healthcare, and manufacturing. The first use of quantum computing has set the stage for many exciting developments in the field, and researchers continue exploring its potential in solving a wide range of problems.

Although quantum computers that offer a significant speedup over a classical computer do not exist, it is necessary to establish proof for a potential breakthrough. Many implementations have been proposed over the years for quantum computers that solve linear systems of equations. Barz et al. [7] were among the first to implement a quantum computer in practice with the solution vector fidelity between 64 percent and 98 percent. Cai et al. [10] outperformed this around the same time with a fidelity ranging from 82.5 percent up to 99.3

percent. Wen et al. [31] in 2018 showed the first implementation of solving an 8x8 linear system of equations.

Since the early 2020s, quantum linear system algorithms (QLSAs) have been finding applications in optimization problems. One such application is the development of quantum interior point methods (QIPMs), which have been introduced by various researchers. In 2020, Casares et al. [11] proposed a polylog algorithm for a feasible interior predictor-corrector algorithm using QLSAs. This algorithm iteratively corrects a prediction of the optimal solution, starting from an initial point in the feasible region. In 2022, Mohammadhossein et al. [23] investigated several QIPMs and proposed an infeasible interior point method that outperforms some feasible IPMs. More recently, in 2023, Zeguan et al. [35] proposed a feasible IPM with improved complexity bounds for solving ℓ_1 norm soft margin support vector machine problems. These developments showcase the potential of QLSAs in optimization and highlight the ongoing efforts in developing efficient QIPMs for practical applications.

1.3 Motivation and Contributions

In recent years, the field of quantum computing has gained significant traction owing to its potential to transform the landscape of computer science. By offering unparalleled computational power, quantum computers provide a radically different methodology to tackle intricate problems that cannot be efficiently solved on classical computers. This thesis concentrates on quantum approaches, namely quantum evolution, quantum supremacy, and quantum optimization.

The discovery of quantum entanglement is revolutionary in the field of quantum computing. Alain Aspect's [4] experimentation on this phenomenon was recognized with the Nobel Prize in Physics in 2022 and has opened up avenues for further exploration in quantum computing. The potential of quantum entanglement to produce highly intricate systems and computations with exponential complexity has generated a great deal of interest and in-

trigue.

The concept of quantum supremacy has also garnered significant attention in the quantum computing arena. Bencokzi et al. [8] have conducted a study comparing classical and quantum approaches to bitcoin mining, and their findings demonstrate the far superior performance of quantum bitcoin mining when sufficient qubits are available. The recent accomplishment by IBM, increasing their quantum computer's qubit count by eight times that of Google in just a year, is a testament to the rapid progress in the advancement of quantum computing technology.

Finally, quantum optimization is a crucial area of study in quantum computing. As stated previously, Zeguan et al. [35] have studied feasible IPM using a QLSA given by Chakraborty [12] with Qiskit Aqua. In this thesis, we aim to improve on their work by using the HHL algorithm along with the latest version of Qiskit. This has the potential to lead to significant advancements in optimization problems that cannot be efficiently solved on classical computers.

The contributions of this thesis include,

- Implement various classical approaches for the Central Path Method.
- Implement a novel modification of the Central Path Method combined with the HHL method.
- Provide extensive analysis of the Central Path Method that uses : Newton's method, McCormick relaxations and HHL method.

1.3.1 Layout

The following is the structure of this thesis:

- The thesis begins with an introduction that provides an overview of the topic, and the motivation for the study.

- The second chapter aims to familiarize readers with the list of notations that will be used throughout the thesis. This chapter also provides basic definitions that will be used as the foundation for the rest of the thesis.

- Chapter three introduces the Central Path Method and provides an intuitive explanation of the central path present in the solution space of a Linear Programming problem. It also discusses the difficulty of the problem.

- Chapter four examines various classical methods that can be used to solve the Central Path Method. This chapter also discusses the limitations and complexities of these methods.

- Chapter five focuses on important parts of the HHL algorithm and presents a modified version of the algorithm that can be used for the Central Path Method.

- The final chapter presents the results of extensive experimentation and analysis of the various methods used in the study. It includes a discussion of the strengths and weaknesses of the methods and suggests areas for future research.

Chapter 2

Preliminaries

2.1 Basic Definitions

In this section, we'll introduce some basic definitions and techniques that will be used repeatedly throughout the rest of the chapters.

2.1.1 Linear Programming

Primal problem

We will consider the maximization problem as our primal problem. It is defined as,

$$\max \; c^T x$$

$$\text{s.t.}$$

$$Ax \leq b$$

$$x \geq 0$$

where $A \in \mathbb{R}^{mxn}$, $x \in \mathbb{R}^n$ is the primal variable, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$.

Dual problem

The dual problem is a minimization problem since the primal is a maximization problem.

$$\min\ b^T y$$

$$\text{s.t.}$$

$$A^T y \geq c$$

$$y \geq 0$$

where $y \in \mathbb{R}^m$ is the dual variable.

Strong duality theorem: If the primal has an optimal solution then the dual has an optimal solution too, and the two optima are equal.

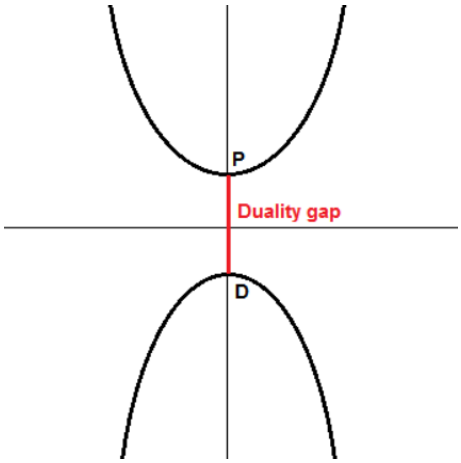


Figure 2.1: Duality Gap

Duality gap: Difference in value between any dual solution and the value of a feasible but suboptimal iterate for the primal problem as shown in Figure 2.1.

Solution Space for an LP

Figure 2.2 shows the formation of a polyhedron because of several constraints shown by the shaded blue lines.

- All extreme points are basic feasible solutions.

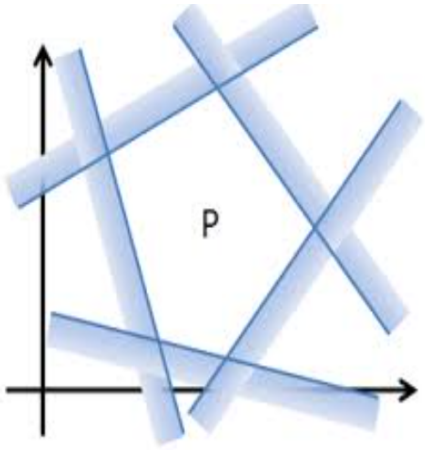


Figure 2.2: Constraints Forming a Polyhedron

- Only one of these points is the optimal solution.

Jacobian Matrix

The Jacobian matrix is a measure of how much the function distorts the space around its inputs.

For a given systen,

$$F(x,y,z) = \begin{bmatrix} f_1(x,y,z) \\ f_2(x,y,z) \\ f_3(x,y,z) \end{bmatrix}$$

The Jacobian matrix is given by,

$$F'(x,y,z) = \begin{bmatrix} \frac{\partial f_1(x,y,z)}{\partial x} & \frac{\partial f_1(x,y,z)}{\partial y} & \frac{\partial f_1(x,y,z)}{\partial z} \\ \frac{\partial f_2(x,y,z)}{\partial x} & \frac{\partial f_2(x,y,z)}{\partial y} & \frac{\partial f_2(x,y,z)}{\partial z} \\ \frac{\partial f_3(x,y,z)}{\partial x} & \frac{\partial f_3(x,y,z)}{\partial y} & \frac{\partial f_3(x,y,z)}{\partial z} \end{bmatrix}$$

Calculating Eigenvalues and Eigenvectors

For a given square matrix A, an eigenvalue is a scalar λ that satisfies the equation $Av = \lambda v$, where v is a non-zero vector known as the eigenvector. Eigenvectors and eigenvalues

are properties of square matrices

To find the eigenvalues/eigenvectors,

- To find the eigenvalues: Solve the characteristic equation $|A-\lambda I|=0$.
- To find the eigenvectors: Substitute each eigenvalue in $(A-\lambda I)v=O$ and solve for v.

Consider for example,

$$A=\begin{bmatrix}5 & 4\\ 1 & 2\end{bmatrix}$$

To find the eigenvalues:

$$\begin{bmatrix}5-\lambda & 4\\ 1 & 2-\lambda\end{bmatrix}=0$$

We get $\lambda=6,1$

To find the eigenvectors: For $\lambda=1$,

$$\begin{bmatrix}5-1 & 4\\ 1 & 2-1\end{bmatrix}*\begin{bmatrix}v_1\\ v_2\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix}$$

we get,

$$\begin{bmatrix}v_1\\ v_2\end{bmatrix}=\begin{bmatrix}-1\\ 1\end{bmatrix}$$

Similarly we get the second eigenvector for $\lambda=6$ as,

$$\begin{bmatrix}4\\ 1\end{bmatrix}$$

2.1.2 Quantum Programming

Bloch Sphere

The Bloch sphere is a geometrical representation of the state of a qubit. Any quantum state can be represented as a point on the surface of the sphere, and quantum operations can be represented as rotations around different axes on the sphere. The Bloch sphere is a useful tool for visualizing and understanding quantum algorithms.

Probability Histogram

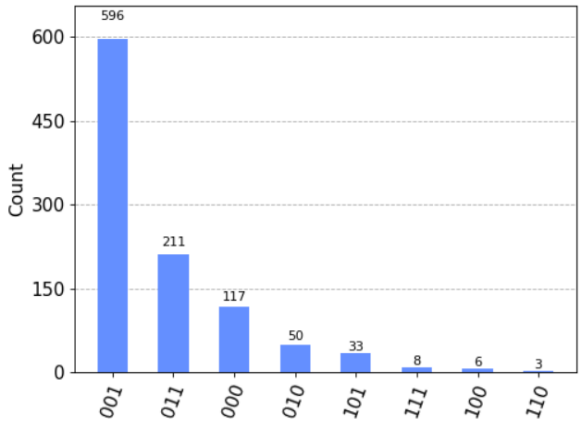


Figure 2.3: Counts vs State

We use graphs like those shown in 2.3 to analyze the result state. ”counts” shown on the y-axis refers to the number of times a particular quantum circuit is executed and the number of times each output state is observed. The x-axis shows all possible states for a given number of qubits.

Expanding Qubit Summation

Summations for the sections with quantum notation use a short form.

$$\sum_{y=0}^{N} |y\rangle = \sum_{y_1=0}^{1} \sum_{y_2=0}^{1} \dots \sum_{y_n=0}^{1} |y_1 y_2 \dots y_n\rangle$$

where, y is a decimal number expressed as $y = 2^{n-1}y_1 + 2^{n-2}y_2 + \dots + 2^0 y_n$.

Quantum Fourier Transform

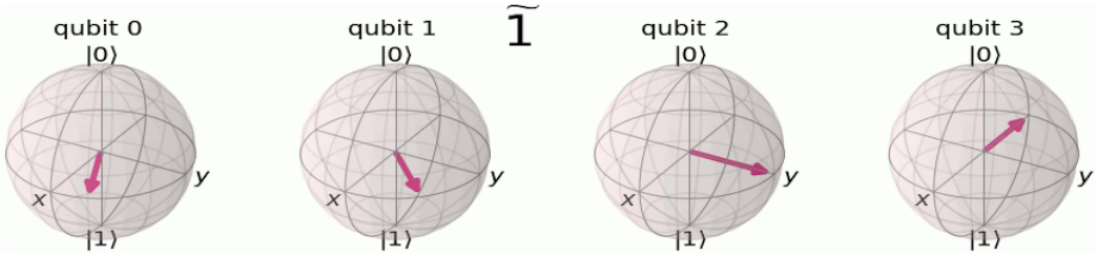


Figure 2.4: Z-Rotation for state 1

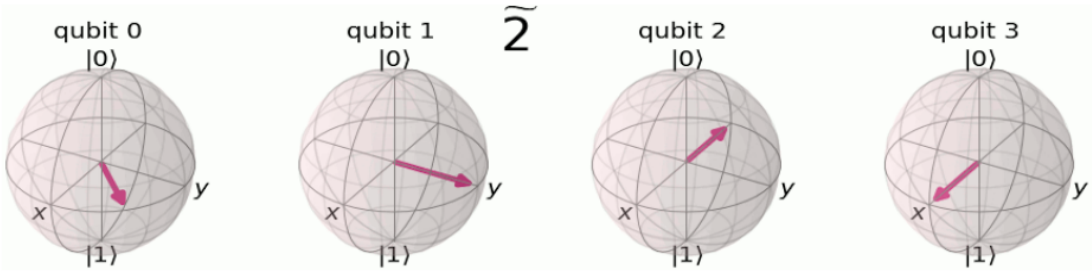


Figure 2.5: Z-Rotation for state 2

QFT can be used to transform a set of n qubits into a new set of qubits, each of which is in a superposition of states that represents a different frequency component of the input sequence. Applying QFT to a set of qubits rotates them along the z axis as shown in Figures 2.4 to 2.6.

The formulation for QFT is as follows,

$$QFT_N(x) = \frac{1}{\sqrt{N}}(|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle) \otimes (|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle) \otimes ... \otimes (|0\rangle + e^{\frac{2\pi i}{N}x}|1\rangle)$$

Inverse Quantum Fourier Transform

Undoes the effect of QFT. For example, if we were asked to find,

$$IQFT(\frac{1}{\sqrt{N}}\sum_{y=0}^{N-1} e^{2\pi i\frac{xy}{N}}|y\rangle)$$

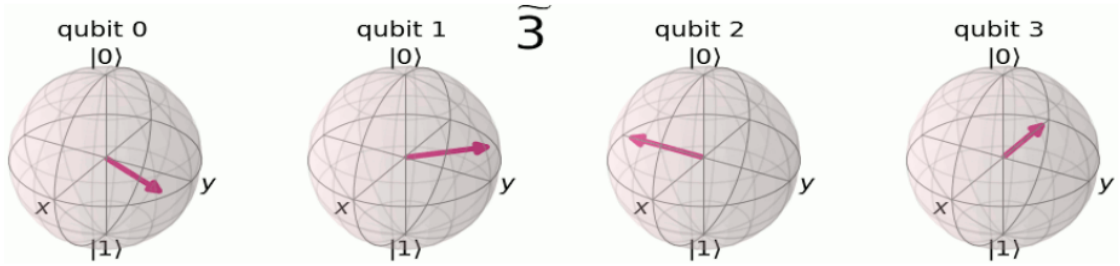


Figure 2.6: Z-Rotation for state 3

we would get $|x\rangle$ as the result since,

$$QFT(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i \frac{xy}{N}} |y\rangle$$

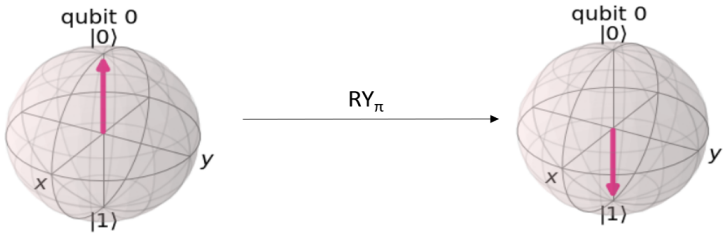
RY Gate

Rotates qubits along y-axis by an amount ψ.

$$RY_{\psi} = \begin{bmatrix} \cos\psi/2 & -\sin\psi/2 \\ \sin\psi/2 & \cos\psi/2 \end{bmatrix}$$

For example, to flip a qubit along y-axis,

$$\begin{bmatrix} \cos\pi/2 & -\sin\pi/2 \\ \sin\pi/2 & \cos\pi/2 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$



State Fidelity

The state fidelity $\mathcal{F}$ for input states ρ_1 and ρ_2 is given by [1],

$$\mathcal{F}(\rho_1, \rho_2) = tr(|\sqrt{\rho_1 \rho_2}|)^2$$

where tr() is the trace of a matrix (sum of diagonal entries).

Alternate Eigenvector Notation

A matrix G can be expressed as a linear combination of the outer products of its eigenvectors [24],

$$G = \sum_{i=0}^{N-1} \lambda_i |u_i\rangle \langle u_i| \tag{2.1}$$

Chapter 3

Optimal Solutions via the Central Path

3.1 Introduction

For the remainder of this thesis, we will consider the the maximization problem seen in Chapter 2, keeping the notations intact. This problem, which we will refer to as the standard maximization problem, aims to maximize a given linear objective subject to a set of linear constraints. Restating the problem,

$$\begin{aligned} &\max\ c^T x \\ &\text{s.t.} \\ &Ax \leq b \\ &x \geq 0 \end{aligned}$$

The decision variable x is constrained to be non-negative, and to simplify the problem, we introduce a vector of slack variables, d, to rewrite the maximization problem. The problem now becomes,

$$\max\ c^T x \tag{3.1}$$

$$\text{s.t.}$$

$$Ax + d = b \tag{3.2}$$

$$x, d \geq 0 \tag{3.3}$$

We utilize the feasible interior point approach outlined in [32] to find the maximization objective 3.1.

3.1.1 Preliminaries

We will look at the logarithmic barrier and Lagrangian functions briefly to get an idea on what a ”central path” is how it can be solved.

3.1.2 Logarithmic Barrier Function

Motivation

Suppose we wanted to rewrite our standard linear maximization problem (3.2 - 3.3) such that we remove a few of the inequalities and add them to the objective function. The new problem needs to adhere to the following form,

$$\max\ c^T x\ +\ \sum_j k(x_j)\ +\ \sum_i k(d_i) \tag{3.4}$$

s.t.

$$Ax + d = b \tag{3.5}$$

$$k(x) = -\infty \iff x < 0 \tag{3.6}$$

$$k(d) = -\infty \iff d < 0 \tag{3.7}$$

The objective function now includes additional terms, namely $\sum_j k(x_j)$ and $\sum_i k(d_i)$. As previously defined in Chapter 2, x_j and d_i are components of vectors x and d, respectively. Here, the function k is defined to be a function that approaches negative infinity when x_j or d_i are assigned non-negative values, indicating that any non-negative values for x or d would significantly improve the objective value. These conditions are represented in constraints 3.6 and 3.7 and serve as replacements for the non-negativity constraints $x \geq 0$ and $d \geq 0$, respectively.

The objective function given by 3.4 can be modified such that the function has a con-

tinuous approximation. We do this by replacing the terms $k(x)+k(d)$ with another term $\mu(l(x)+l(d))$. Here, $\mu > 0$.

$$\max\ c^T x\ +\mu\ (\sum_j l(x_j)\ +\ \sum_i l(d_i)) \tag{3.8}$$

s.t.

$$Ax+d=b \tag{3.9}$$

There are several forms the newly introduced function l can take. A few commonly used ones include exponential barrier functions, hyperbolic barrier functions, power barrier functions, logarithmic barrier functions, and so on. We follow the approach used in [32] and use a logarithmic barrier function.

$$\max\ c^T x\ +\mu\ (\sum_j \log x_j\ +\ \sum_i \log d_i) \tag{3.10}$$

s.t.

$$Ax+d=b \tag{3.11}$$

Function l is replaced by a log term in the objective given in 3.10. It can be seen that this objective function does not deviate from the objective given in 3.8 because, when any component of the vector x or d has a non-negative value, $\log x_j$ becomes $-\infty$. The problem 3.10, 3.11 may not intuitively be seen as an approximation to our standard maximization problem, but as the value of μ tends to 0, the objective function goes to $\max c^T x$ which is now equivalent to our original standard maximization problems objective.

Figure 3.1 depicts a fraction of an example solution space of the standard maximization problem. Consider the optimal solution to be at the vertex formed by the intersection of the three visible hyper planes. Three points are shown marked with "x" which show the values

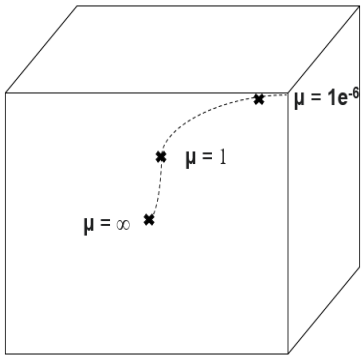


Figure 3.1: The Central Path

for the variables of the log barrier maximization problem and give a maximum objective value 3.10 for a particular value of μ. When μ is ∞, the values of variables stay at the values they were initialized at. When μ takes on a finite value of 1, the values of the variables are finite and inside the polyhedron. Finally, as the value of μ approaches 0 and takes on a minute value of $1e^{-6}$, the objective function value is a close approximation of the standard maximization problems objective. Intuition about the existence of such a path is given in 3.3.

3.1.3 Lagrangian Function

A key feature of the Central Path Method is the use of the lagrangian function [27] to construct a sequence of central points that monotonically approach the optimal solution of the standard maximization problem. For the purpose of getting an intuitive understanding of what the lagrangian function does, consider a generalized maximization problem with equality constraints. The objective function is to maximize $f(x)$ given the equality constraint $g(x) = 0$.

$$\max \ f(x)$$

s.t.

$$g(x) = 0$$

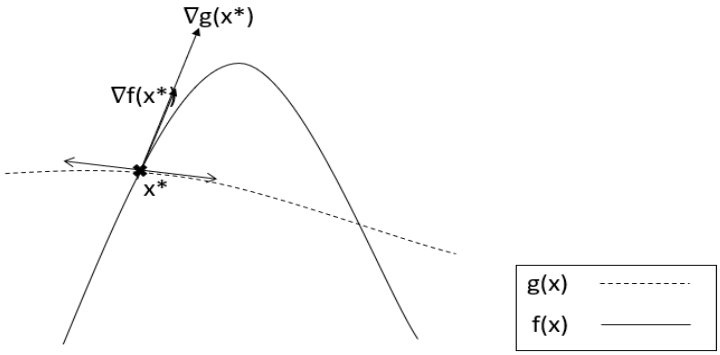


Figure 3.2: Visualization of Lagrangian Function

Figure 3.2 shows a projection of the functions $f(x)$ and $g(x)$ on a 2-D plane. It can be seen that the function $g(x)$ represented by the dotted line intersects the objective $f(x)$ at two different points. A tangent line is drawn at the point where the function $f(x)$ is maximum. The norm of $g(x)$ at this maximum point x^* is shown by the line $\nabla g(x^*)$ and $\nabla f(x^*)$ is the gradient pointing towards the direction of maximum steepness. Since $g(x) = 0$, $\nabla f(x^*)$ follows the same direction as $\nabla g(x^*)$. Therefore, $f(x^*)$ can now be considered a scaled version of $\nabla g(x^*)$.

$$\nabla f(x^*) = \lambda \nabla g(x^*)$$

where λ is known as the lagrangian multiplier.

Extending to an objective with multiple equality constraints,

$$\nabla f(x^*) = \sum_{i=1}^{m} \lambda_i \nabla g_i(x^*) \tag{3.12}$$

where f is the objective function, g_1, g_2, g_3, ... are the constraint functions, and λ_1, λ_2, λ_3, ... are the Lagrange multipliers.

A function that achieves 3.13 by setting the first order derivatives to 0 is known as the Lagrangian function L.

$$L(x,\lambda) = f(x) - \sum_{i=1}^{m} \lambda_i g_i(x) \tag{3.13}$$

Lagrange multipliers and dual variables are closely related [16] because they both describe the relationships between the variables in the original problem at the optimal solution. The Lagrange multipliers are used to express the constraints of the problem as a set of equations, while the dual variables are the coefficients of the objective function in the dual problem and reflect the importance of the constraints in the original problem. 3.13 can now be expressed as,

$$L(x,y) = f(x) - \sum_{i=1}^{m} y_i g_i(x) \tag{3.14}$$

where the Lagrange multipliers λ is now replaced by the dual variable y.

3.2 The Central Path Method

The central path is used to guide the iterative process of the interior point algorithm and to obtain the set of equations defining this path, we apply the Lagrangian function defined by 3.14 to the barrier function 3.10 - 3.11. The Lagrangian function is now given by,

$$L(x,d,y) = c^T x + \mu(\sum_j \log x_j + \sum_i \log d_j) + y^T(b - Ax - d) \tag{3.15}$$

where we substitute $g(x)$ with $y^T(b - Ax - d)$.

Obtaining critical points is done by equating the first order derivatives to 0 [32].

$$\frac{\partial L}{\partial x_j} = c_j + \mu\frac{1}{x_j} - \sum_i y_i a_{ij} = 0 \qquad \forall j = 1,2,...,n$$
$$\frac{\partial L}{\partial d_i} = \mu\frac{1}{d_i} - y_i = 0 \qquad \forall i = 1,2,...,m$$
$$\frac{\partial L}{\partial y_i} = b_i - \sum_j a_{ij}x_j - d_i = 0 \qquad \forall i = 1,2,...,m$$

which can then be simplified to,

$$Ax + d = b \tag{3.16}$$
$$A^T y - q = c \tag{3.17}$$
$$XQe = \mu e \tag{3.18}$$
$$YDe = \mu e \tag{3.19}$$

where we substitute q in place of $\mu X^{-1}e$.

In exploring the Central Path Method algorithm, the initial step involves starting with an arbitrary point (x,y,d,q). Since we use feasible IPMs, the starting vectors must be either primal or dual feasible. Another crucial requirement is that for each vector in the point (x,y,d,q), the vector must be non-negative.

The next step in the algorithm involves determining the direction in which the point must travel to converge towards the central path. This can be challenging since it requires an understanding of the specific location on the central path where the point must end up. The parameter μ is essential in identifying this location, and selecting an appropriate value of μ is crucial to ensure convergence towards the correct point. Figure 3.3 demonstrates the outcome of selecting a high value of μ, which leads to the analytic center, while selecting a low value may cause the point to hit the boundary of the feasible region.

A reasonable value for μ is,

$$\mu = \delta\frac{\gamma}{n+m}$$

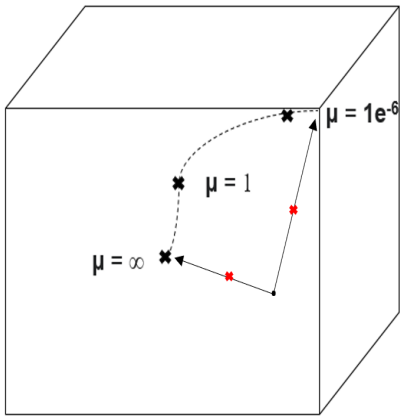


Figure 3.3: Choosing μ

where δ is about 0.1 and γ is the duality gap.

When the final convergence location is identified, the next step is to determine the direction $(\Delta x, \Delta y, \Delta d, \Delta q)$ that will bring the new point $(x+\Delta x, y+\Delta y, d+\Delta d, q+\Delta q)$ closer to the final point. To achieve this, equations 3.16 - 3.19 can be modified by replacing x with Δx, and so forth for the other variables. This results in a set of equations that require solving.

$$A\Delta x + \Delta d = \rho \tag{3.20}$$

$$A^T \Delta y - \Delta q = \sigma \tag{3.21}$$

$$Q\Delta x + X\Delta q + \Delta x \Delta q e = \mu e - XQe \tag{3.22}$$

$$D\Delta y + Y\Delta d + \Delta y \Delta d e = \mu e - YDe \tag{3.23}$$

where $\rho = b - Ax - w$, $\sigma = c - A^T y + z$, and $\gamma = z^T x + y^T w$.

Subsequently, the point can be moved in the determined direction by an appropriate amount that ensures the non-negativity of (x, y, d, q). The complete procedure is presented in Algorithm 1. We note that the step size for primal variables θ_1 is different from the step size for dual variables θ_2 as it results in faster convergence.

Algorithm 1: The Central Path Algorithm

Input: $c \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$

Output: x

Set: $(x, y, w, z) > 0$

while $\gamma \geq \varepsilon$ **do**

- Set: $\rho = b - Ax - w$
- Set: $\sigma = c - A^T y + z$
- Set: $\gamma = z^T x + y^T w$
- Set: $\mu = \delta \frac{\gamma}{n+m},\ 0 < \delta < 1$
- Compute step directions using
- $A\Delta x + \Delta d = \rho$
- $A^T \Delta y - \Delta q = \sigma$
- $Q\Delta x + X\Delta q + \Delta x \Delta q e = \mu e - XQe$
- $D\Delta y + Y\Delta d + \Delta y \Delta d e = \mu e - YDe$
- Compute step size using
- $x = x + \theta_1 \Delta x$
- $w = w + \theta_1 \Delta w$
- $y = y + \theta_2 \Delta y$
- $z = z + \theta_2 \Delta z$

3.3 Discussion

Hardness

An optimal solution can be obtained through iteratively solving the nonlinear set equations 3.20 to 3.23. Upon examining Algorithm 1 in its entirety, it is apparent that computing the step direction is the most challenging aspect of the procedure, while the remaining steps primarily involve comparison or assignment operations.

Various approaches can be employed to solve these equations, including the use of New-

ton's step to approximate the nonlinear inequality as a linear inequality, utilizing Gurobi's built-in non-convex solver, and adopting a quantum algorithm to attain exponential speedup. These methods will be studied in the subsequent chapters.

Existence of Central Path

The equations characterising the central path may not always have a solution. It is easy to see that any example with an unbounded barrier function will not have a solution. One such example is as follows,

$$\begin{aligned} &\max\ 0 \\ &\text{s.t.} \\ &x \geq 0 \end{aligned}$$

The objective of this example is to maximize a constant value given the decision variable x is positive. The barrier function can then be written as $\mu \log x$, which doesn't have a bound on the maximum. In general, the following theorem given in [32] helps in identifying if a given barrier problem has a solution.

Theorem 3.1. *There exists a solution to the barrier problem if and only if both the primal and the dual feasible regions have nonempty interior*

Chapter 4

Solving the Central Path Equations

This chapter will revisit the nonlinear set of equations given in 3.2. We examine the complexities involved in solving nonlinear sets of equations and how it differs from the process of solving linear equations. Traditional methods for solving linear equations, such as substitution and elimination, are ineffective for nonlinear equations. Instead, we must resort to alternative methods to find solutions.

To motivate the need for effective methods to solve nonlinearities, we will consider an example with two variables, x and y, for the sake of understanding. Let a system of equations be defined by,

$$3x + 8y = 3$$

$$x + 5y = 7$$

Figure 4.1 shows a plot of these linear equations. It can be seen that a solution to this set of equations is at the intersection of the two lines of the plot; i.e., we find the solution to be $x = -5.857 and y = 2.571$. Continuing the reasoning to a nonlinear set of equations,

$$2x + 7y + 3xy = 7$$

$$7x + y + 5xy = 3$$

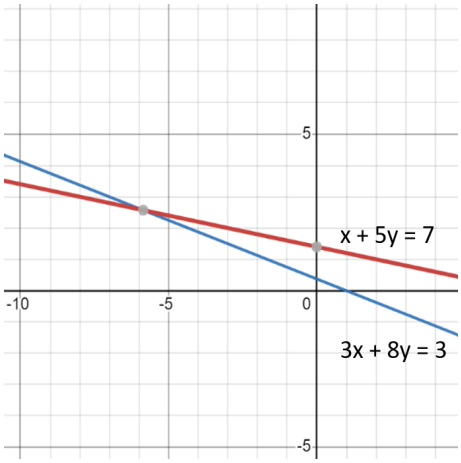


Figure 4.1: Solution to a linear set of equations

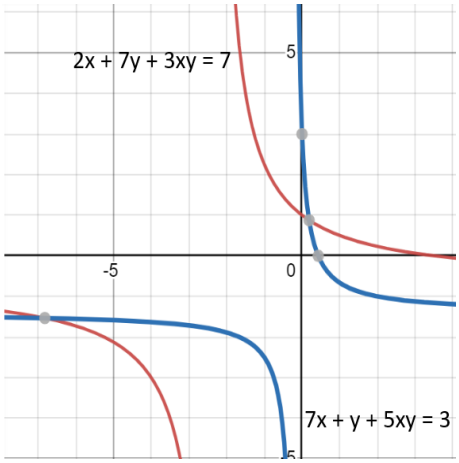


Figure 4.2: Solution to a nonlinear set of equations

Figure 4.2 immediately shows us that a solution to the nonlinear set of equations is not clear. Additional bounds should be known to get a unique solution. For example, if we add on a constraint that states that the variables *xandy* must be strictly positive, we find the solution $x = 0.187$ and $y = 0.877$.

Graphical methods, like the one used in the example above, involve plotting the equations on a graph and finding the point of intersection. However, this method can be hard to realize in practice and is only sometimes accurate. For the purposes of this thesis, we look at nonlinear approximation methods and choose two broadly used variants.

A detailed description of these methods is given in the following section.

4.1 Nonlinear Approximations

Several methods have been developed over the years that can be used to solve nonlinear equations, including graphical methods, iterative methods, substitution methods, and numerical methods. As stated previously, graphical methods are hard to realize in practice as the number of variables grows. Substitution methods involve expressing one variable in terms of another variable in order to solve for it, which can be more complex than linear equations but can be applicable in some cases where equations are solvable in terms of elementary functions. Numerical methods use a computer to approximate the solution by generating a large number of points on the graph of the equation. These methods are very accurate but require significant computational power and can be time-consuming.

Iterative methods involve starting with an initial guess for the solution and repeatedly refining the guess until it is close enough to the real solution. A renowned method that uses this methodology is the Newton-Raphson method [13]. The Newton-Raphson method uses a multivariate Taylor series expansion to approximate the root of a nonlinear equation. The use of a simplified version of this method, known as the Newton method, in the Central Path Method is given in 4.1.1. Another iterative method is the bisection method [18]. It involves dividing the interval that contains the solution into two halves and repeatedly narrowing down the interval until a solution is found. The method is based on the Intermediate Value Theorem, which states that if a continuous function $f(x)$ changes sign over an interval $[a,b]$, then there must be a solution in that interval.

In addition to the iterative methods, we take a look at substituting the nonlinear curves with piecewise linear lines. Here, we divide the domain of the nonlinear equation into small intervals and approximate the equation with a straight line in each interval. This results in a series of linear equations, which can be easily solved using standard mathematical techniques. One such technique is with the use of McCormick relaxations [22].

4.1.1 Newton's Approximation

We will begin by providing a concise overview of Newton's method given in [32], followed by a practical example to demonstrate its application. We will then apply this method to our specific problem domain by replacing Equations 3.20 through 3.23 with their respective Newton approximations.

Newton's method is based on the idea of approximating a function using its tangent line at a given point and then finding the root of the tangent line equation. This process is repeated iteratively, with each new estimate of the root being used to generate a better approximation.

The general formula for Newton's Method is:

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)} \tag{4.1}$$

Where x_n is the current estimate of the root, f(x_n) is the function evaluated at x_n, and f'(x_n) is the function's derivative evaluated at x_n. The formula generates a new estimate of the root, x_{n+1}, which is closer to the actual root of the function.

Numerical Example

Let us consider an example where we need to solve a system of two nonlinear equations.

$$x + 2y + 4xy - 5 = 0$$

$$2x + y + 3xy - 5 = 0$$

We can write the function and its jacobian as,

$$F = \begin{bmatrix} x + 2y + 4xy - 5 \\ 2x + y + 3xy - 5 \end{bmatrix}, \; F' = \begin{bmatrix} 1 + 4y & 2 + 4x \\ 2x + y & 1 + 3x \end{bmatrix}$$

Using Equation 4.1 and choosing our starting point as $(x,y) = (-50,-50)$, Step 1:

$$[x',y'] = [-50,-50] - \begin{bmatrix} 1+4y & 2+4x \\ 2x+y & 1+3x \end{bmatrix}^{-1} \times \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$(x',y') = (-13.703,-36.757)$$

We now use this new point and solve 4.1. We find the roots of our system in six steps,

- Step 1: (x',y') = (-13.703, -36.757)
- Step 2: (x',y') = (-6.788, -19.472)
- Step 3: (x',y') = (-3.368, -10.922)
- Step 4: (x',y') = (-1.73, -6.825)
- Step 4: (x',y') = (-1.034, -5.086)
- Step 5: (x',y') = (-0.838, -4.596)
- Step 6: (x',y') = (-0.82, -4.55)

One of the main advantages of Newton's Method is its rapid convergence rate, especially for well-behaved functions. This makes it a popular choice for solving nonlinear equations in our thesis. There are, however, a few limitations to the method, such as the possibility of the method failing to converge or converging to a wrong root. Therefore, it is essential to carefully choose the initial guess and test the method's performance for different scenarios.

We will slightly change the notations for convenience when analyzing Newton's method applied to the Central Path Method. Let a be our starting point characterized by (x,y,d,q).

We go back to Equations 3.16 through 3.19. We define $F(a)$ as,

$$F(a) = \begin{bmatrix} Ax + d - b \\ A^T y - q - c \\ XQe - \mu e \\ YDe - \mu e \end{bmatrix}$$

and therefore,

$$F'(a) = \begin{bmatrix} A & 0 & I & 0 \\ 0 & A^T & 0 & -I \\ Q & 0 & 0 & X \\ 0 & D & Y & 0 \end{bmatrix}$$

Substituting $x_{n+1} = \Delta a$ and $x_n = a$, Equation 4.1 changes to,

$$F'(a)\Delta a = -F(a), \tag{4.2}$$

where

The final set of equations can be obtained by simplifying 4.2,

$$A\Delta x + \Delta d = \rho \tag{4.3}$$

$$A^T \Delta y - \Delta q = \sigma \tag{4.4}$$

$$Q\Delta x + X\Delta q = \mu e - XQe \tag{4.5}$$

$$D\Delta y + Y\Delta d = \mu e - YDe \tag{4.6}$$

where, $\rho = b - Ax - w$ and $\sigma = c - A^T y + z$.

Here, it is apparent that Equations 3.20 - 3.20 resemble Equations 4.3 - 4.3, with the exclusion of non-linear terms.

Algorithm 2: The Linear Central Path Algorithm

Input: $c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$

Output: x

Set: $(x, y, w, z) > 0$

while $\gamma \geq \varepsilon$ **do**

Set: $\rho = b - Ax - w$

Set: $\sigma = c - A^T y + z$

Set: $\gamma = z^T x + y^T w$

Set: $\mu = \delta \frac{\gamma}{n+m}$, $0 < \delta < 1$

Compute step directions using

$$\begin{bmatrix} A & 0 & I & 0 \\ 0 & A^T & 0 & -I \\ Q & 0 & 0 & X \\ 0 & D & Y & 0 \end{bmatrix} \times \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta d \\ \Delta q \end{bmatrix} = - \begin{bmatrix} \rho \\ \sigma \\ XQe - \mu e \\ YDe - \mu e \end{bmatrix}$$

Compute step size using

$x = x + \theta_1 \Delta x$

$w = w + \theta_1 \Delta w$

$y = y + \theta_2 \Delta y$

$z = z + \theta_2 \Delta z$

4.1.2 McCormick Relaxations

Piecewise approximations are a common technique used to approximate nonlinear equations that are difficult to solve analytically. This method involves dividing the domain of the

equation into smaller subdomains and approximating the equation within each subdomain using a more straightforward, linear function. The resulting piecewise function approximates the original nonlinear equation that can be used for numerical simulations and analysis. This approach is widely used, and a common relaxation used here is the McCormick relaxation [20].

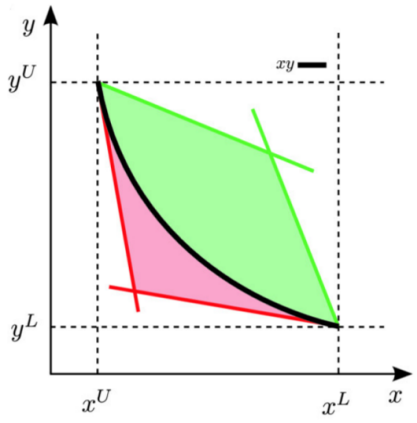


Figure 4.3: McCormick Relaxation

We focus on inequalities with quadratic terms of the form xy, where x and y are variables. Although various other types of non-convex curves are possible in nonlinear programming problems, we do this because the equations we have seen in Section 3.2 have this form. Figure 4.3 shows the curve $f(xy)$ outlined in black. With predetermined knowledge about the bounds on the variables x and y, we can generalize a nonlinear maximization problem to,

$$\begin{aligned} &\max f(xy) \\ &\text{s.t,} \\ &g(xy) \leq 0 \\ &x^L \leq x \leq x^U \\ &y^L \leq y \leq y^U \end{aligned} \tag{4.7}$$

(4.8)

Where x^L and y^L are the lower bounds and x^U and y^U are the upper bounds on x and y, respectively. The bounds on x and y may not naturally exist in a problem, prompting the need to impose bounds manually. As a result, multiple regions with local maximums may arise, and selecting the global maximum would be necessary.

In order to eliminate nonlinearities, we introduce the variable $w = xy$ and employ the techniques outlined in [22] to convert nonlinear inequalities into linear ones. This involves using the non-negative constraints on variables x and y, and by intuition, we know that the product of these constraints is also non-negative. As an illustration, one set of constraints can be represented as $x^U - x \geq 0$ and $y^U - y \geq 0$, which results in the inequality $w - x^U y - y^U x + x^U y^U \geq 0$. Considering all such constraints, we get,

$$w \geq x^L y + xy^L - x^L y^L \tag{4.9}$$

$$w \geq x^U y + xy^U - x^U y^U \tag{4.10}$$

$$w \leq x^U y + xy^L - x^U y^L \tag{4.11}$$

$$w \leq x^L y + xy^U - x^L y^U \tag{4.12}$$

Equations 4.11 and 4.12 are known as the over-estimators of the curve given by $f(xy)$

as they place an upper bound on the function. Equations 4.9 and 4.10 are known as the under-estimators of the curve given by $f(xy)$ as they place a lower bound on the function.

Next, we determine the limits for the step direction variables Δx, Δy, Δd, and Δq. Following this, we have seen the limits for the new points as x', y', d', and q', obtained by moving in the step direction, where $x' = x + \theta\Delta x$, $y' = y + \theta\Delta y$, $d' = d + \theta\Delta d$, and $q' = q + \theta\Delta q$. We will start by deriving the bounds for one variable and then extend the analysis to all points.

We know that,

$$x + \theta\Delta x > 0,$$

where x is non-negative and $0 \leq \theta \leq 1$. This gives a lower bound to x as,

$$-x \leq \Delta x$$

Finding the upper bound is tricky as we do not have much more information. We, therefore, consider some constant U_x to be a manual upper bound to $x + \theta\Delta x$. Our upper bound can then be written as,

$$\Delta x \leq U_x - x$$

Similarly,

$$-y \leq \Delta y \leq U_y - y$$

$$-d \leq \Delta d \leq U_d - d$$

$$-q \leq \Delta q \leq U_q - q$$

Equations 4.9 - 4.12 can now be written as,

$$w_{\Delta y \Delta d} \geq (-y)\Delta d + \Delta y(-d) - (-y)(-d)$$
$$w_{\Delta y \Delta d} \geq (U_y - y)\Delta d + \Delta y(U_d - d) - (U_y - y)(U_d - d)$$
$$w_{\Delta y \Delta d} \leq (U_y - y)\Delta d + \Delta y(-d) - (U_y - y)(-d)$$
$$w_{\Delta y \Delta d} \leq (-y)\Delta d + \Delta y(U_d - d) - (-y)(U_d - d)$$

and,

$$w_{\Delta x \Delta q} \geq (-x)\Delta q + \Delta x(-q) - (-x)(-q)$$
$$w_{\Delta x \Delta q} \geq (U_x - x)\Delta q + \Delta x(U_q - q) - (U_x - x)(U_q - q)$$
$$w_{\Delta x \Delta q} \leq (U_x - x)\Delta q + \Delta x(-q) - (U_x - x)(-q)$$
$$w_{\Delta x \Delta q} \leq (-x)\Delta q + \Delta x(U_q - q) - (-x)(U_q - q)$$

The choice of U_x is crucial. We take a simple example to understand the effects on commercial solvers when bounds are unnecessarily large on a simple LP where the solution is otherwise trivial. Table 4.1 shows the time taken so solve the following LP,

$$\max x + 6y$$
$$\text{s.t,}$$
$$xy \leq 10$$

As the upper bound decreases by a factor of ten each time with a starting bound of 1000, we see that the time taken to get a solution decreases significantly.

Table 4.1: Upper Bound vs Time for Nonlinear Approximations

Upper Bound	Time Taken (s)
1000	0.06
100	0.021
10	0.011
1	0.005

4.2 Discussion

Drawbacks of Nonlinear Approximations

Nonlinear approximations have a few drawbacks that make them unsuitable for specific problems. The first drawback is that nonlinear approximations can be computationally expensive, especially for large and complex systems. Next, they may not always provide the most accurate results. In addition, the convergence rate of nonlinear approximations can be slow, which can be a significant limitation. Finally, as stated above, it is not always easy to identify the upper and lower bounds of the variables of a system. Hence, there might be a need for manual intervention.

One can argue, however, that because a better direction is obtained with nonlinear approximations when compared to linear, it might go towards the solution quicker in some specific cases.

Using Gurobi

Gurobi is an efficient optimization software capable of handling complex nonlinear problems. In particular, when manually implementing McCormick relaxations, we can use its built-in capabilities for obtaining upper and lower bounds during the branch-and-bound process. More information about our use of Gurobi and the specifics of our approach will be provided in Chapter 6.

Complexity Analysis for Newton's Method

Common methods used to solve the linear set of equations given in 2 include the Gaussian elimination method which requires storing the entire matrix of coefficients in memory along with intermediate results obtained during multiple passes through the matrix. Therefore, the complexity of the Newton method in IPM can be expressed as $O(n^3L)$, where n is the number of variables in the problem and L is the number of iterations required to find the solution within the desired tolerance level. This is because the Newton method requires the solution of a linear system at each iteration, and the cost of solving a linear system is typically $O(n^3)$. This shows a significant growth in the time requirement when using classical algorithms. There is therefore a need to turn to quantum algorithms for an exponential speedup.

Chapter 5

Quantum Accelerated Central Path Method

5.1 Introduction

We have seen in Chapter 4 a method for solving a linear programming problem using Newton's method. The main takeaway from the method involved ignoring the nonlinear terms that characterize the central path. This led to a set of linear equations (4.3 - 4.6) that need to be solved iteratively given an initial starting point x, y, d, q in order to find and converge to the optimal solution.

One way to speed up the central path method is to use quantum computing to perform certain computations. For example, the quantum version of the interior point method (qIPM) [23] can be used to solve SDPs more efficiently than the classical interior point method. The qIPM uses quantum computers to perform certain linear algebraic operations, such as matrix inversion and eigenvalue computation, which can be much faster than their classical counterparts.

Our proposed solution is to use the HHL algorithm to solve the Newton step (4.3 - 4.6). We first start with an overview of the HHL algorithm and then explain how we intend to integrate the algorithm with the Central Path Method.

5.2 HHL Overview

The HHL algorithm has been implemented on various quantum computing platforms, such as superconducting qubits and trapped ions. However, the practical implementation of

the HHL algorithm is still challenging due to the limitations of current quantum computing technology. Nevertheless, the HHL algorithm is considered one of the most promising quantum algorithms for solving systems of linear equations.

The algorithm begins by encoding the matrix and vector of the system of linear equations into the quantum state of a quantum computer. Then, by applying a series of quantum operations, the algorithm efficiently extracts a quantum state that encodes the solution of the system. This is achieved by applying a series of quantum operations, including a quantum Fourier transform, a controlled rotation operation, and a final measurement step.

The inputs currently restrict the use of the HHL algorithm to solve a linear system of equations. There are several main considerations for the inputs. These include:

1. Matrix type: The input matrix should be Hermitian because it is required to have real eigenvalues in order to perform phase estimation accurately. In addition to this, the time evolved version of a Hermitian matrix is unitary making it optimal to use as a quantum gate.

2. Matrix condition number: The condition number of the matrix is an important consideration. A well-conditioned matrix has a low condition number and is easier to invert using the HHL algorithm, while a poorly conditioned matrix can result in a larger number of auxiliary qubits and longer runtimes.

3. Number of registers available: The HHL algorithm provides an approximate solution, with a precision that depends on the number of auxiliary qubits and the accuracy of the phase estimation step.

4. Matrix sparsity: The sparsity of the matrix A with only a few non-zero entries. It also affects the number of gates required for the algorithm, and the overall runtime.

A linear system of equations can be represented using quantum notation by the follow-

ing,

$$|q\rangle = G^{-1}|r\rangle$$

Writing the matrix G and vector r in the eigenbasis of G, the equation can be rewritten as,

Since G is a diagonal matrix in its eigenvector basis the inverse of 2.1 is given by,

$$G^{-1} = \sum_{i=0}^{N-1} \lambda_i^{-1} |u_i\rangle \langle u_i|$$

$|r\rangle$ can also be represented in the eigenbasis of G as

$$|r\rangle = \sum_{i=0}^{N-1} q_j |u_j\rangle$$

and so,

$$|q\rangle = \sum_{i=0}^{N-1} \lambda_i^{-1} r_i |u_i\rangle \tag{5.1}$$

where N is the size of vector r and u_i denotes the i^{th} eigenvector of G. $|q\rangle$ is also known as the eigenstate of matrix G.

Equation 5.1 gives a requirement of constructing a circuit that determines and reverses the eigenvalues of the matrix G. This will result in obtaining a state in place of $|r\rangle$, which should be consistent with the solution to the linear system of equations. One such circuit that achieves this is given in Figure 5.1.

Figure 5.1 depicts three primary stages of the HHL algorithm. In the initial phase, we execute the quantum phase estimation algorithm. Given its widespread utilization and significance, a dedicated section, Section 5.3, has been allocated to discuss this subroutine thoroughly. The main idea of this stage is to obtain the eigenvalues of matrix G and store

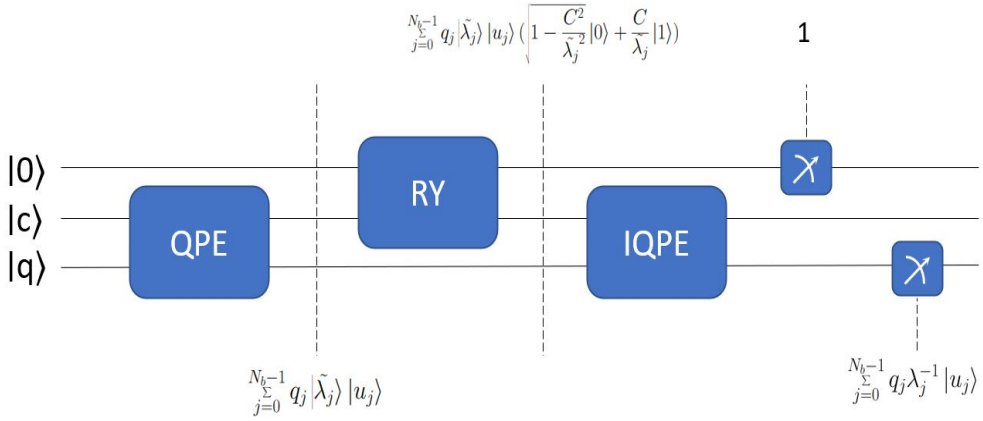


Figure 5.1: HHL Circuit Diagram

them in the auxiliary register. The state is in the eigenvector basis of G.

The second stage of the HHL algorithm is the inversion stage. In classical algorithms, computing the inverse of a matrix is a time-consuming task, especially for matrices of large size. However, in quantum algorithms, matrix inversion can be performed much more efficiently using an *RY* gate conditioned on the eigenvalues.

$$\begin{bmatrix} \cos\psi/2 & -\sin\psi/2 \\ \sin\psi/2 & \cos\psi/2 \end{bmatrix}$$

where $\psi = 2\arcsin\frac{C}{\lambda}$; for some constant C bounded by the smallest eigenvalue. A measurement of 1 in the auxiliary bit after applying the *RY* gate gives us the inverse eigenvalues. The matrix for RY gate can now be represented as,

$$\begin{bmatrix} \sqrt{1-\frac{C^2}{\lambda_j^2}} & -\frac{C}{\lambda_j} \\ \frac{C}{\lambda_j} & \sqrt{1-\frac{C^2}{\lambda_j^2}} \end{bmatrix}$$

Therefore,

$$RY_\psi(|0\rangle) = \sum_{j=0}^{N-1} r_j \left|\lambda_j\right\rangle \left|u_j\right\rangle \left(\sqrt{1-\frac{C^2}{\lambda_j^2}}\,|0\rangle + \frac{C}{\lambda_j}\,|1\rangle\right)$$

The final stage of the HHL algorithm involves converting the result in the eigenvector basis to the computational basis using the inverse Fourier transform. If, after applying the *RY* gate, the measurement outcome is not equal to 1, the entire process must be repeated to achieve an accurate result.

5.3 Quantum Phase Estimation

The main driver of the HHL algorithm is the quantum phase estimation (QPE) subroutine. QPE is a fundamental concept in quantum computing that is used in identifying the phase of a given unitary operator. This phase is, more often than not, the eigenvalues of a matrix. This allows for the determination of the eigenvectors of the matrix, which aids in determining the solution of a linear system of equations. The QPE subroutine is run multiple times to obtain an estimation of the eigenvalues with high precision.

QPE forms the basis for a wide range of quantum algorithms. One such algorithm is Shor's algorithm [30]. This algorithm is considered one of the most critical and groundbreaking algorithms in quantum computing. It is an efficient quantum algorithm that can solve the problem of prime factorization, which is a crucial tool in the cryptography field. Another widely used algorithm that takes advantage of this subroutine is the quantum counting algorithm [9]. This algorithm given by Gilles Brassard et al. is a crucial capability in

the field of quantum computing, as it enables the efficient use of Grover's search algorithm.

The QPE algorithm begins by preparing a quantum state known as the eigenstate of a matrix as shown in 5.1, which encodes information about the matrix's eigenvalues. This state is then applied to time evolved hermitian matrices (Unitary matrices), and a quantum Fourier transform is applied to the state. This allows for the extraction of the phase information of the eigenvalues, which can be used to estimate them. The precision of the eigenvalue estimation depends on the number of qubits used in the QPE algorithm.

Given its widespread usage as a subroutine in quantum computing, we will provide a comprehensive overview of the implementation details of quantum phase estimation in the following subsection while refraining from delving into the mathematical derivations. The focus will be on providing a clear and thorough understanding of the practical aspects of QPE, making it accessible for the HHL algorithm to use.

5.3.1 Implementation

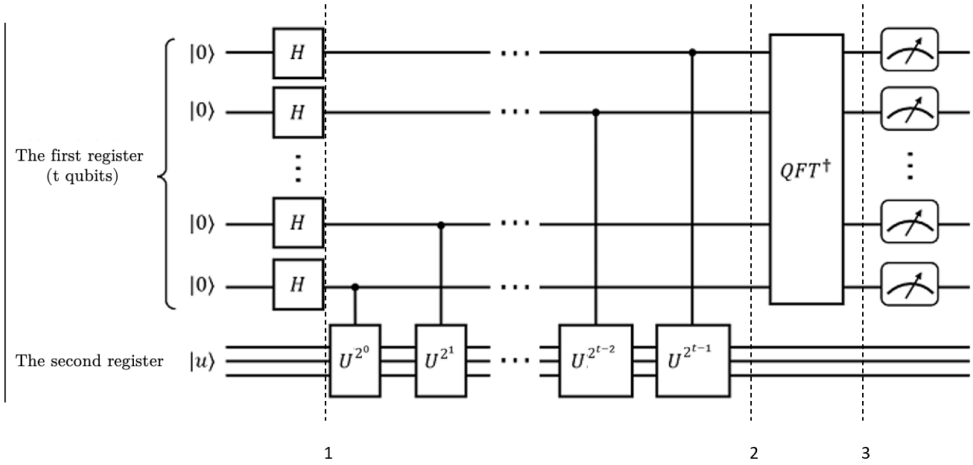


Figure 5.2: QPE Circuit Diagram. Adapted from,
https://www.mindspore.cn/mindquantum/docs/en/r0.6/quantum_phase_estimation.html

QPE utilizes the concept of phase kickback to encode the phase of the unitary operator U in the Fourier basis onto t qubits within the counting register. The inverse quantum

Fourier transform (QFT') is then employed to translate this information from the Fourier basis into the computational basis. Applying an appropriate measurement makes it possible to determine the eigenphase with high precision, even if we do not know the exact form of the operator U.

Figure 5.2 shows a generalized quantum phase estimation circuit. Initially, the algorithm starts with a quantum input state $|u\rangle$ of an arbitrary size N, and t qubits of auxiliary bits. These auxiliary qubits serve as the counting register, and their state is manipulated and transformed during the course of the algorithm to determine the phase of the unitary operator U. "After the initial preparation, a Hadamard gate is applied to all the auxiliary qubits, transforming them into a superposition state. This superposition allows for the encoding of multiple phases onto the auxiliary qubits simultaneously. The state at the end of the preparation stage is,

$$|u_1\rangle = \frac{1}{\sqrt{N}}(|0\rangle + |1\rangle)^{\otimes t} |u\rangle$$

Following the Hadamard transformation, the gates denoted by U^{2^p} are applied, where p ranges from 0 to $t-1$. These gates are controlled unitary (CU) gates, and they are applied at each auxiliary qubit. The application of these CU gates is crucial for phase kickback, as it encodes the unitary operator U's phase information onto the auxiliary qubits' state. The resulting transformation at any arbitrary stage p is given by,

$$U^{2^p} |u\rangle = e^{2\pi i 2^p \theta} |u\rangle$$

The controlled unitary gates are applied successively to repeatedly rotate the phase of the auxiliary qubits, gradually encoding the phase θ onto the state of the qubits in the Fourier basis. This encoding process allows us to represent the phase as a number between 0 and

2^t. The magnitude of the phase rotation performed by each controlled unitary gate is determined by the power 2^p, where p ranges from 0 to $t-1$. The more controlled unitary gates applied, the more precise the encoded phase information becomes. Hence, after applying the final unitary gate, $U^{2^{t-1}}$, we have essentially performed a quantum Fourier transform of $N\theta$. The resulting state is,

$$|u_1\rangle = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} e^{2\pi i m\theta} |m\rangle \otimes |u\rangle$$

By performing the inverse quantum Fourier transform on the state of the t auxiliary qubits, we can undo the initial quantum Fourier transform and retrieve the encoded phase information, represented as the state $|N\theta\rangle$.

$$|u_3\rangle = |N\theta\rangle |u\rangle$$

This step completes the quantum phase estimation algorithm and provides us with an estimate of the eigen phase of the unitary operator U.

Improving HHL's accuracy (ε)

There is a restriction on the maximum value the scaled eigenvalue can take. This is because the eigenphase information is encoded in the state of the t auxiliary qubits. The number of qubits used in the auxiliary register determines the number of bits required to represent the phase information. Given that each qubit can only store a value of either 0 or 1, the maximum number of possible combinations is limited to 2^t. This results in a restriction on the maximum value the encoded eigenvalue can take, which is limited to 2^t.

It is, therefore, crucial to making the right choice of the size of the auxiliary register since this is important in determining the precision of the quantum phase estimation algo-

rithm. A larger auxiliary register allows for greater precision in the estimated eigenphase, while a smaller auxiliary register results in less precision.

Let us consider an example where $\theta = 1/11$, which is the recurring decimal 0.0909.... This means that the phase change due to the unitary gates must be $\frac{2\pi}{11}$; since $e^{2i\pi\theta}$ is the scaling factor.

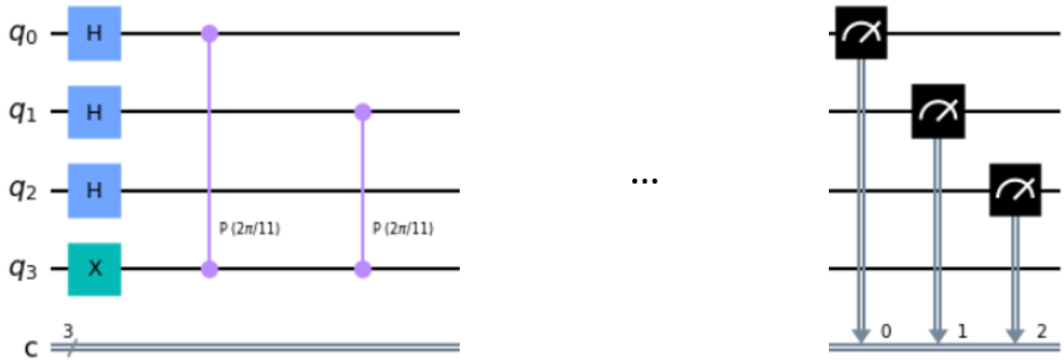


Figure 5.3: QPE Example using 3 auxiliary registers

The initial state of the quantum system is depicted in Figures 5.3 and 5.5. In Figure 5.3, there are 3 qubits, while in Figure 5.5, there are 5 qubits. The input eigenvector that we are using in both of these circuits is $|1\rangle$. To perform the phase estimation, a unitary gate that applies a phase shift of $2\pi/11$ is present in both of these circuits. These circuits have been designed based on the principles discussed throughout this section.

Results of the implementation of Figure 5.3 are displayed in Figure 5.4. It can be observed that the output of the phase estimation algorithm is not a clear representation of the phase applied by the unitary gate. The highest probability of occurrence is given to the state $|001\rangle$, which corresponds to a value of $1/8 = 0.125$. The next highest probability is for the state $|000\rangle$, which corresponds to the value 0. The desired outcome, represented by the phase 0.09090, is not accurately reflected in the output, although we see a high weightage given to state $|001\rangle$.

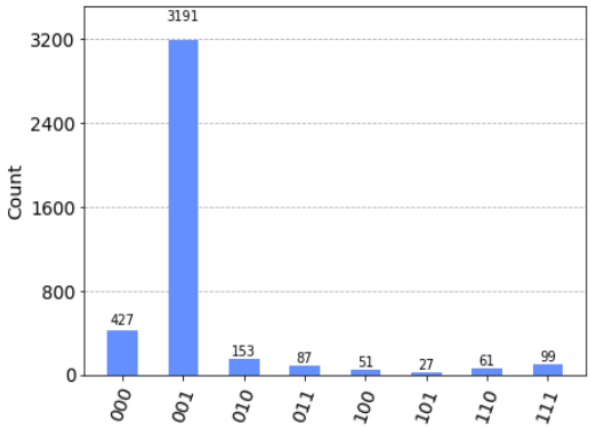


Figure 5.4: QPE Result with 3 auxiliary registers

By increasing the number of qubits in the auxiliary register from 3 to 5, a noticeable improvement in the results is observed. Figure 5.6 displays the impact of increasing the number of qubits. We can observe that a high probability is assigned to the state $|00011\rangle$, representing the decimal value 0.0935. This value is in close proximity to the desired result of 0.09090... and is slightly higher than the expected outcome. As a result, we can conclude that the next probable state (with a lower probability) is $|00010\rangle$, which corresponds to the value 0.0625. This is what is observed in 5.6.

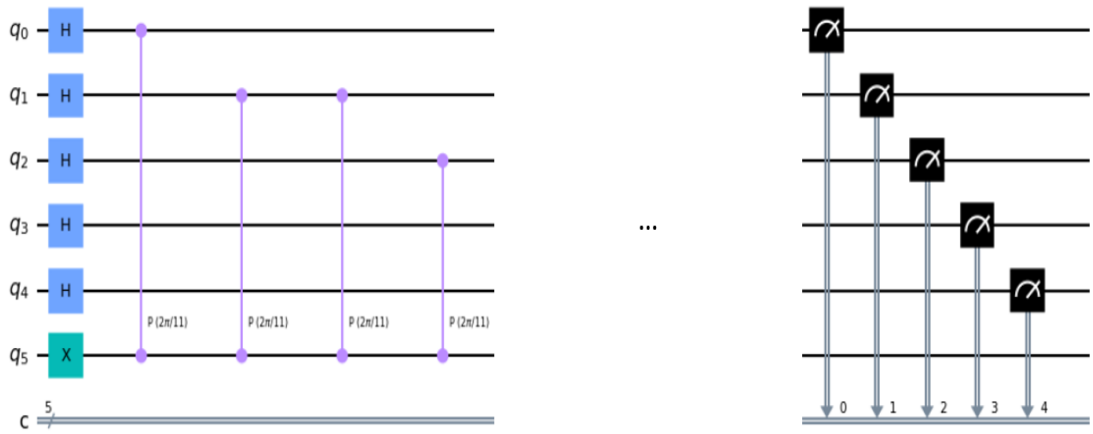


Figure 5.5: QPE Example using 5 auxiliary registers

5.4 HHL and the Central Path

Algorithm 3: Quantum Accelerated Central Path Algorithm

Input: $c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$

Output: x

Set: $(x, y, w, z) > 0$

while $\gamma \geq \varepsilon$ **do**

- Set: $\rho = b - Ax - w$
- Set: $\sigma = c - A^T y + z$
- Set: $\gamma = z^T x + y^T w$
- Set: $\mu = \delta \frac{\gamma}{n+m}$, $0 < \delta < 1$
- Compute step directions $(\Delta x, \Delta w, \Delta y, \Delta z)$ using (5.1)
- Compute step size using
- $x = x + \theta_1 \Delta x$
- $w = w + \theta_1 \Delta w$
- $y = y + \theta_2 \Delta y$
- $z = z + \theta_2 \Delta z$

Algorithm 3 shows the rewritten Central Path Method. The linear systems of equa-

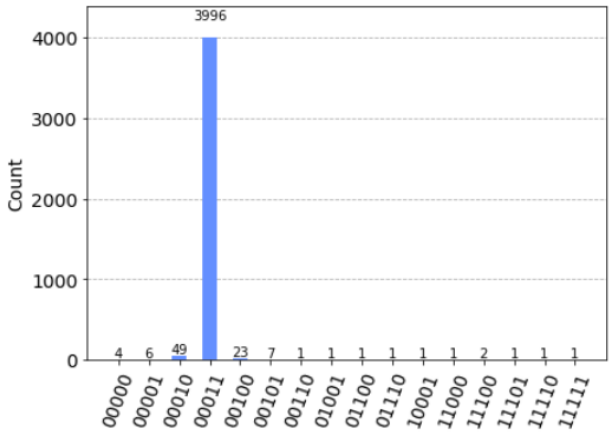


Figure 5.6: QPE Result with 3 auxillary registers

tions that arise during the method's iterations are solved using HHL. This allows for faster convergence of the Central Path Method.

5.4.1 Complexity Analysis

We present a theorem that provides an upper bound for the solution obtained through the utilization of the HHL algorithm when solving the linear set of equations that defines the central path.

Theorem 5.1. *Consider the linear optimization problem defined by . The total number of iterations to get an* ε*-solution, i.e., a solution that satisfies* $z^T x + y^T w \leq \varepsilon$*, is bounded by*

$$O(N\frac{s^2\kappa^2}{\varepsilon}(\log N)^2).$$

where N *is the size of the input vector,* s *is the sparsity of the matrix,* κ *is its condition number and* ε *is the desired accuracy.*

Proof. The proof for this theorem stems from the fact that the bound for the number of iterations required by central path method is given by [28],

$$O(N\log\frac{N}{\varepsilon})$$

The bound for the HHL method is given by [5],

$$O(\log(N)\frac{\kappa^2 s^2}{\varepsilon})$$

The new bound then becomes,

$$O(N\frac{s^2\kappa^2}{\varepsilon}(\log N)^2).$$

□

5.5 Discussion

Before moving on to the experiments chapter, we will discuss a few shortcomings and address a few modifications for a more practical implementation to overcome the limitations of the new central path approach.

- QPE has a trade-off relationship with the time taken at each step. In other words, increasing the precision of QPE often comes at the cost of longer computation time, and vice versa.

- One of the primary limitations of the HHL algorithm is its restrictions on the input. It is imperative to start with optimal instances to ensure the continuation of favorable outcomes, and this is a crucial consideration for the practical application of the algorithm.

- Generating a time evolved matrix from the Hermitian input is generally hard in a practical setting. At present, the only matrices that have been identified as satisfying the necessary criteria are local hamiltonians [26] and sparse hamiltonians [6].

- QPE has a range that ranges from 0 to 2^t. It is important to consider the range of values for the input to ensure that the QPE method can be utilized effectively. Instances that fall outside this range cannot be used for the QPE process.

For the sake of this thesis, we will consider the HHL algorithm to be a black box. Instead of attempting to design and implement the algorithm, the focus is on using an external library that has already been developed and tested. This will allow us to concentrate on the application of the algorithm rather than its internal workings and saves time and resources. Using an external library also ensures that the results obtained are reliable and accurate, as the library has likely been rigorously tested and optimized. This approach allows us to make the most of the HHL algorithm while focusing on the thesis's main objectives.

Chapter 6

Evaluation and Results

As established in previous chapters, the iterative process of finding the optimal solution to a linear programming problem can be highly computationally intensive, particularly when it comes to solving the linear set of equations. However, recent advancements in mathematical methods and the emergence of quantum computing have introduced new and innovative approaches to addressing this challenge. These approaches, each with unique strengths and limitations, offer the potential for improved efficiency and accuracy in solving linear equations.

This thesis chapter is dedicated to evaluating the Central Path Method in three distinct scenarios. The first scenario involves using Newton's approximation as a linear solution for the Interior Point Method . The second scenario, which we will refer to as the nonlinear solution for the IPM, utilizes McCormick envelopes. Finally, the third scenario employs the quantum HHL algorithm, resulting in a quantum solution for the IPM.

The primary objective of this thesis is to compare these three methods in terms of accuracy, efficiency, and scalability and to provide insight into their strengths and limitations. To achieve this goal, we will conduct a thorough theoretical analysis of each method, followed by a series of numerical experiments to evaluate their performance on various test cases. The results of this study will provide valuable insights into the capabilities and limitations of each method. They can inform future research on the development of new mathematical methods and quantum algorithms which involve solving linear systems of equations.

6.1 Experimental Setup

In order to effectively implement and run the Central Path Method, we utilize a range of specialized packages and software.

We utilize the Julia programming language as the foundation for implementing our work. The choice of Julia is driven by several factors, including its interoperability with other languages and systems, making it advantageous for integration with other tools such as Qiskit. Furthermore, the language features a growing ecosystem of packages devoted to linear optimization and mathematical programming, including the well-known JuMP library for modeling and solving optimization problems. The official documentation for JuMP [2] provides several reasons that highlight the importance of using this library as a modeling language for our problem.

- A user friendly syntax.
- Ease of changing solvers
- JuMP communicates with most solvers in memory, avoiding the need to write intermediary files.
- Benchmarking has shown that JuMP can create problems at similar speeds to special-purpose modeling.

The JuMP library is a valuable tool for solving mathematical optimization problems as it offers seamless integration with various solvers. In this instance, we will be utilizing the Gurobi solver to solve linear and nonlinear programming problems.

Gurobi was chosen as the preferred solver because it offers a wide range of parameters that allow for control over the solver's behavior. The "Method" and "NonConvex" parameters are particularly interesting, as they allow us to fine-tune the solver's behavior. For example, by setting the "Method" parameter to 2, we instruct the solver to use interior point methods when solving linear programming problems, making it easier to compare results

obtained from the in-built optimizer with the Central Path Method. Additionally, setting the ”NonConvex” parameter to 2 enables the Gurobi solver to tackle nonlinear problems efficiently.

In order to perform experimentation with the HHL algorithm, we have chosen to use the Qiskit library in Python rather than writing our implementation in Julia. Qiskit is a comprehensive library for quantum computing, containing implementations of various quantum algorithms and valuable tools for quantum computing. The Julia programming language is still utilized in this process by integrating Python via the ”PyCall” package. With this integration, we can import Qiskit and treat the HHL circuit as a blackbox for our experimentation.

The computational demands of many optimization tasks, such as those involved in the Central Path Method, are often substantial. This is particularly true for problems requiring lots of intermediate memory and complex constraints, which can require extensive computation times on even the most powerful personal computers. In such cases, using supercomputing resources can provide a significant advantage. Supercomputers are designed to handle large and complex calculations as they have specialized hardware and software that allow for highly parallel and efficient processing. For the purposes of our thesis, we use the Cedar supercomputer [3] provided by Compute Canada.

To optimize the use of resources and save time, jobs on Cedar are submitted to the SLURM workload manager as job arrays. Each job has a time limit of 7 days and has access to 500 GB of memory. Job execution occurs on one of the 96 available nodes in this memory category. This approach allows for efficient processing and management of tasks as they share a lot of similarities with each other.

Table 6.1: Compute Canada Usage Statistics

Resource	Total CPU Usage (in core years)	Projected CPU Usage (in core years)
cedar-compute	8.08	9.27

We present a summary of the CPU usage statistics for the experiments performed in the subsequent sections of this thesis. This information is presented in Table 6.1. The table demonstrates a total usage of 8.08 core years, equivalent to continuously running computations on a single CPU core for approximately eight years. This high computational demand clearly highlights the need for utilizing a supercomputer, as it would not be feasible to run these programs on a local personal computer.

6.1.1 Measures of Performance

We set the accuracy of the circuit ε to be 10^{-3}. This is done in the constructor of the HHL method in Qiskit. We also define a few measures that will be analyzed in results,

State Fidelity

State fidelity refers to the measure of the similarity between the actual state of a quantum system and the desired target state. It quantifies how close the actual state of the system is to the target state and provides a way to evaluate the performance of quantum operations and algorithms. The values state fidelity can take on lie between 0 and 1, with a value of 1 indicating an exact match between the actual and target states. A value less than 1 indicates some deviation from the desired state. We will use the inbuilt "state_fidelity()" method available in Qiskit to calculate this value.

RRS/SSE

To evaluate the performance of the HHL method, we use the sum of squared estimate of errors (SSE) measure. It measures the discrepancy between the data and an estimation model. The SSE is calculated by taking the difference between the actual and estimated data, squaring the difference, and then summing up all the squared differences. The SSE provides a quantitative measure of the accuracy of the HHL method in solving the linear system of equations. The lower the SSE, the better the accuracy of the HHL method, and

vice versa.

$$SSE = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

, where y_i is a given value and f(x_i) is the result of Quantum Central Path Mehtod.

To ensure clear and consistent terminology, we will establish some naming conventions for the various forms of the Central Path Method that we will use in this study. The Central Path Method that employs Newton's method will be referred to as the "Linear Central Path Method." The approach utilizing the non-convex solver provided by Gurobi will be referred to as the "Nonlinear Central Path Method." Finally, implementing the Central Path Method using the quantum HHL algorithm will be referred to as the "Quantum Central Path Method." By defining these terms, we aim to facilitate clear communication and distinguish between the different methods under consideration.

6.1.2 Instance Generation

For the evaluation of the Classical and Quantum algorithms, we utilize two distinct types of instances.

The first instance is in the form of a tridiagonal matrix. These instances which is characterized by having non-zero elements only on its main diagonal, lower diagonal, and upper diagonal [21]. This specific form is chosen because it is well-suited for efficient implementation using the latest qiskit library's HHL algorithm which has inbuilt optimizations in place for such inputs.

The second type of instances used in the evaluation are referred to as "easy" instances. These instances are chosen because they are the standard instances commonly used for evaluating feasible Interior Point Methods (IPMs), as illustrated in [25]. By choosing these instances for evaluation, readers will be able to make fair comparisons with other papers in the future. These easy instances are characterized by,

$$A(i,j) = \begin{cases} 0, & \text{if } i \neq j \text{ and } j \neq i+m \\ 1, & \text{otherwise} \end{cases} \tag{6.1}$$

for $i = 1...m$ and $j = 1...n$ and $n = 2m$.

Instances for the Linear and Nonlinear Central Path Method take the following form,

- Efficient Instances: The input matrix is a tridiagnoal matrix, the vector b and the slack variables are vectors of ones. We start with a matrix size of 2×2 and go up to a size of 25×25.
- Easy Instances: The input matrix is a matrix following 6.1, the vector b and the slack variables are vectors of ones. We start with a matrix size of 2×4 and go up to a size of 25×50.

Instances for the Quantum Central Path Method take the following form,

- Efficient Instances: The input matrix is a tridiagnoal matrix, the vector b and the slack variables are vectors of ones. We start with a matrix size of 2×2 and go up to a size of 5×5.
- Easy Instances: The input matrix is a matrix following 6.1, the vector b and the slack variables are vectors of ones. We start with a matrix size of 2×4 and go up to a size of 4×8.

6.2 Results

In this section, we will look at a comprehensive analysis of the results obtained from the extensive experimentation carried out on the three methods.

6.2.1 Linear Central Path Method

Table 6.2 displays the outcomes of the Linear Central Path Method for easy instances. Column 2 displays the number of iterations completed, column 5 shows the IPM optimal, and the final column, column 6, shows the amount of time taken by the Linear Central Path Method.

Easy instances have the characteristic that their optimal value is 0 as shown in column 3 of Table 6.2. This occurs when x is a vector of zeros. We set the "Method" parameter in Gurobi to 2. This sets the optimizer to use interior point methods to solve the LP. The time taken by the optimize!() call is given in column 4.

Analyzing the results in Table 6.2, it is evident that the Linear Central Path Method requires a minimal number of iterations, typically around 7, to arrive at a solution regardless of the size of the instance. This highlights the effectiveness of the Linear Central Path Method in quickly finding solutions to easy cases. Additionally, the optimal values produced by the IPM Optimal are observed to be very close to 0, demonstrating the accuracy of the Linear Central Path Method in solving these instances. Finally, we notice that the time taken by our implementation of the Linear Central Path Method is faster than the inbuilt Gurobi solver. This can be attributed to fact that the inbuilt Gurobi solver uses presolve reductions and spatial branching techniques.

Table 6.3 displays the results of the Linear Central Path Method for efficient instances. The columns in the table retain the same headings as those in Table 6.2.

Analysis of the results in Table 6.3 tells us that the number of iterations required by the Linear Central Path Method increases as the size of the instance increases. This indicates that the method may require more computational resources to solve larger and more complex instances. However, the optimal values produced given in IPM Optimal are found to match those of the optimal.

Despite the increase in the number of iterations required, our implementation of the Linear Central Path Method remains faster than the inbuilt Gurobi solver even for efficient

Table 6.2: Linear Solutions to Easy Instances

Instance	Iterations	Optimal Value	Optimal Time (s)	IPM Optimal	IPM Time (s)
2×4	6	0	2.777	-2.01E-06	1.572
3×6	6	0	3.052	-3.02E-06	1.791
4×8	7	0	3.112	-4.04E-07	1.767
5×10	7	0	2.972	-5.05E-07	1.75
6×12	7	0	3.178	-6.05E-07	1.797
7×14	7	0	3.142	-7.06E-07	1.762
8×16	7	0	2.972	-8.07E-07	1.753
9×18	7	0	3.05	-9.08E-07	1.755
10×20	7	0	3.034	-1.01E-06	1.746
11×22	7	0	3.078	-1.11E-06	1.769
12×24	7	0	3.745	-1.21E-06	2.129
13×26	7	0	3.032	-1.31E-06	1.75
14×28	7	0	3.035	-1.41E-06	1.771
15×30	7	0	3.12	-1.51E-06	1.769
16×32	7	0	3.043	-1.61E-06	1.831
17×34	7	0	3.004	-1.72E-06	1.757
18×36	7	0	3.057	-1.82E-06	1.828
19×38	7	0	3.012	-1.92E-06	1.765
20×40	7	0	3.059	-2.02E-06	1.814
21×42	7	0	3.008	-2.12E-06	1.78
22×44	7	0	3.064	-2.22E-06	1.801
23×46	7	0	3.095	-2.32E-06	1.857
24×48	7	0	2.991	-2.42E-06	1.764
25×50	7	0	3.029	-2.52E-06	1.811

instances. Although the improvement in speed is not as significant as when solving easy instances, our implementation is still able to outperform the inbuilt Gurobi solver in terms of computational time.

We conclude the analysis of the Linear Central Path Method by observing the change of γ or the duality gap with respect to the instance size. Our findings, as presented in Figure 6.1, show that for easy instances, the gap generally increases linearly as the input size increases. This linear increase in the gap might be due to the relatively fast convergence observed in Table 6.2 as the size of the instance increases. However, when it comes to

Table 6.3: Linear Solutions to Efficient Instances

Instance	Iterations	Optimal Value	Optimal Time (s)	IPM Optimal	IPM Time (s)
2×2	7	1	3.13	1	2.451
3×3	7	1	3.133	1	2.459
4×4	7	2	3.013	2	2.443
5×5	7	2	3.01	2	2.419
6×6	8	2	3.075	2	2.473
7×7	9	3	3.019	3	2.457
8×8	8	3	3.039	3	2.424
9×9	9	3	2.995	3	2.46
10×10	9	4	3.044	4	2.441
11×11	9	4	3.229	4	2.487
12×12	9	4	3.056	4	2.446
13×13	10	5	3.084	5	2.394
14×14	9	5	2.957	5	2.403
15×15	10	5	3.087	5	2.427
16×16	10	6	3.038	6	2.449
17×17	9	6	3.091	6	2.407
18×18	10	6	2.95	6	2.303
19×19	10	7	2.886	7	2.27
20×20	10	7	3.396	7	2.47
21×21	10	7	3.128	7	2.445
22×22	10	8	3.151	8	2.454
23×23	10	8	3.259	8	2.453
24×24	10	8	3.114	8	2.413
25×25	11	9	3.125	9	2.459

efficient instances 6.2, we do not observe a clear pattern in the change of the gap. This suggests that the convergence of the Linear Central Path Method may be more complex for efficient instances, and may require further investigation.

6.2.2 Nonlinear Central Path Method

To solve our optimization problem using the Nonlinear Central Path Method, we utilized the inbuilt non-convex solver in Gurobi by setting the ”Method” parameter to 2. The Gurobi solver tackles the non-convex problem by transforming the quadratic constraint into

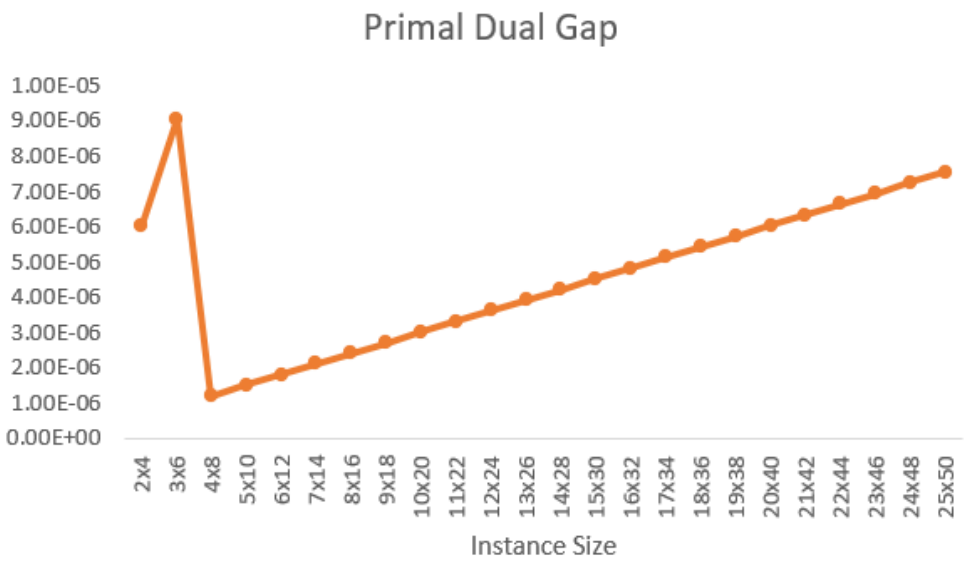


Figure 6.1: [Linear] Duality Gap vs Instance Size (Easy Instances)

piecewise linear approximations. Gurobi then employs spatial branching, a technique that partitions the solution space into smaller, manageable sub-spaces.

The optimization process heavily relies on bounds. Therefore, we started with large bounds and iteratively refined them until we found the optimal range. This involved determining a balance between the size of the bounds and the amount of time it takes to solve a single step in the Nonlinear Central Path Method. After experimentation, we determined that a range of $[-1.5, 1.5]$ resulted in a manageable solving time while still producing accurate results.

Additionally, in some instances, the algorithm may converge extremely slowly, resulting in a high number of iterations required to reach the desired threshold of $\gamma = 10^{-6}$. In such cases, the threshold is capped at $\gamma = 10^{-3}$ as an acceptable solution.

Table 6.4 displays the results of the Nonlinear Central Path Method for easy instances. The optimal values produced by the method are consistent with the optimal solutions, although they perform worse compared to their linear counterpart. Despite this, the Nonlinear Central Path Method proves to be much faster than the Linear Central Path Method.

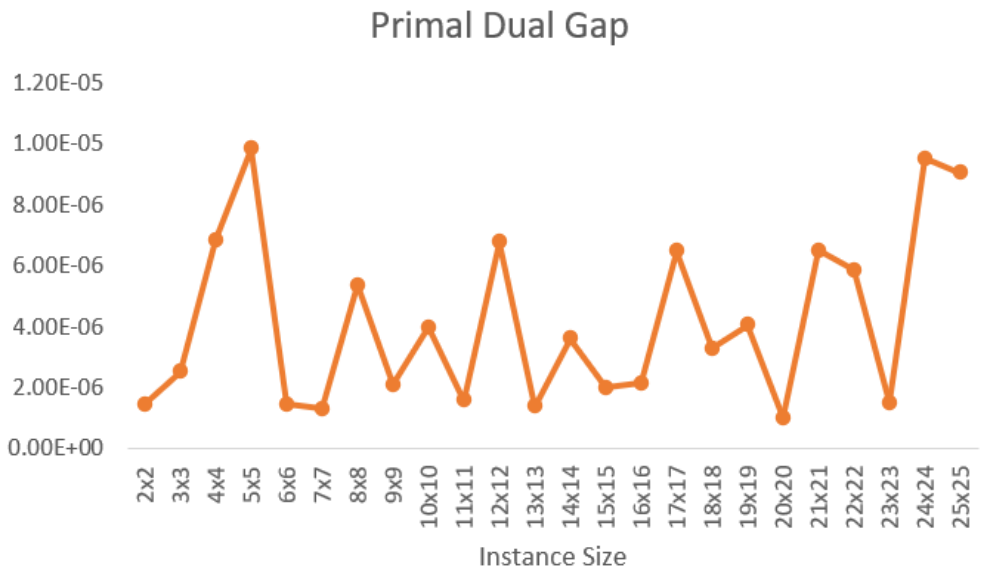


Figure 6.2: [Linear] Duality Gap vs Instance Size (Efficient Instances)

The efficient instance results are given in 6.5. The computation time and the method's optimal solution are less favorable compared to when the same efficient instances were solved using Newton's method. For example, with a 12×12 matrix size, the Nonlinear Central Path Method took almost 10 hours to find a solution of 6, while the optimal solution is 4. The last row in the table reveals that when solving a 14x14 matrix, a large number of solutions were searched and over 500gb of memory was consumed. The reason for a large amount of memory consumed can be attributed to the fact that Gurobi uses spatial branching. Spatial branching is a process used in some optimization algorithms to explore different branches of the search space simultaneously. It can take a long time because, in some cases, there may be a large number of possible branching options that need to be explored, and each option requires the algorithm to solve a new subproblem. This problem becomes more prominent as the size and complexity of the optimization problem increases.

Our findings for the duality gap vs instance size are illustrated in Figure 6.3. It demonstrates that for easy instances, the gap generally decreases linearly as the input size increases, with high values for γ as expected. However, when considering efficient instances,

Table 6.4: Nonlinear Solutions to Easy Instances

Instance	Iterations	Optimal Value	Optimal Time (s)	IPM Optimal	IPM Time (s)
2×4	8	0	2.777	-3.99E-02	0.744
3×6	8	0	3.052	-5.99E-02	0.753
4×8	8	0	3.112	-7.98E-02	0.839
5×10	8	0	2.972	-9.98E-02	0.858
6×12	8	0	3.178	-1.20E-01	0.924
7×14	8	0	3.142	-1.40E-01	0.934
8×16	8	0	2.972	-1.60E-01	1.106
9×18	8	0	3.05	-1.80E-01	1.158
10×20	8	0	3.034	-2.00E-01	1.212
11×22	8	0	3.078	-2.19E-01	1.352
12×24	8	0	3.745	-2.39E-01	1.184
13×26	8	0	3.032	-2.59E-01	1.28
14×28	8	0	3.035	-2.79E-01	1.263
15×30	8	0	3.12	-2.99E-01	1.354
16×32	8	0	3.043	-3.19E-01	1.469
17×34	8	0	3.004	-3.39E-01	1.462
18×36	8	0	3.057	-3.59E-01	1.259
19×38	8	0	3.012	-3.79E-01	1.261
20×40	8	0	3.059	-3.99E-01	1.321
21×42	8	0	3.008	-4.19E-01	1.317
22×44	8	0	3.064	-4.39E-01	1.412
23×46	8	0	3.095	-4.59E-01	1.402
24×48	8	0	2.991	-4.79E-01	1.479
25×50	8	0	3.029	-4.99E-01	1.516

as depicted in Figure 6.4, a significant number of γ values are of the order of 10^{-6}.

6.2.3 Quantum Central Path Method

We now present the results for the Quantum Central Path Method. It should be noted that there is a slight difference in the header conventions used in Tables 6.2 and 6.3. We utilize the AER library of Qiskit for quantum simulations. Specifically, we employ the statevector simulator, which offers a precise solution to the quantum circuit. This feature is particularly useful when the quantum circuit's size and complexity are limited, and the

Table 6.5: Nonlinear Solutions to Efficient Instances

Instance	Iterations	Optimal Value	Optimal Time (s)	IPM Optimal	IPM Time (s)
2×2	7	1	3.13	1	0.818
3×3	9	1	3.133	2	0.886
4×4	9	2	3.013	2	8.602
5×5	9	2	3.01	3	1.378
6×6	15	2	3.075	3	34.641
7×7	10	3	3.019	4	8.669
8×8	21	3	3.039	4	653.515
9×9	10	3	2.995	5	83.122
10×10	29	4	3.044	5	5762.589
11×11	10	4	3.229	6	477.242
12×12	29	4	3.056	6	33905.434
13×13	10	5	3.084	7	8655.362
14×14	0	5	2.957	MEMLIMIT	

number of qubits is low. However, as the circuit size and qubit count increase, the computational cost of the simulation becomes prohibitively high. Tables 6.6 - 6.9 convert input instances to suitable inputs for the inbuilt HHL algorithm. It does this by using the ”matrix_resize” method provided in Qiskit. This method first checks whether the input matrix can be represented exactly by quantum states by making sure the input vector is a power of 2. It then converts the given square matrix to a Hermitian matrix.

We see two new columns in Table 6.6. Column 7 in the table focuses on measuring the sum of squared estimate of errors. This measurement serves as an indicator of the accuracy of the method in solving the central path problem. Moreover, the worst case state fidelity reached during the iterations using the HHL algorithm is recorded in column 6. This information provides insight into the robustness of the method, as well as its ability to maintain a high degree of precision in the face of potential fluctuations or disturbances during the iterations.

The results for easy instances of the Quantum Central Path Method are presented in Table 6.6. We observe that there is a significant duality gap present, with the largest value

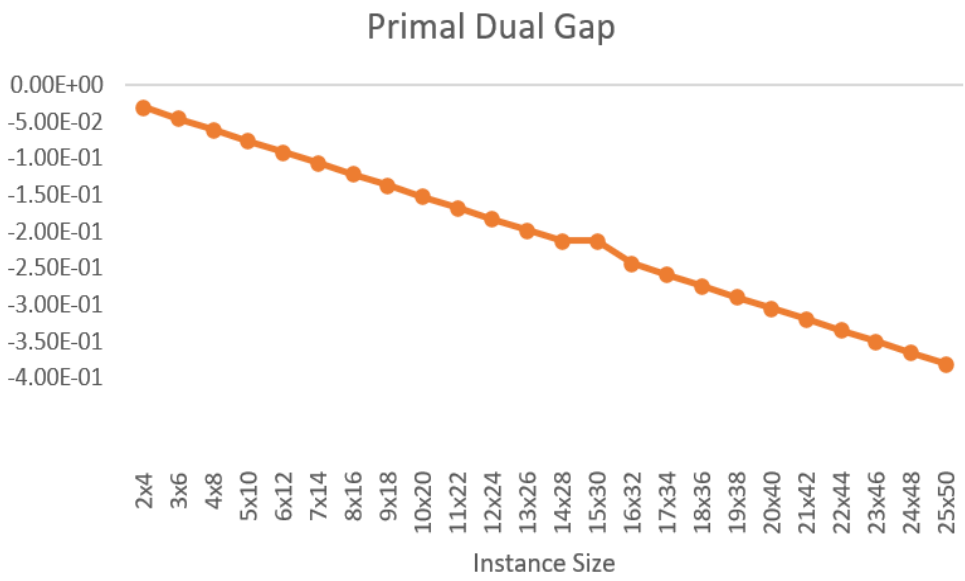


Figure 6.3: [Nonlinear] Duality Gap vs Instance Size (Easy Instances)

Table 6.6: Quantum Solutions to Easy Instances

Instance	Duality Gap	Optimal Value	IPM Optimal	IPM Time (s)	Minimum Fidelity	SSE
2×4	-192.0106321	0	-0.245	12908.325	0.999697	178.161
3×6	-278.3466245	0	-0.364	122732.773	0.999967	259.468
4×8	-371.128	0	-0.486	120468.344	0.999967	345.958

being -371.128 in row 3. The IPM Optimal values are found to be very close to the actual optimals, however, the Sum of Squared Errors (SSE) column indicates that these values are not good approximates. The large SSE values suggest that the optimals are not reliable estimates.

On the other hand, the results for efficient instances, presented in Table 6.8, reveal that the SSE values are relatively low and are accompanied by small gap values. It should be noted that for the 5x5 instance, the maximum runtime limit of 7 days was reached and the last iteration values were recorded. Despite this limitation, the results for the efficient instances demonstrate that the Quantum Central Path Method is capable of producing accurate and reliable results.

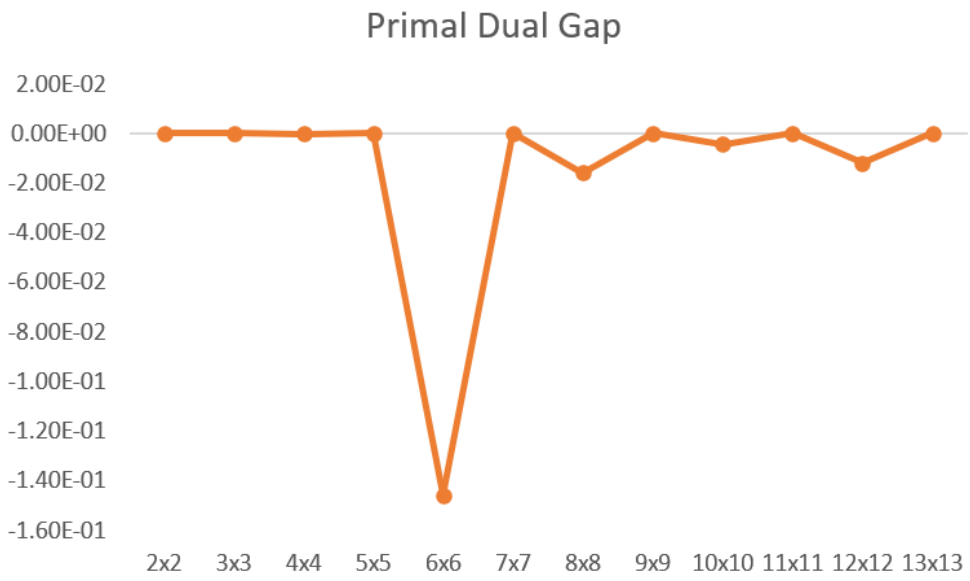


Figure 6.4: [Nonlinear] Duality Gap vs Instance Size (Efficient Instances)

Table 6.7: Quantum Metadata for Easy Instances

Instance	HHL Input Size	Iterations	CPU Efficiency	Memory Used (GB)
2×4	32×32	4	1 D 12 H	5.97
3×6	64×64	4	9 D 8H	42.76
4×8	64×64	4	9 D 10 H	46.32

Table 6.8: Quantum Solutions to Efficient Instances

Instance	Duality Gap	Optimal Value	IPM Optimal	IPM Time (s)	Minimum Fidelity	SSE
2×2	-0.272872201	1	1.112556415	6212.561031	0.999697	0.531
3×3	-0.459883783	1	2.083048712	19043.03069	0.999017	1.081
4×4	-0.257149333	2	2.027472051	107388.935	0.99688	0.63
5×5	2.854052961	2	-2.758604754	TIMELIMIT	0.998971	0.384

Table 6.9: Quantum Metadata for Efficient Instances

Instance	HHL Input Size	Iterations	CPU Efficiency	Memory Used (GB)
2×2	16×16	4	22 H	3.96
3×3	32×32	6	2 D 3 H	8.46
4×4	32×32	18	8 D 8 H	23.11
5×5	64×64	9	36 D 21 H	53.86

Tables 6.7 and 6.9 provide detailed information on the computing resources consumed by the Quantum Central Path Method. As the size of the instances increases, the input matrix size required by the HHL algorithm also grows. This is because the matrix needs to be transformed into a hermitian matrix. Our experiments have shown that the largest matrix size used was 64×64, which required a memory capacity of approximately 50GB. The CPU Efficiency column in the tables indicates the equivalent time it would take to run the job with the given resources. For example, for the 5x5 instance, the job was run for a time equivalent to 37 days.

6.3 Discussion

In this chapter, we presented the results of our experiments on the Nonlinear Central Path Method, Linear Central Path Method, and Quantum Central Path Method. Through our analysis, we could compare the performance of these methods for different instance sizes. We found that the Nonlinear Central Path Method was faster than the Linear Central Path Method. At the same time, the Quantum Central Path Method was able to keep up in terms of the quality of the solution.

During the experimentation process, several interesting observations were made. Firstly, the solutions after each iteration remained primal feasible and dual infeasible when starting with a primal feasible point. On the other hand, if the starting point was dual feasible, the solutions remained dual feasible and primal infeasible throughout the iterations. This consistency in the feasibility of the solutions provides valuable insight into the behavior of these methods.

Additionally, it was observed that capping the value of γ to a lower value had a limited impact on the solution. This suggests that these methods have a high degree of utility and may be effectively applied in a broader range of applications. This could be a possibility for open questions that are needed to fully understand the implications and their impact on the optimization process.

Chapter 7

Conclusion

This thesis proposed a novel approach to solving linear programming problems by integrating the HHL algorithm with the central path method. The study explored three variants of the Central Path Method: the Linear Central Path Method, the Nonlinear Central Path Method, and the Quantum Central Path Method. Extensive experimentation was conducted on both easy and efficient instances. Although the HHL algorithm demonstrated high fidelity, it incurred a significant time penalty and could not complete certain instances within the allotted time. Nevertheless, integrating quantum algorithms with classical optimization methods holds great promise for solving large-scale linear programs. This research highlights the potential of quantum algorithms in solving complex optimization problems and suggests further optimization of the integration process as well as exploration of the practical applications of this approach in various fields.

Bibliography

[1] qiskit.quantum_info.state_fidelity. https://qiskit.org/documentation/stubs/qiskit.quantum_info.state_fidelity.html. Accessed: 2023-02-05.

[2] Should I use JuMP? · JuMP. https://jump.dev/JuMP.jl/stable/should_i_use/. Accessed: 2023-02-05.

[3] Supercomputer cedar. https://www.sfu.ca/research/supercomputer-cedar#:~:text=Simon%20Fraser%20University%27s%20Supercomputer%20Cedar,way%20for%20new%20research%20breakthroughs. Accessed: 2023-02-05.

[4] Pioneering quantum information science. *Nature Computational Science*, 2(11):687–688, Nov 2022.

[5] Amira Abbas, Stina Andersson, Abraham Asfaw, Antonio Corcoles, Luciano Bello, Yael Ben-Haim, Mehdi Bozzo-Rey, Sergey Bravyi, Nicholas Bronn, Lauren Capelluto, Almudena Carrera Vazquez, Jack Ceroni, Richard Chen, Albert Frisch, Jay Gambetta, Shelly Garion, Leron Gil, Salvador De La Puente Gonzalez, Francis Harkins, Takashi Imamichi, Pavan Jayasinha, Hwajung Kang, Amir h. Karamlou, Robert Loredo, David McKay, Alberto Maldonado, Antonio Macaluso, Antonio Mezzacapo, Zlatko Minev, Ramis Movassagh, Giacomo Nannicini, Paul Nation, Anna Phan, Marco Pistoia, Arthur Rattew, Joachim Schaefer, Javad Shabani, John Smolin, John Stenger, Kristan Temme, Madeleine Tod, Ellinor Wanzambi, Stephen Wood, and James Wootton. Learn Quantum Computation Using Qiskit, 2020.

[6] Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge, 2003.

[7] Stefanie Barz, Ivan Kassal, Martin Ringbauer, Yannick Ole Lipp, Borivoje Dakić, Alán Aspuru-Guzik, and Philip Walther. A two-qubit photonic quantum processor and its application to solving systems of linear equations. *Scientific Reports*, 4(1), aug 2014.

[8] Robert Benkoczi, Daya Gaur, Naya Nagy, Marius Nagy, and Shahadat Hossain. Quantum bitcoin mining. *Entropy*, 24(3), 2022.

[9] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In *Automata, Languages and Programming: 25th International Colloquium, ICALP'98 Aalborg, Denmark, July 13–17, 1998 Proceedings 25*, pages 820–831. Springer, 1998.

[10] X.-D. Cai, C. Weedbrook, Z.-E. Su, M.-C. Chen, Mile Gu, M.-J. Zhu, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Experimental quantum computing to solve systems of linear equations. *Physical Review Letters*, 110(23), jun 2013.

[11] Pablo Casares and Miguel Martin-Delgado. A quantum interior-point predictor-corrector algorithm for linear programming. *Journal of Physics A: Mathematical and Theoretical*, 53, 09 2020.

[12] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The power of block-encoded matrix powers: Improved regression techniques via faster hamiltonian simulation. 2019.

[13] Mircea Cirnu and Irina Badralexi. ON NEWTON-RAPHSON METHOD. *Romanian Economic Business Review*, 5:91–94, 01 1995.

[14] G. Dantzig. Maximization of linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation. Proceedings of the Conference on Linear Programming*, 01 1951.

[15] Antoine Deza, Eissa Nematollahi, M. Reza Peyghami, and Tamás Terlaky. The central path visits all the vertices of the Klee–Minty cube. *Optimization Methods & Software - OPTIM METHOD SOFTW*, 21, 09 2004.

[16] A. M. Geoffrion. Duality in Nonlinear Programming: A Simplified Applications-Oriented Development. *SIAM Review*, 13(1):1–37, 1971.

[17] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, 103(15), oct 2009.

[18] Younbae Jun and Junho Jeon. Modified bisection method for solving nonlinear equations. *International Journal of Scientific and Innovative Mathematical Research (IJSIMR)*, 7(9):8–11, 2019.

[19] Narendra Karmarkar. A New Polynomial-Time Algorithm for Linear Programming-II. *Combinatorica*, 4:373–395, 12 1984.

[20] G.P. McCormick, GEORGE WASHINGTON UNIV WASHINGTON D C INST FOR MANAGEMENT SCIENCE, ENGINEERING., and United States. Air Force. Office of Scientific Research. *Computability of Global Solutions to Factorable Nonconvex Programs: Part I - Convex Underestimating Problems*. Serial. George Washington University, School of Engineering and Applied Science, Institute for Management Science and Engineering, 1975.

[21] J.M. McNamee. Chapter 5 - Newton's and Related Methods. In J.M. McNamee, editor, *Numerical Methods for Roots of Polynomials, Part I*, volume 14 of *Studies in Computational Mathematics*, pages 131–206. Elsevier, 2007.

[22] Alexander Mitsos, Benoit Chachuat, and Paul Barton. Mccormick-based relaxations of algorithms. *SIAM*, 20, 01 2009.

[23] Mohammadhossein Mohammadisiahroudi, Ramin Fakhimi, and Tamás Terlaky. Efficient Use of Quantum Linear System Algorithms in Interior Point Methods for Linear Optimization, 05 2022.

[24] Jr Morrell and Hiu Wong. Step-by-step hhl algorithm walkthrough to enhance the understanding of critical quantum computing concepts, 08 2021.

[25] Mousaab, Bouafia and Adnan, Yassine. Complexity analysis of primal-dual interior-point methods for linear optimization based on a new efficient Bi-parameterized kernel function with a trigonometric barrier term. *RAIRO-Oper. Res.*, 56(2):731–750, 2022.

[26] Daniel Nagaj. Local Hamiltonians in Quantum Computation, 2008.

[27] H Nakayama, H Sayama, and Y Sawaragi. A generalized Lagrangian function and multiplier method. *Journal of Optimization Theory and Applications*, 17:211–227, 1975.

[28] Jiming Peng, Cornelis Roos, and Tamás Terlaky. Self-regular functions and new search directions for linear and semidefinite optimization. *Math. Program.*, 93:129–171, 06 2002.

[29] Cornelis Roos, Tamás Terlaky, and Jean-Philippe Vial. *Interior Point Methods for Linear Optimization*. 01 2005.

[30] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, oct 1997.

[31] Yiğ it Subaşı, Rolando D. Somma, and Davide Orsucci. Quantum Algorithms for Systems of Linear Equations Inspired by Adiabatic Quantum Computing. *Physical Review Letters*, 122(6), feb 2019.

[32] Robert J Vanderbi. *The Central Path*, volume 196, page 257–269. Springer, four edition, 2014.

[33] Stephen Wright. A path-following infeasible-interior-point algorithm for linear complementarity problems. *Optimization Methods and Software*, 2(2):79–106, 1993.

[34] Stephen J. Wright. *Primal-Dual Interior-Points Methods*. SIAM, Philadelphia, Pa, USA, 1997.

[35] Zeguan Wu, Mohammadhossein Mohammadisiahroudi, Brandon Augustino, Xiu Yang, and Tamás Terlaky. An Inexact Feasible Quantum Interior Point Method for Linearly Constrained Quadratic Optimization. *Entropy*, 25:330, 02 2023.