UPPSALA
UNIVERSITET

Learning Preconditioners for Interior Point Methods

Aleix Nieto Juscafresa
Daiyuan Xu
Jonathan Franklin

Project in Data Science: Report

January 2024

PROJECT REPORT

Institutionen för informationsteknologi

Abstract

This paper highlights the potential of preconditioners in addressing computational challenges in large-scale linear optimization. We explore preconditioners in the context of interior-point methods (IPMs) for linear optimization problems. To tackle the computational bottleneck of IPMs, namely solving linear equation systems arising from the KKT conditions, we implement different preconditioners. By applying these techniques to network-flow problems transformed into linear programming, our results demonstrate that ILU and ILUT preconditioners improve the efficiency and convergence of IPMs via the primal-dual algorithm. In addition, the numerical problems encountered throughout the experiments are discussed along with the implementation of a novel data-driven approach to create preconditioners trained with data. The project concludes with a discussion of its possible future works.

Keywords: network flow, interior point methods, primal-dual algorithm, preconditioners

2 Introduction

Linear optimization problems are ubiquitous in research and engineering. While the simplex method has traditionally been a standard approach for solving these problems, interior point methods (IPMs) stand out as a robust and widely adopted alternative, particularly for large-scale optimization problems. These methods initiate from a feasible point and determine a search direction by solving a linear equation system derived from the Karush-Kuhn-Tucker (KKT) optimality conditions. The computational bottleneck in IPMs lies in solving this linear equation system. While iterative methods are common for large-scale problems, their efficiency often hinges on the availability of a good preconditioner. Unfortunately, devising effective preconditioners for the diverse class of linear equation systems in linear programming has proven challenging [1].

Recent advancements in data-driven optimization propose replacing handcrafted preconditioners with parameterized functions trainable on data [2, 3]. However, applying these techniques to interior-point methods is an unexplored area. The original idea of the project was to bridge this graph by employing graph neural networks to train data-driven preconditioners by choosing a suitable loss function. However, this idea did not prosper as we needed symmetric and positive definite matrices resulting from the IPM problems in order to train the preconditioners with data [4]. In the IPM context, the matrices do not satisfy these assumptions, and due to the lack of time, we have not been able to adapt this problem to the case of IPMs. Instead, we have conducted experiments using well-known preconditioners with which we have indeed obtained results.

The network flow problem optimizes goods or information flow in a network. It is represented by a directed graph with edge capacities, ensuring efficient flow without exceeding capacity. Applicable to various systems like traffic or fluid dynamics, the network-low problem is a natural choice for our study as it can be transformed into a normal form linear programming (LP) problem, which allows us to apply interior-point methods in conjunction with the primal-dual algorithm. At each iteration of the primal-dual algorithm, a linear system that originates from the KKT conditions is solved. In this project, several established preconditioners are applied to these linear systems, aiming to accelerate the convergence of the IPM.

We conduct a series of experiments by generating a diverse set of IPM problems. These experiments encompass varying sizes of network flow problems, allowing us to comprehensively assess the performance of our proposed methodologies. As part of our investigation, we apply different preconditioners within the IPM framework. Interestingly, while some preconditioners exhibit inconsistent results, others, such as incomplete LU (ILU) and incomplete LU with threshold (ILUT), consistently outperform the direct approach. However, our investigation did not come without challenges, as we meticulously addressed numerical issues inherent in implementing these preconditioners. This comprehensive analysis sheds light on the nuanced effectiveness of different preconditioners and emphasizes the crucial consideration of numerical intricacies in their practical application within IPMs.

In conclusion, our work not only extends the applicability of preconditioners to IPM problems, but also provides a detailed derivation of the entire process behind optimizing a linear problem through an IPM with a natural example such as the case of a network-flow problem.

3 Background

The proposed preconditioner is utilized in the primal-dual algorithm which is an optimization algorithm used for solving linear programming (LP) and quadratic programming (QP) problems. It is an iterative method that efficiently solves large-scale optimization problems by moving through the interior of the feasible region [5].

In this section, we elaborate on the process of transforming a network-flow problem into a linear problem, outlining the steps involved in this process. Following this, we discuss the derivation of a linear equation system from the Karush-Kuhn-Tucker (KKT) conditions and explain how this system yields the search direction crucial for our interior point method (IPM) in the primal-dual algorithm. Additionally, we will delve into how we choose a feasible point for the IPM.

3.1 Constructing the corresponding LP from the network-flow problem

The very first step of the IPM is to obtain a linear programming problem. The problem we face is a network-flow problem, where the goal is to determine the maximum flow within a given network graph. As shown in Appendix A, the flow problem can be transformed into an LP problem. The network-flow problem takes the following form:

$$\begin{aligned} \text{maximize} \quad & \sum_{(s,v)\in E} f(s,v), \\ \text{subject to} \quad & \\ & \sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w), \quad \forall v \in V, v \neq s,t. \\ & f(u,v) \leq c(u,v), \qquad\qquad\qquad \forall (u,v) \in E. \\ & f(u,v) \geq 0, \qquad\qquad\qquad\quad \forall (u,v) \in E. \end{aligned} \tag{1}$$

Nodes s and t are the source node and the sink node respectively. The function $f(a,b)$ is the flow from node a to node b, and $c(a,b)$ is the capacity of the edge (a,b). The vertex set is denoted as V and the edge set is denoted as E. The above constraint means the LP problem we want to solve is the maximum flow from the source to the sink.

The first equation means the input flow and the output flow from the identical vertex (node) should be equivalent. The second inequality illustrates that the flow of one edge should not be greater than the capacity of the corresponding edge. The third means the flow should be non-negative.

3.2 Primal-dual interior-point method for linear programming

The linear programming problem can be stated in its standard form as:

$$\min_x \; c^T x \;\; \text{s.t.} \;\; Ax = b,\, x \geq 0. \tag{2}$$

The equality constraints are linear and the restriction $x \geq 0$ applies componentwise, that is, all components of the vector $x \in \mathbb{R}^n$ are required to be non-negative.

By deriving the dual problem for (2) and using the KKT conditions we can obtain the linear system that we will solve in the primal-dual algorithm (see Appendix B for a detailed explanation of how the whole process is derived):

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \nabla x \\ \nabla \lambda \\ \nabla s \end{bmatrix} = - \begin{bmatrix} Ax - b \\ A^T\lambda + s - c \\ -XSe + \mu e \end{bmatrix}. \tag{3}$$

The solution $\nabla y = (\nabla x, \nabla \lambda, \nabla s)$ to (3) gives us the update direction in the primal-dual algorithm.

Note the stopping criterion checks both the central residual via η, and (approximate) primal and dual feasibility. This idea is further developed in Appendix B.1.

[0]See more about feasible points in Appendix B.2.

Algorithm 1 Primal-dual method

Input: : System of linear equations $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$
Output: Solution to the linear equation system $x_\star$
Initialize starting guess $(x^{(0)}, \lambda^{(0)}, s^{(0)})$, with $(x^{(0)}, s^{(0)}) > 0$ a strictly feasible point[1]
$\eta^{(0)} \leftarrow (x^{(0)})^T s^{(0)}$; $\sigma \in (0, 1)$
for $k = 0, 1, \ldots,$ until $\eta^{(k+1)} \leq \delta$ and $\delta \leq \left(\|r_{\text{primal}}\|_2^2 + \|r_{\text{dual}}\|_2^2\right)^{\frac{1}{2}}$ **do**
    $\mu \leftarrow \sigma \eta^{(k)} / n$
    Solve (3) with $(x, \lambda, s) = (x^{(k)}, \lambda^{(k)}, s^{(k)})$ ▷ Primal-dual update direction $\nabla y = (\nabla x^{(k)}, \nabla \lambda^{(k)}, \nabla s^{(k)})$
    Update $y^{(k+1)} = y^{(k)} + \alpha_k \nabla y$. ▷ $\alpha_k \in (0, 1]$ choice is discussed in Appendix B.3
    $\eta^{(k+1)} \leftarrow (x^{(k+1)})^T s^{(k+1)}; k \leftarrow k + 1$
end for

3.3 Preconditoners

The convergence speed of an iterative method for solving equation systems of the form $Ax = b$ depends on the condition number $k(A)$ of the input matrix A. Therefore, it is common practice to improve the spectral properties of the equation system before solving it. This is typically achieved by preconditioning, where the original system of equations is multiplied with a preconditioning matrix to improve its spectral properties. The underlying idea of preconditioning is to compute a cheap approximation of the inverse of A which is utilized to improve the convergence properties. There are two common ways to achieve this. First, it is possible to directly approximate the inverse of the matrix $M \approx A^{-1}$ called sparse approximate inverse methods [6] and the original equation system is replaced by $M^{-1}Ax = M^{-1}b$, which aims to improve the convergence properties. Another possibility is to approximate the original matrix A instead $M \approx A$ while restricting the obtained preconditioner M to be easily invertible [4, 7, 8]. See Appendix C for a more detailed introduction to preconditioners.

4 Methods

After obtaining Equation (3) different methods are applied to solve the linear system to obtain the direction updates ∇x, $\nabla \lambda$ and ∇s. This can be a bottleneck unless the method is chosen carefully. Some solvers can use a preconditioner that transforms the linear system into a form that is better for a computationally efficient iterative solution by improving the conditioning of the system.

The left side matrix from Equation (3) had some difficult matrix properties. It was non-symmetric, non-positive definite, and non-diagonally dominant. For symmetric positive-definite (spd) matrices, methods such as conjugate gradient can be used to obtain solutions to the linear system. However, with the matrix properties of our synthetic generated IPM problems described in Section 5.1, three solvers stand out. These three solvers are LU decomposition, GMRES and BICGSTAB. See Appendix D for a description of these solvers and how they have been implemented.

The results section will present a comparative analysis of GMRES (iterative solver) and LU decomposition (direct solver), detailing the trade-offs between computational efficiency and accuracy. The performance will be measured in terms of both the number of iterations required to reach a specified tolerance and the total computation time. This analysis will be crucial for understanding the practicality of each solver in real-world scenarios where the matrix properties may vary significantly.

Furthermore, the influence of preconditioners on the performance of the GMRES solver will also be evaluated, examining whether their use can enhance the conditioning of the matrix and thereby improve computational outcomes.

The BICGSTAB solver was successfully implemented and tested. Its results are not presented within this report due to constraints on the scope and focus of the current analysis, which was directed towards methods most suitable for the matrix characteristics under study.

5 Experiments

All experiments reported are run on an Intel Core i7-12700H CPU @ 2.3GHz in order to ensure a fair comparison in computation time for the different preconditioners. In practice, it is possible to further accelerate our experiments if specialized hardware such as GPUs are used.

The overall goal is to reduce the total computational time required to solve the problem up to a given precision measured by the combined duality gap and primal and dual feasibility residuals as described in Algorithm 1. The time used to compute the preconditioner beforehand therefore needs to be traded-off with the achieved speed-up through the usage of the preconditioner. Therefore, we compare the different methods based on both the time required to compute the preconditioners and the time needed to solve the preconditioned linear equation systems in the primal-dual algorithm. Note that in some occasions, the primal-dual algorithm using preconditioners, despite converging more rapidly, will take more iterations than the primal-dual without preconditioning.

5.1 Syntethic IPM problems

We evaluate the effectiveness of different preconditioners on a dataset comprising synthetic network-flow problems generated using `pynetegen`, a Python module for generating random network flows problem instances based in NETGEN algorithm [9]. `Pynetgen` provides a versatile set of tools for creating, manipulating, and studying network structures, allowing us to create realistic synthetic network-flow scenarios for our evaluation. We have considered three sizes of problems that are summarized in Table 1.

Problem size	**Small**	**Medium**	**Large**
Number of nodes limits	[20, 99]	[100, 499]	[500, 1500]
Number of sources/sinks limits	[1, 5]	[5, 10]	[10, 25]

Table 1: Sizes for the generated network-flow problems. The number of edges for the problems is calculated based on the number of nodes and source/sink limits.

As a result of the transformation of a randomly generated network-flow problem into an LP problem, the matrix obtained in the linear system (6) using the KKT conditions presents a sparsity pattern that is shown in Figure 1.

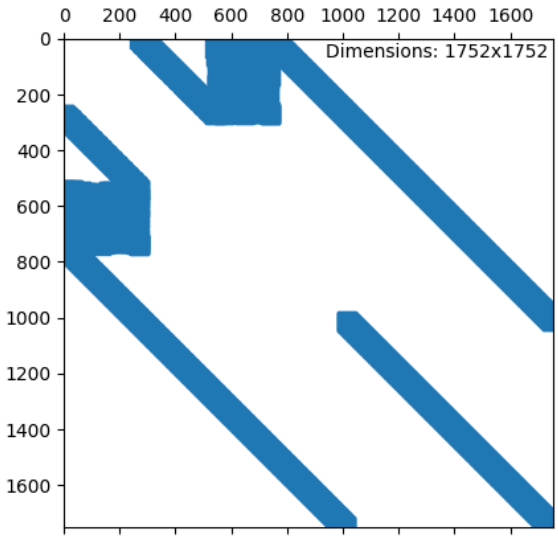

Figure 1: Matrix sparsity pattern. Blue represents non-zero elements.

Results

The results for ILU and ILUT [10, 11] preconditioners on small synthetically generated problems are shown in Table 2. We can see that the preconditioned primal-dual algorithm is able to outperform the direct method significantly in terms of solving time.

Solver	Preconditioner	$\left(\|r_{\text{primal}}\|_2^2 + \|r_{\text{dual}}\|_2^2\right)^{\frac{1}{2}}$	P-time ↓	PD-time (iter.) ↓	Failure rate ↓
Direct	None	2.45e−6	-	0.58 (14.53)	0.0
GMRES	None	2.29e−6	-	14.67 (15.52)	0.03
GMRES	ILU	2.61e−6	7.72e−6	0.25 (14.46)	0.0
GMRES	ILUT	2.64e−6	6.06e−6	0.20 (14.57)	0.2

Table 2: Averaged results for 30 synthetic small network-flow problems. The 1st column lists whether the solution has been obtained directly or via the GMRES solver. The 2nd column lists the different preconditioners used to solve the system in each iteration of the primal-dual algorithm. The 3rd column lists the combined primal and dual feasibility errors after convergence. The remaining columns list performance-related figures for the preconditioned primal-dual algorithm: 4th column lists the total computation time for the preconditioners in the primal-dual algorithm (P-time), 5th lists the time it takes for the primal-dual algorithm to converge (PD-time) neglecting the time it takes to calculate the preconditioners and the number of iterations required. Finally, the 6th column is the experiment's failure ratio. All times are in seconds. Arrows indicate whether higher/lower is better.

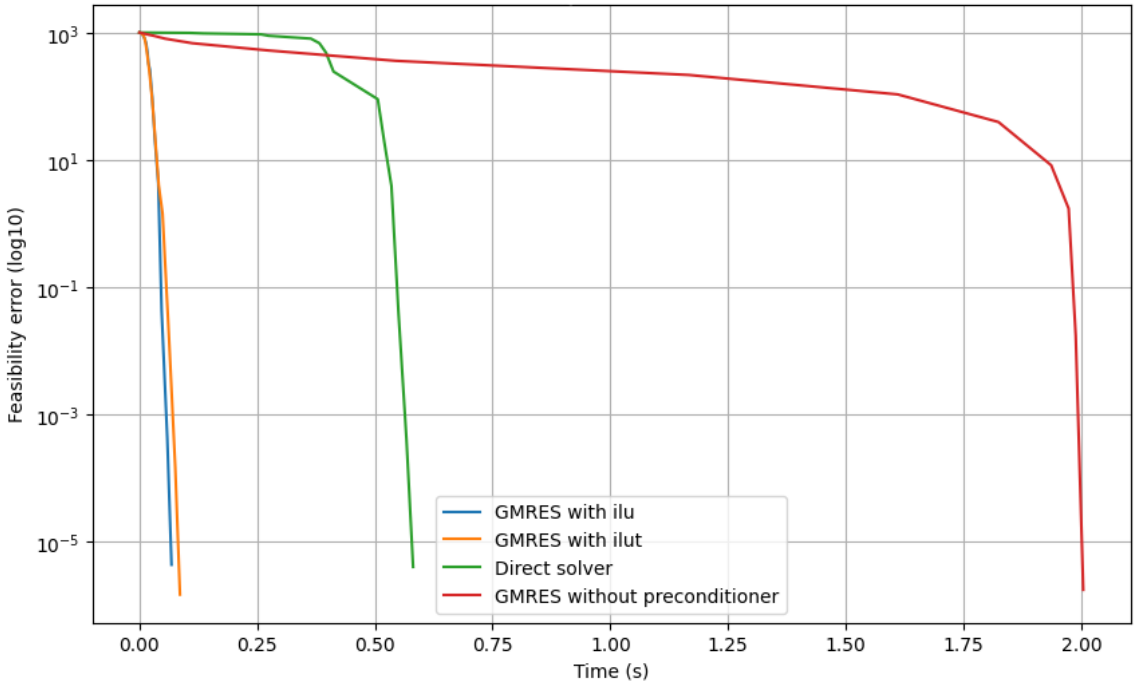

Figure 2: Convergence of each method over time.

The solution times varied quite a bit, especially for GMRES without a preconditioner. It has an average of 14 seconds, but as can be seen in Figure 2 it can be much faster than that in some instances. It had a fastest solve time of 0.05 and a slowest solve time of 105.05. For comparison, the direct solver had a maximum solve time of 3.32.

The BICGSTAB solver was implemented as well with good results, however the GMRES solver outperformed BICGSTAB in most cases so it was not included.

Numerical problems

IPMs rely on a primal-dual approach and follow a central path toward the optimal solution. However, as we have observed in the experiments, numerical issues can arise, especially when approaching the feasible region boundary along this path.

As discussed in Appendix B, IPMs operate by transforming the original problem into a sequence of unconstrained problems that move toward the optimal solution across a central path in the feasible region. This is achieved by introducing logarithmic barrier functions to penalize infeasibility by blowing up near the boundary of the feasible region.

As one approaches the boundary along the central path, the KKT matrix, which involves the Hessians and Jacobians of the original problem and its dual, can become increasingly ill-conditioned. We have encountered this situation when at some iterations, the KKT matrix becomes close to singular or has a very large condition number, amplifying the effects of round-off errors in calculations and making it sensitive to numerical errors.

Another numerical problem we have encountered is a result of decreasing the barrier parameter μ along the central path balancing the trade-off between the objective function and the barrier function penalizing infeasibility. As it becomes small near the boundary, it indicates that the optimization problem is approaching feasibility and small μ values can amplify rounding errors and lead to numerical instability during matrix factorizations or iterative solvers used within the IPM.

The issues mentioned above have been common when dealing with medium and large problems. When dimensions increase, maintaining accuracy becomes challenging due to finite precision in floating-point arithmetic. The varying magnitudes of values in larger problems involve more variables, constraints, and computations, resulting in numerically demanding calculations leading to significant disparities, causing the accumulation of round-off errors during computations. This limitation impacts numerical stability, compromises algorithmic convergence, and results in a loss of accuracy in computed solutions.

6 Discussions

Diversifying problem types One limitation of our project is that we have only considered standard network-flow problems. Further thinking would be to consider a limited capacity for each node. Moreover, exploring other problems such as resource allocation problems within constrained environments or optimizing air traffic control systems considering limited airspace capacities could provide valuable insights.

Addressing computational challenges The study confronted substantial computational challenges, especially with larger-sized problems. Despite employing the variety of strategies mentioned in Appendix E, achieving consistent improvements in convergence across all experiments remained elusive.

Exploring feasible initial points and hyperparameters One intriguing avenue for future investigation involves devising methods for generating feasible initial points that adhere to the constraints. Furthermore, optimizing the hyperparameters within the primal-dual algorithm, such as experimenting with alternative alpha selections beyond the standard backtracking line search, could present a promising approach to enhancing convergence rates.

7 Related work

The search for higher efficiency in solving existing numerical problems requires attention to the optimization of preconditioners for the GMRES solver. Notably, recent strides in efficient preconditioning methodologies have been explored [12], which has shown new strategies that can greatly improve the GMRES solver's performance in numerical calculations.

The application of the primal-dual algorithm stands as a notable approach. Nonetheless, it is imperative to acknowledge the existence of alternate IPM variations such as the primal-corrector algorithm [5]. This variant has shown new strategies that can greatly improve the GMRES solver's performance in numerical calculations.

While our project predominantly relies on the interior point methods for addressing linear optimization problems, it is crucial to recognize that many of the engineering problems involve non-linear programming complexities. In this context, the work on non-linear optimization techniques employing the interior method, becomes highly pertinent [13].

8 Conclusions

In this project, we focused on exploring the efficacy of various preconditioners in conjunction with interior point methods. Our approach involved constructing linear programming problems derived from network flow problems and employing the primal-dual algorithm for their resolution. Through the application of multiple preconditioners such as ILU and ILUT in combination with the GMRES solver during the linear equation-solving phase of the primal-dual method, we observed notable enhancements in solving efficiency. Nevertheless, this investigation identified numerical challenges and other pertinent issues that require further investigation and resolution in subsequent studies. This work lays a foundation for future research aimed at addressing these remaining complexities and advancing the application of preconditioners in optimizing interior point methods.

References

[1] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[2] S. Banert, J. Rudzusika, O. Öktem, and J. Adler, "Accelerated forward-backward optimization using deep learning." arXiv preprint arXiv:2105.05210, 2021.

[3] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark." arXiv preprint arXiv:2103.12828, 2021.

[4] P. Häusner, O. Öktem, and J. Sjölund, "Neural incomplete factorization: learning preconditioners for the conjugate gradient method." arXiv preprint arXiv:2305.16368v1, 2023.

[5] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of computational and applied mathematics*, vol. 124, no. 1-2, pp. 281–302, 2000.

[6] T. Schuster, *The method of approximate inverse: theory and applications*, vol. 1906. Springer, 2007.

[7] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *Journal of computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

[8] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[9] D. Klingman, A. Napier, and J. Stutz, "Netgen: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems," *management science*, vol. 20, no. 5, pp. 814–821, 1974.

[10] E. Chow and Y. Saad, "Experimental study of ilu preconditioners for indefinite matrices," *Journal of computational and applied mathematics*, vol. 86, no. 2, pp. 387–414, 1997.

[11] Y. Saad, "Ilut: A dual threshold incomplete lu factorization," *Numerical linear algebra with applications*, vol. 1, no. 4, pp. 387–402, 1994.

[12] J. A. Loe and R. B. Morgan, "Toward efficient polynomial preconditioning for gmres," *Numerical Linear Algebra with Applications*, vol. 29, Dec. 2021.

[13] A. Forsgren, P. E. Gill, and M. H. Wright, "Interior methods for nonlinear optimization," *SIAM Review*, vol. 44, no. 4, pp. 525–597, 2002.

[14] L. Trevisan, "Stanford university - cs261: Optimization." Handout 15, 2011. https://theory.stanford.edu/~trevisan/cs261/lecture15.pdf.

[15] M. Bierlaire, *Optimization: principles and algorithms*. EPFL Press, 2015.

[16] B. Polyak, "Newton's method and its use in optimization," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1086–1096, 2007.

[17] R. Tibshirani, "Carnegie mellon university - 10-725/36-725: Convex optimization," 2019. https://www.stat.cmu.edu/~ryantibs/convexopt/scribes/primal-dual-scribed.pdf.

Appendix

A Network flow problem into LP problem

For the first constraint in (1), it is complex to construct the solution by dealing with all paths from the sources to the sinks, so we should consider the dual problem.

First, we need to use a single variable x_p to illustrate all paths from s to t. Let P be the set of such paths:

$$\begin{aligned} \text{maximize} \quad & \sum_{p \in P} x_p, \\ \text{subject to} \quad & \\ & \sum_{p \in P:(u,v) \in p} x_p \leq c(u,v), \quad \forall (u,v) \in E. \\ & x_p \geq 0, \qquad\qquad\qquad\quad \forall p \in P. \end{aligned}$$

Now construct the dual of the above problem. We use one dual variable $y_{u,v}$ for each edge $(u,v) \in E$, obtaining the following dual problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{(u,v) \in E} c(u,v) y_{u,v}, \\ \text{subject to} \quad & \\ & \sum_{(u,v) \in p} y_{u,v} \geq 1, \qquad \forall p \in P. \\ & y_{u,v} \geq 0, \qquad\qquad \forall (u,v) \in E. \end{aligned}$$

The dual variable $y_{u,v}$ of each edge in E, illustrates whether this edge is in the cut:

$$y_{u,v} = \begin{cases} 1, & u \in A, v \notin A, \\ 0, & otherwise, \end{cases}$$

where A is the minimum cut of the flow.

Then,

$$\sum_{u,v} c(u,v) y_{u,v} = \sum_{u \in A, v \notin A} c(u,v) = capacity(A).$$

The linear program assigns a weight to each edge, which we may think of as a "length," and the constraints specify that, along each possible path, s and t are at a distance of at least one. This means that dual variables are expressing a way of "separating" s from t and this can be seen as a linear programming relaxation of the minimum cut problem.

According to [14], for problem (1), we still have one variable for each vertex v (except s and t), denoted y_v, corresponding to the conservation constraints. We can now construct an optimization problem as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{(u,v) \in E} c(u,v) y_{u,v}, \\ \text{subject to} \quad & \\ & y_v + y_{s,v} \geq 1, \qquad\quad \forall v : (s,v) \in E. \\ & y_v - y_u + y_{u,v} \geq 0, \quad \forall (u,v) \in E, u \neq s, v \neq t. \\ & -y_u + y_{u,t} \geq 0, \qquad \forall u : (u,t) \in E. \\ & y_{u,v} \geq 0, \qquad\quad \forall (u,v) \in E. \end{aligned}$$

We also have the vertex dual variables $y_u, \forall u \in V, u \neq s, t$ We want the normal form of an LP, which means only equality constraints and non-negative variables, so we define a slack variable $y_u = a_u - b_u$ for each inequality and use it to substitute the vertex dual variables.

The linear problem is as follows:

$$\begin{aligned}
\min \quad & \sum_{(u,v)\in E} c(u,v)y_{u,v}, \\
\text{subject to} \quad & \\
& a_v - b_v + y_{s,v} - r_{s,v} = 1, && \forall v : (s,v) \in E. \\
& a_v - b_v - a_u + b_u + y_{u,v} - r_{s,v} = 0, && \forall (u,v) \in E, u \neq s, v \neq t. \\
& -a_u + b_u + y_{u,t} - r_{u,t} = 0, && \forall u : (u,t) \in E. \\
& y_{u,v}, r_{u,v} \geq 0, && \forall (u,v) \in E. \\
& a_u, b_u \geq 0, && \forall u \in V, u \neq s, t.
\end{aligned}$$

Let e be the number of edges, v the number of vertices, and s and t the number of sources and sinks respectively. We have the number of variables of $n = e + 2(v - s - t) + e = 2(e + v - s - t)$, number of equality constraints $m = e$. Then we can easily find the shape for the required

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

We control the first e elements in variables are the $y_{u,v}$, the last e elements are the slack variables $r_{u,v}$ for each equality constraint, and the middle $m - 2e$ variables are the a_u, b_u. So the value of c is $(c_1, c_2, \dots, c_e, 0, \dots)$, A is the corresponding coefficients of all variables and b is like $(1, 0, \dots, 0)$.

B From the LP problem to the primal-dual algorithm

Let us derive the dual problem for (2) by first writing the Lagrangian:

$$\mathcal{L}(x, \lambda, s) = c^T x + \lambda^T (b - Ax) - s^T x = (c - A^T\lambda - s)^T x + \lambda^T b$$

The Lagrangian function is linear as a function of x and it is imperative to avoid values of the penalty parameter that generate unbounded problems. If the Lagrangian function is unbounded, it can indeed imply that the optimization problem does not have a finite solution or a well-defined optimum. This is because the Lagrangian function can take on arbitrarily large positive or negative values, which makes it impossible to find a finite optimum.

The boundedness of the Lagrangian function ensures that the solution space remains feasible and finite, allowing for a clear, achievable optimum or set of optima. Therefore, ensuring the boundedness of the Lagrangian function is essential for a well-defined and solvable optimization problem. Recovering the Lagrangian function, the only possibility for it to be bounded is if it is constant, i.e.,

$$c - A^T\lambda - s = 0$$

In this case the dual function is $q(\lambda, s) = \lambda^T b$ and the dual problem is written as

$$\max_{(\lambda,s)} b^T\lambda \ \text{ s.t. } \ A^T\lambda + s = c, \ s \geq 0, \tag{4}$$

where $s \in \mathbb{R}^n$ corresponds to the number of inequality constraints, which aligns with the dimension of x, given our constraint $x \geq 0$ is applied componentwise. Additionally, $\lambda \in \mathbb{R}^m$, corresponds to the number of equality constraints.

The set of feasible points is defined as:

$$\mathcal{F} = \{x \in \mathbb{R}^n | Ax = b, x \geq 0\}.$$

In this context, we define the set of interior points $\mathcal{S}$ as:

$$\mathcal{S} = \{x \in \mathbb{R}^n | Ax = b, x > 0\},$$

which we assume is non-empty. Note that the points in $\mathcal{S}$ are not interior points of the set $\mathcal{F}$, but instead, they are interior within the set $\mathcal{X} = \{x \in \mathbb{R}^n | Ax = b\}$.

In our case, $\mathcal{X} \subset \mathbb{R}^n$ is a closed convex set and the function $g(x) = -x$ defining the inequality constraints is a convex function. Then, for all $x \in \mathcal{F}$ and for all $\delta > 0$, there exists $\tilde{x} \in \mathcal{S}$ such that

$$||\tilde{x} - x|| \leq \delta.$$

This tells us that any feasible point can be arbitrarily well approximated by an interior point.

Definition. *A continuous function $B : \mathcal{S} \to \mathbb{R}$ is a **barrier function** if*

$$\lim_{x \in \mathcal{S},\ g(x) \to 0} B(x) = +\infty,$$

where $g(x)$ penalizes the inequality constraints $x \geq 0$.

Consider the non-smooth exact transition of inequality constraints to the criterion:

$$\min_{x \in \mathcal{S}} \ c^T x - \mu \sum_{i=1}^{n} I_{x \geq 0}(x). \tag{5}$$

where the indicator function $I_{x \geq 0}(x)$ acts as an indicator function which equals 0 if x is non-negative componentwise (i.e., $x_i \geq 0$ for all i) and equals ∞ otherwise. The sum penalizes the objective based on violations of the non-negativity constraint for each component of x.

This is the point at which the log-barrier function becomes useful. The goal is to use it to approximate the indicator function as follows:

$$\min_{x \in \mathcal{S}} \ f(x, \mu) \stackrel{def}{=} c^T x - \mu \sum_{i=1}^{n} \log(x_i),$$

where log denotes the natural logarithm and $\mu > 0$ denotes the barrier parameter.

By using a logarithmic barrier function to account for the bounds $x \geq 0$, we obtain the parametrized optimization problem

$$x(\mu) = \arg\min_{x \in \mathcal{S}} \ c^T x - \mu \sum_{i=1}^{n} \log(x_i). \tag{6}$$

Definition. *Consider the linear problem*

$$\min_{x} \ c^T x \ \text{ subject to } \ Ax = b, x \geq 0.$$

*The **primal central path** is the curve, parameterized by μ, described by (6).*

Because the logarithmic function requires its arguments to be positive, the primal central path needs to lie in the interior of $\mathcal{S}$. It is well-known that for any sequence $\{\mu_k\}$ with $\mu_k \downarrow 0$ (the trajectory followed by points $x(\mu)$ constitute the central path), all limit points of $\{x(\mu_k)\}$ are solutions of the original LP problem (2).

On the other hand, when $\mu \to \infty$, the objective function $c^T x$ plays no role and only the barrier on the constraints is minimized. In this case, we obtain the point x_∞, called the ***analytical center*** of $\mathcal{F}$. See page [15, p. 423] for a detailed explanation.

It is important to note that the identification of the central path is a difficult problem. For each value of μ, we must indeed solve a non-linear optimization problem. Therefore, the interior point algorithms use the central path as an indicator of the direction to progress toward the optimal solution, but without trying to follow it exactly. It means that for a given μ, the corresponding non-linear optimization problem is solved approximately.

The optimality conditions, also called Karush-Kuhn-Tucker conditions or KKT conditions for the barrier problem are:

$$\begin{aligned} Ax - b &= 0 \quad \text{(primal constraint)} \\ A^T\lambda + s &= c \quad \text{(dual constraint)} \\ XS\mathrm{e} &= \mu\mathrm{e} \quad \text{(complementarity constraint)} \\ (x, s) &> 0 \quad \text{(primal constraint and dual constraints).} \end{aligned} \tag{7}$$

Where $\mathrm{e} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$, and the matrices X and S are defined as

$$X = \begin{bmatrix} x_1 & 0 & \cdots & 0 & 0 \\ 0 & x_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & x_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & x_n \end{bmatrix}, \quad S = \begin{bmatrix} s_1 & 0 & \cdots & 0 & 0 \\ 0 & s_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & s_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & s_n \end{bmatrix}.$$

We note that the perturbed KKT(μ) conditions (7) derived for our barrier problem and the original KKT conditions for the linear problem are the same, except for the third one. In the perturbed KKT(μ) conditions the right-hand side is $\mu\mathrm{e}$ while for the normal KKT conditions we have $XS\mathrm{e} = 0$.[2]

We can thus characterize the optimal solution to the barrier problem using dual variables. We consider the primal and dual variables together and work in the space $\mathbb{R}^{n+m+n}$ with the variables (x, λ, μ). In this space, the feasible set is

$$\mathcal{F} = \{(x, \lambda, s) | Ax = b, A^T\lambda + s = c, x \geq 0, s \geq 0\},$$

and the interior points are

$$\mathcal{S} = \{(x, \lambda, s) | Ax = b, A^T\lambda + s = c, x > 0, s > 0\},$$

again assumed to be non-empty. The central path concept is also extended as follows.

Definition. *Consider the linear problem*

$$\min_x \ c^T x \ \text{ subject to } \ Ax = b, x \geq 0.$$

*The **primal-dual central path** is the curve described by $(x(\mu), \lambda(\mu), s(\mu))$ with $\mu \geq 0$, where $(x(\mu), \lambda(\mu), s(\mu))$ solves (7).*

The system (7) includes two sets of linear equations, a set of slightly non-linear equations ($XS\mathrm{e} = \mu\mathrm{e}$), and two sets of inequations. We proceed in the following manner:

1. Consider only the iterates in $\mathcal{S}$.
2. Primal-dual interior-point updates are motivated by a Newton's method step [16] for solving the nonlinear equations in (7). We ignore the inequalities and write the KKT conditions as a set of nonlinear equations

$$r(x, \lambda, s) = 0,$$

[2] See [15, pp. 443-444] for a detailed explanation about the difference between KKT(μ) and KKT.

where

$$r(x, \lambda, s) = \begin{bmatrix} Ax - b \\ A^T\lambda + s - c \\ XS\mathrm{e} - \mu\mathrm{e} \end{bmatrix} = 0. \tag{8}$$

This is a nonlinear equation in (x, λ, s), and hard to solve; so let us linearize, and approximately solve. Let $y = (x, \lambda, s)$ be the current iterate, and $\nabla y = (\nabla x, \nabla\lambda, \nabla s)$ be the update direction for the primal-dual algorithm to follow. Define

$$\begin{aligned} r_{\text{prim}} &= Ax - b \\ r_{\text{cent}} &= XS\mathrm{e} - \mu\mathrm{e} \\ r_{\text{dual}} &= A^T\lambda + s - c \end{aligned}$$

the primal, central, and dual residuals at current $y = (x, \lambda, s)$.

3. Calculate the step along the direction such that no iterate leaves $\mathcal{S}$. Now we make our first-order approximation

$$0 = r(y + \nabla y) \approx r(y) + \nabla r(y)\nabla y$$

and we want to solve for ∇y in the above.

The Jacobian matrix of the system (8) is:

$$\nabla r(x, \lambda, s)^T = \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix}.$$

The Newton equations for an iterate (x, λ, s) can be expressed as:

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \nabla x \\ \nabla\lambda \\ \nabla s \end{bmatrix} = -r(x, \lambda, s) = - \begin{bmatrix} Ax - b \\ A^T\lambda + s - c \\ -XSe + \mu e \end{bmatrix}. \tag{9}$$

The solution $\nabla y = (\nabla x, \nabla\lambda, \nabla s)$ is our primal-dual update direction. Note that the update directions for the primal and dual variables are inexorably linked together.

The methods in the primal-dual framework recognize the importance of the path of solutions $x(\mu)$ to (5) in the design of algorithms but differ from the non-dual approach (e.g. primal barrier algorithm) in that they treat the dual variables explicitly in the problem, rather than as adjuncts to the calculation of the primal iterates. In this project, we will focus on the primal-dual algorithm.

B.1 Stopping criterion

In interior-point methods, the algorithm iteratively approaches an optimal solution while maintaining the feasibility within the interior of the feasible region. The termination criterion of the algorithm that we have chosen [5, 17] is based on three factors:

- $\|r_{\text{primal}}\|_2$ (*Primal Feasibility Error*): It measures the extent to which the current solution $x^{(k)}$ satisfies the primal constraints $Ax = b$. The primal feasibility error is the ℓ_2-norm of the residual $r_{\text{primal}} = Ax^{(k)} - b$.
- $\|r_{\text{dual}}\|_2$ (*Dual Feasibility Error*): It measures the extent to which the current solution satisfies the dual constraints $A^T\lambda^{(k)} + s^{(k)} = c$. The dual feasibility error is the ℓ_2-norm of the residual $r_{\text{dual}} = A^T\lambda^{(k)} + s^{(k)} - c$.

- $\eta^{(k+1)}$ (*Duality Gap*): It represents the difference between the objective function values of the primal and dual problems at the k-th iteration, given by $\eta^{(k+1)} = c^T x^{(k)} - b^T \lambda^{(k)}$. This expression is further simplified to $x^T s$ as follows:

$$\begin{aligned} c^T x - b^T y &= x^T c - x^T A^T y \quad \text{(using the fact that } Ax = b\text{)} \\ &= x^T (c - A^T y) \\ &= x^T s. \end{aligned}$$

At the optimal solution, strong duality holds, and the duality gap is zero, indicating that the primal and dual objective values coincide. If the solution is not optimal, the duality gap will be positive, indicating the amount by which the primal and dual solutions differ.

The termination criterion for the algorithm is a combination of these three factors being within a specified tolerance δ. The algorithm can stop when both the duality gap and the combined primal and dual feasibility errors are less than or equal to δ, ensuring the solution is close to being optimal and feasible within the specified tolerance. The stopping criterion is formally given by:

$$\text{Stop if } \eta^{(k+1)} \leq \delta \text{ and } \left(\|r_{\text{primal}}\|_2^2 + \|r_{\text{dual}}\|_2^2\right)^{\frac{1}{2}} \leq \delta.$$

This criterion ensures that the primal and dual solutions are within the acceptable range of optimality and feasibility as determined by δ.

In addition, a maximum number of iterations was implemented, to ensure that in cases where the error does not converge or does not converge fast enough, the algorithm terminates anyway.

This stopping criterion - based on primal feasibility error, dual feasibility error, and duality gap - was chosen because it balances precision and computational efficiency. The primal and dual feasibility errors ensure that the solution adheres to the constraints of the primal and dual problems. The duality gap measures the convergence towards optimality. By requiring all these factors to be within a specified tolerance, the algorithm ensures a solution that is both feasible and close to optimal. The additional implementation of a maximum number of iterations acts as a safeguard against excessive computation time where convergence is slow or unattainable.

B.2 Feasible points

The definition of the feasible point is the point whose variables satisfy all constraints, i.e., the points in $\mathcal{S}$. Note that if our initial point $(x, \lambda, s)^T \in \mathcal{S}$, then this point is feasible and

$$r(x, \lambda, s) = \begin{bmatrix} 0 \\ 0 \\ XSe - \mu e \end{bmatrix}.$$

However, in our particular problem, there does not exist a feasible point of the dual variables. For a fixed λ, due to the format of the normal form of the LP, we have that in each constraint, some rows of A^T are opposite, so the $A^T\lambda$ definitely holds some opposite number, which means they should be opposite or 0. However, we have the equation $A^T\lambda + s = c$, and c contains 0, which means we cannot guarantee an initial vector s to be both positive and feasible.

We solved this by including the feasibility constraints into the linear system (8). This means we only need initial s and x to be positive. To make it easier to converge, we used all 1's for the initial variables x, λ, and s.

B.3 Step size

Following the implementation of [17], at each step, we need to find α and ensure we arrive at $y^+ = y + \alpha \nabla y$, i.e.,

$$x^+ = x + \alpha \nabla x, \quad \lambda^+ = \lambda + \alpha \nabla \lambda, \quad s^+ = s + \alpha \nabla s.$$

The specific choice of α has two main goals:

- Maintain both primal and dual feasibility: $x_i > 0$, $s_i > 0$.
- Reduce $||r(x, \lambda, s)||$.

A multi-stage backtracking line search is used for this purpose. We start with largest step size $\alpha_{\max} \leq 1$ that makes $s + \alpha \nabla s \geq 0$:

$$\alpha_{\max} = \min\{1, \min\{-s_i / \nabla s_i : \nabla s_i < 0\}\}.$$

Then, with parameters β, $\gamma \in (0, 1)$, we set $\alpha = 0.99\alpha_{\max}$, and:

- Update $\alpha = \gamma\alpha$, until $x^+ > 0$ componentwise.
- Update $\alpha = \gamma\alpha$, until$||r(x^+, \lambda^+, s^+)|| \leq (1 - \beta\alpha)||r(x, \lambda, s)||$.

C Preconditioners

Specifically if we have a linear system $Ax = b$, a preconditioner M would be applied such that $M^{-1}Ax = M^{-1}b$. A preconditioner can be chosen to approximate A^{-1} which would mean $M^{-1}A$ approximates the identity matrix with eigenvalues around 1, which brings the condition number closer to 1, leading to a well-conditioned matrix where iterative algorithms tend to perform more efficiently.

In the realm of preconditioners, some well-known examples include Jacobi, Successive Over-Relaxation (SOR), and Incomplete Lower-Upper Decomposition (ILU). Each has its unique strengths, making them suitable for different kinds of mathematical problems.

The choice of the preconditioner is problem-dependent, as no single preconditioner can universally optimize all computational problems due to the diverse nature of matrix characteristics. While identifying an effective preconditioner requires time and effort, this initial investment can significantly reduce the overall computational time of solving the system.

It is important to weigh this benefit against the effort and time required for its implementation. For some problems, especially those where the improvement in computation time is marginal, the resources invested in developing and applying a preconditioner may not be justified. Typically preconditioners with iterative algorithms outperform direct solvers for large, sparse linear systems.

The implemented preconditioners in the project are the following:

- **Jacobi or diagonal preconditioner**: The preconditioner is constructed using only the diagonal elements of the matrix A. It is effective when A has dominant diagonal entries, simplifying the preconditioning process.
- **ILU (incomplete LU decomposition)**: An approximation of the LU decomposition for sparse matrices. It computes only a subset of the lower and upper triangular matrix elements, enhancing computational efficiency.
- **ILUT (incomplete LU factorization with threshold)**: Extends ILU by incorporating a dropping tolerance, which controls the sparsity level of the LU factors, balancing memory usage and accuracy.
- **Spectral shift**: Modifies the spectrum of the matrix by shifting its eigenvalues. This can improve the matrix's condition and enhance the convergence of iterative solvers.
- **NeuralIF**: A neural network trained to predict a suitable preconditioner $P = \Lambda_\theta(A)\Lambda_\theta(A)^T$. The function Λ_σ maps A to a lower triangular matrix with strictly positive elements on the diagonal. The data-driven preconditioner is trained by minimizing the distance between the learned factorization and the original input matrix A using the Frobenius norm across multiple problems [4].

The NeuralIF preconditioner was used to train on our data, with no convergence. It was applied to predict but threw errors. After investigating it was discovered that it yielded infinite values when applied to our generated problems. After implementing it on symmetric and positive-definite matrices with the same methodology, it was successful, indicating that the issue is the matrix properties of our system.

The Jacobi or diagonal preconditioner did not improve the solvers on our system and led to errors, likely due to our matrix not being diagonally dominant. See Figure 1.

The Spectral preconditioner made the solver slower, and sometimes failing, and never improved the convergence.

The only preconditioners that worked well were the ILU and ILUT preconditioners. They were well suited for the problem and more computationally efficient than the direct solver.

D Implemented solvers

- The **LU decomposition** method decomposes matrix A into LU, where L and U are lower and upper triangular matrices, respectively. This approach is particularly effective for dense, non-singular square matrices and provides an exact solution to $Ax = b$. It was implemented using `np.linalg.solve` from NumPy, an efficient solver for linear systems of equations. While this library is optimized, it can still be slow since it does not approximate the solution but rather computes it directly. This method is referred to as the direct solver in the results.
- The **generalized minimal residual (GMRES)** method is an iterative method suitable for large, sparse linear systems. It builds a Krylov subspace at each step and minimizes the residual over this subspace, providing an approximate solution to $Ax = b$. This method is particularly useful when direct methods become computationally expensive. It was implemented using `spla.gmres` from SciPy's `linalg` module.
- The **biconjugate gradient stabilized (BICGSTAB)** method is an iterative solver for large, sparse, non-symmetric linear systems. It combines the features of the biconjugate gradient method and GMRES, offering faster and more stable convergence in some cases. It was implemented using `spla.bicgstab` from SciPy's `linalg` module.

E Strategies to avoid numerical issues

In this section, we list the different strategies we have followed in order to avoid the numerical problems encountered in the application of the preconditioners:

- **Analyzed matrix properties:** Understanding the characteristics of our matrix (non-symmetric, non-positive definite, etc.)
- **Regularization:** Adding a small value to the diagonal entries to prevent singularity.
- **Handling small/zero diagonal values:** Replacing very small or zero diagonal values with standard values in order to alleviate numerical instability issues by ensuring better conditioning of the system.
- **Scaling the matrix of the system:** Applying row and column scaling to normalize the system, enhancing numerical robustness.
- **Implementing reverse Cuthill-McKee algorithm:** Reordering the matrix to a band-diagonal form to improve sparsity pattern exploitation.
- **Precision arithmetic:** Utilizing double or extended precision computations to reduce round-off errors.
- **Different iterative solvers:** Switching between different iterative solvers apart from GMRES. The considered ones were BICGSTAB, LGMRES, TFQMR, and CGS solvers.
- **Loosening termination criterion:** Adjusting convergence thresholds to balance computational stability with solution accuracy.