Optimization Methods for Clustering

Members: Luca Tusini (2092227), Davide Christian Mancosu Bustos (2089208), Karim Eugenio Hamdar (2092041)

Summary

Intro

This is a gentle explanation of what we have done for our group project. The essay is divided into three main parts: "Paper Review," "Numerical Experiments," and "Conclusion."

In each section, we will describe the main problems, objectives, and results that occurred during the work, providing a clear understanding of our progress. The "Paper Review" will comprehensively, yet briefly, summarize the given articles, focusing on the parts most relevant to our work.

In the "Numerical Experiments" section, we will describe the problem we addressed—the Maximum Clique Problem—and how we managed to solve it using the Frank-Wolfe algorithm and its variants.

We will conclude by presenting and analyzing the results obtained from the chosen test set: the DIMACS benchmark. At the end of this essay, we will analyze the overall work and results, offering commentary on our achievements and possible improvements.

1 Paper's review

In the following section, we will briefly present the three assigned papers, focusing on the parts most important for the project.

1.1 Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization

The main objective of the article is to provide and prove convergence results for Frank-Wolfe-type algorithms based on the gap between primal and dual functions. To achieve this, the paper first introduces the context of the analysis: constrained convex optimization problems. It then discusses well-known variants of the classic Frank-Wolfe (FW) algorithm and the concept of "curvature." The curvature constant C_f is a critical measure in this convergence analysis, quantifying how much a convex, differentiable function deviates from its linear approximation.

The article presents two main theorems:

1. Primal Convergence: This theorem relates to the classic result that the FW algorithm has a sublinear convergence rate. The key difference here is that the authors account for the accuracy Δ of the solution to the linear subproblem and C_f

2. Primal-Dual Convergence: This is the main result proposed by the authors. It proves that after a number of iterations, the FW variants introduced have an iterate for which the duality gap is bounded by a quantity computed using C_f, Δ, and a constant β. The theorem also highlights that this result holds even if the linear subproblem is solved only approximately.

The article introduces the concept of "atomic sets." If the feasible set D is the convex hull of another set A, then the gauge function of D is called the atomic norm defined by A. The FW-type algorithms presented are particularly suitable for solving constrained convex problems in such domains. The authors then explain some of these domains and comment on the computational complexity of the related linear subproblems. Among these, we were particularly interested in the one concerning vectors.

Optimization over Vectors:

The application of FW-type algorithms to the unit simplex and L_1-ball for solving constrained convex optimization problems is well-studied. The algorithm naturally encourages sparse solutions: at each iteration, it will add at most one new non-zero coordinate to x, and the linear subproblems involve finding the largest entry of the gradient. In both scenarios, the cost per iteration of the FW-type algorithm is proven to be linear.

The article concludes by reporting some improvments achieved by this new framework and briefly discussing some special cases of factorized matrix norms domains.

1.2 On the Global Linear Convergence of Frank-Wolfe Optimization Variants

Motivated by the poor convergence rate that vanilla Frank-Wolfe (FW) achieves in some scenarios, the paper focuses on several of its improved variants, demonstrating that they all exhibit global linear convergence, even under weaker conditions than the strong convexity of the objective. Moreover, the authors provide an interpretation of the constant in the convergence rate as the product of two numbers: the condition number of the objective function and the condition number of the constraint set.

As in the previous paper, the very first part of this paper concerns an introduction to the FW algorithm and a specification of the problem framework, which is the same as in the previous one. Recognizing that the classic FW algorithm exhibits the so-called "zig-zagging phenomenon" when the solution lies on the boundary of the feasible set, the paper introduces some improved variants. Here, we focus primarily on:

- Away-Steps FW: This variant considers the possibility, at each iteration, of moving away from an active atom in $\mathrm{S(t)}$ (the set of already explored vertices), addressing the aforementioned problem.
- Pairwise FW: Very similar in structure to the previous algorithm, with the main difference being that mass weight is now transferred between two atoms: from the atom identified by the Away-Steps direction to the one selected by the FW.

The article proceeds by introducing an initial sketch of proofs that highlight the main contributions of the authors. Regarding the linear convergence proof for the Away-Steps FW, it is stated that the worst-case inner product between the Pairwise FW direction and any potential negative gradient direction can be lower-bounded by a quantity called the Pyramidal width of A, which depends only on the geometry of the feasible set M.

Next, the authors present the two main theorems proposed:

- **Theorem 1**: This first theorem states that the suboptimality hth_tht decreases geometrically at each iteration ttt that is a "good step" (i.e., not a drop step or a swap step). For a "good step," the suboptimality decreases as follows:

$$h_{t+1} \leq (1-\rho)\, h_t, \text{ where } \rho = \frac{\mu}{4L}\left(\frac{\delta}{M}\right)^2$$

Theorem 2: This second theorem focuses on the duality gap and states that g_t^{FW} is upper bounded by a quantity that takes into account the primal error h_t.

Moving on, multiple notions and properties about the pyramidal width are presented. Among them, the pyramidal width of the unit simplex with dimension d is the same as its width.

The article concludes with results from two experiments and an appendix in which some of the previous topics are covered in more depth, along with the proofs of previous lemmas and theorems.

1.3 Frank–Wolfe and friends: a journey into projection-free first-order optimization methods

This survey paper provides an in-depth exploration of the Frank-Wolfe (FW) algorithm and its variants, highlighting their importance in projection-free first-order optimization.

The authors begin by presenting the classic FW method and its fundamental properties, including its $O(\frac{1}{k})$ convergence rate for convex objectives and its ability to produce sparse solutions. The paper then delves into various applications of FW methods, ranging from traffic assignment problems to submodular optimization and machine learning tasks such as LASSO, matrix completion, and SVM training. A key focus is on the algorithm's efficiency when dealing with atomic domains and structured constraints.

Notably, the survey discusses the application of FW methods to the problem of finding maximal cliques in graphs, formulating it as an equivalent non-convex standard quadratic optimization problem over the simplex. For a simple undirected graph $G = (V, E)$, the maximal clique problem is formulated as an equivalent non-convex standard quadratic optimization problem (StQP):

$$max\ \{x^T A_G x + \frac{1}{2}\|x\|^2 : x \in \Delta_{n-1}\}$$

where A_G is the adjacency matrix of G, and Δ_{n-1} is the standard simplex. This formulation allows Frank-Wolfe variants to be effectively used for identifying maximal cliques. The paper notes that due to the specific structure of this problem, FW methods can be fruitfully applied to find maximal cliques (in our project we mainly focus on the maximum clique problem). This formulation allows FW variants to be effectively used for identifying maximal cliques, showcasing the algorithm's versatility in tackling combinatorial problems.

The survey also discusses important theoretical aspects, such as the duality gap, affine invariance, and the curvature constant, which play crucial roles in the convergence analysis. Furthermore, the authors present recent developments in FW variants, including Away-Step FW, Pairwise FW, and Fully-Corrective FW, explaining how these modifications address the zigzagging phenomenon and potentially achieve linear convergence rates under certain conditions.

The paper concludes by examining extensions of FW methods to various optimization frameworks, including block coordinate, distributed, and trace norm optimization, showcasing the versatility and ongoing relevance of these algorithms in modern optimization challenges.

2. Numerical Experiments

In this next section we explain and present in deep the experiments carried out along with the results obtained. We will discuss such results and conclude with a global conclusion about the work done.

2.1 Problem Description

The problem we deal with in our project is the Maximum Clique Problem (MCP) which is a fundamental problem in graph theory. It involves finding the largest clique within an undirected graph where a clique is a subset such that there exists an edge connecting every two distinct vertices. The size of the clique, or the number of vertices it contains, is the criterion for finding the maximum one.

MCP has numerous applications across various fields such as Social Network Analysis, Telecommunication Networks or Biochemistry, making the problem attractive for researches. One of the biggest challenges of MCP is that it is NP-hard, meaning that finding an exact solution can be computationally infeasible for large graphs. It is still possible to use an iterative algorithm to find approximate solutions.

The formulation of MCP used in this study is based on the continuous quadratic programming approach proposed by Motzkin and Straus. This approach translates the problem into a continuous quadratic programming optimization problem of the form:

$$max(x^T Ax)$$
$$subject\ to\ x \in \Delta$$

where:

- Δ is the n-dimensional simplex defined by $\Delta := \{x \in R^n : x \geq 0\ and\ 1^T x = 1\}$.
- A is the adjacency matrix of the graph G.

2.2 Frank-Wolfe

In the context of MCP, the solution is expected to lie on the boundary of the feasible region and is expected to be sparse. Given this, the fact that the problem formulation is quadratic and that we are dealing with a structured feasible set, the Unit Simplex, the Frank-Wolfe algorithm aligns very well to the problem.

To apply the Frank-Wolfe algorithm, which is typically formulated for minimization problems, we transformed the maximization problem into a minimization one. This can be done simply by minimizing the negative of the objective function:

$$min - (x^T Ax)$$
$$subject\ to\ x \in \Delta$$

This transformation allows us to apply all the knowledge acquired during the course without having to cope with conversions of any type since the two formulation of the problem are completely equivalent.

While the classic Frank-Wolfe (FW) algorithm is useful, it has limitations, especially when the solution lies on the boundary of the feasible region like in our scenario. Variants of the FW algorithm are designed to address these limitations, and we will implement two of them:

1. **The Away-Step FW**: The Away-Step Frank-Wolfe algorithm modifies the original FW algorithm by incorporating an additional step that allows the algorithm to move away from certain directions. This is particularly useful when the optimal solution is at a vertex of the feasible region. The AFW algorithm can improve convergence rates by including steps that retract away from vertices of the polytope, allowing for better handling of solutions on the boundary. The algorithm iterates between standard FW steps and away-steps, which involve choosing an away direction and potentially moving in that direction to reduce the objective function more efficiently.
2. **Pairwise FW**: The Pairwise Frank-Wolfe algorithm is another variant that addresses the boundary solution problem by considering pairs of vertices for updates. In each iteration, the algorithm simultaneously considers moving towards one vertex while moving away from another. This pairwise consideration enables the algorithm to make more significant progress in each step compared to the classic FW algorithm. The PFW algorithm can thus offer improved convergence properties, particularly in cases where the feasible region is a polytope and the solution lies on the boundary.

We now have to do a last consideration: in practice, the presence of "infeasible" local maximizers is a common issue. These are not characteristic vectors for cliques, and a clique cannot be recovered from them through any simple transformation. Such points can cause the algorithms to fail by terminating without producing a clique. To address this issue, a regularization term is added to the objective function.

Many different terms exist and have been proven to work; we focus on two of them:

$$1) \quad \Phi_B(x) \coloneqq \frac{1}{2}\|x\|_2^2,$$

$$2) \quad \Phi_2(x) \coloneqq \alpha_2 \sum_{i=1}^{n}\left(e^{-\beta x_i} - 1\right), \ with\ \beta > 0\ and\ 0 < \alpha_2 < \frac{2}{\beta^2}$$

The first term is the two-norm regularization function introduced by Bomze; the second one is an approximation of the following nonsmooth function: $f(x) = -\alpha \parallel x \parallel_0$, where $\parallel x \parallel_0$ represents the number of non-zero entries in x. In our work for α_2 and β we used, respectively, 0.05 and 5.

2.3 Structure of the work

Our experiments has consisted in implementing the classic FW algorithm along with its two main variants (the Away Step and the Pairwise FW) to the MCP. To do so we relied on the yet explain quadratic formulation of that problem taking into account, one per time, the two regularization term yet introduced.

We developted out work using the Python programming language. We split the implementation into two main parts: in the first we focus on solving the MCP with the two-norm regularization function (L2) while in the second we focused on the other regularization term (L0). The structure of both parts is the same: we implement first the classic FW algorithm, then the two variants. Both part present a final section in which are resumed the results obtained. A more general global comparison is provided at the end of this chapter.

The python files provided are three:

- "main.ipynb" which is the core code. In this version of the code a single initial node is tested and metrics like the CPU time and the duality gap are stored to be then plotted.
- "Dual_Main.ipynb": equal to the previous one but implement a stopping condition based on the duality gap. We used this version to study the convergence rates in the different scenarios.
- "TestALL.ipynb": also this version is equal to the first one but test all the vertices as initial node. The output returned is the size of the maximum clique found and for which node it has been found. We used this version to find the actual maximum clique reachable.

The reason behind the choice of code separation is due only to a better handling of the analysis and to keep the "main.ipynb" as clean as possible since it will be the one we care the most.

2.4 Test Set

In order to asses the performance of our implementations, we utilized three dataset out of the DIMACS benchmark (https://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark) set which is a well-established set of instances widely used in the research community for testing graph algorithms, among which the MCP.

Specifically, we focused on three families within this benchmark: Brock, Hamming, and Keller:

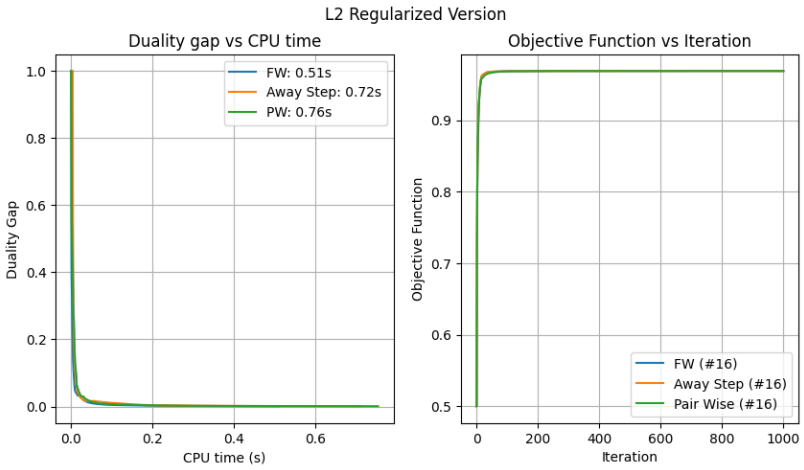
1) The Brock family are graphs generated randomly with cliques hidden among nodes that have a relatively low degree, making them difficult due to their dense and complex structures.
2) The hamming family are derived from the Hamming code used in error correction. These graphs are characterized by their vertices representing binary strings of a fixed length, with edges connecting vertices whose strings differ by a certain number of bits.
3) The keller family are graph based on Keller's conjecture related to tiling and covering spaces. These graphs are formed by vertices representing lattice points in multi-dimensional space, with edges connecting points at a unit distance apart.

The istances provided by the DIMACS site are in ".clq" file format which is a specific format used to represent graph data. In all the codes provided there exists an intial section entitle "Loading Data" create ad hoc to load the file in such format, retrive important information and create the adjacency matrix onto which later algorithms will work.

2.5 Results: CPU Time and Duality Gap

In this first section of analysis of the results we will focus on two main metrics: the CPU time taken by the algorithm to run and the behaviour of the duality gap over time. We perform the analysis first, without a stopping condition over the duality gap and then with it. The main objective of this part is to highlight differences in the nature of the algorithms and try to explain why.

We report in "figure 1" the behaviour of the metrics yet introduced for the Hamming8-4 instance divided for the two regularization term used. Plots are aslo correlated with the CPU time (in seconds) taken and the maximum clique found (#n). The line search used in this case is the "Diminishing Stepsize". We will report and focus only on these plots for two main reasons: is the only instance for which we found the largest known clique size and because in the end the behaviour of the metrics is the same for all istances so it would be non to informative to report multiples plots for different istances. What change in the end is the difference from the clique size found and the maximum known clique but this will be discussed in the next session.



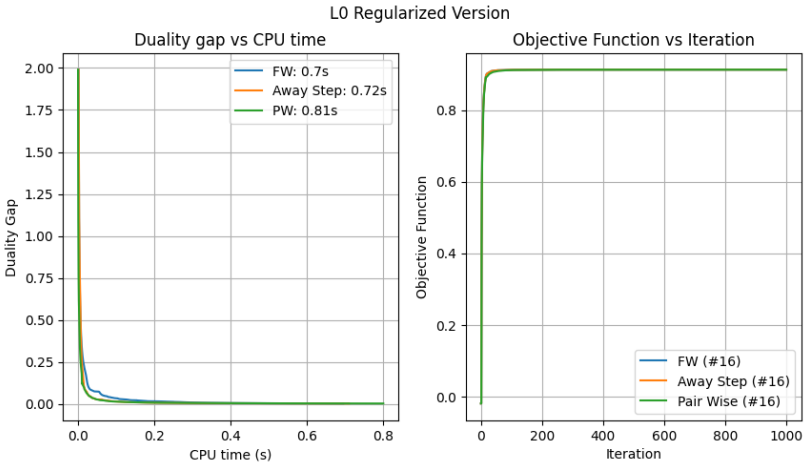


Figura 1: Performance metrics for Hamming8-4 Instance with "Diminishing Stepsize"

Moving on to the observations at first glance is easy to see that all the algorithms seems to perfoms well and almost identical. Intresting to notice is also the very strong "L" shape the mesaures shows. This points out the fact that, with this type of line search, is not necessary to let the algorithm runs for many iterations and for this reason we kept the number of iterations quite low: around 1000 even if just 500 is enough.

Concerning the CPU time all the algorithms takes more or less few seconds to run the 1000 iterations but of course this mesaures is not much informative since it heavly depends on the size of the input graph and also on the version of the FW considered. Infact it is natural that the two variants required more time since the computational effort is more: in both cases more operations are conparison are performed. For this reason later we analysis the scenario with a stopping conditions: to verify if the variants paying the price of more computation performs better in terms of convergence.

As a final comment to this preliminary part can be say that out of the three algorithms implemented: Away-Step and Classic FW seems to be the best one nonetheless Pairwise still remains very close.

In "Figure 2", we analyze the performance of the algorithms using the Lipschitz constant dependent stepsize for line search. As before this figure illustrates the metrics for the Hamming8-4 instance under this different line search strategy, separated by the two regularization terms used.

What is immediately notable is the strange shape the curves for the classic FW and the Away-step variant shows: both drops rapidly but then seems to return costant for then again decrease this time not so rapidly. Also the steep of the curves is not as strong as in the previous scenario meaning a slower optimization process. For this reason we increment the number of iteration up to 6000 also because in this way we highligth the fact that, as we excpeted for this line search, the classic FW does not perform well even if in the end it still finds the same maximum clique as the others. As said in the introduction of the line-search infact the "Diminishing Stepsize" is an usual choice for the classic FW, not for its variants.

What in the end this plots suggests is that the “Lipschitz constant dependent stepsize” line-search works but not as well as the first one introduced and so can still be considered a good alternative to the “Diminishing Stepsize” line-search.

Also the fact that we considered more iterations allows us to make some further comments. For example the plots highligth well how fast the algorithm are, notable the Pairwise is faster that the Away-Step. Finally, in this scenario both the variants of the FW works very well and even if the PW return more gentle curves the Away-Step again, should be considered the best one.

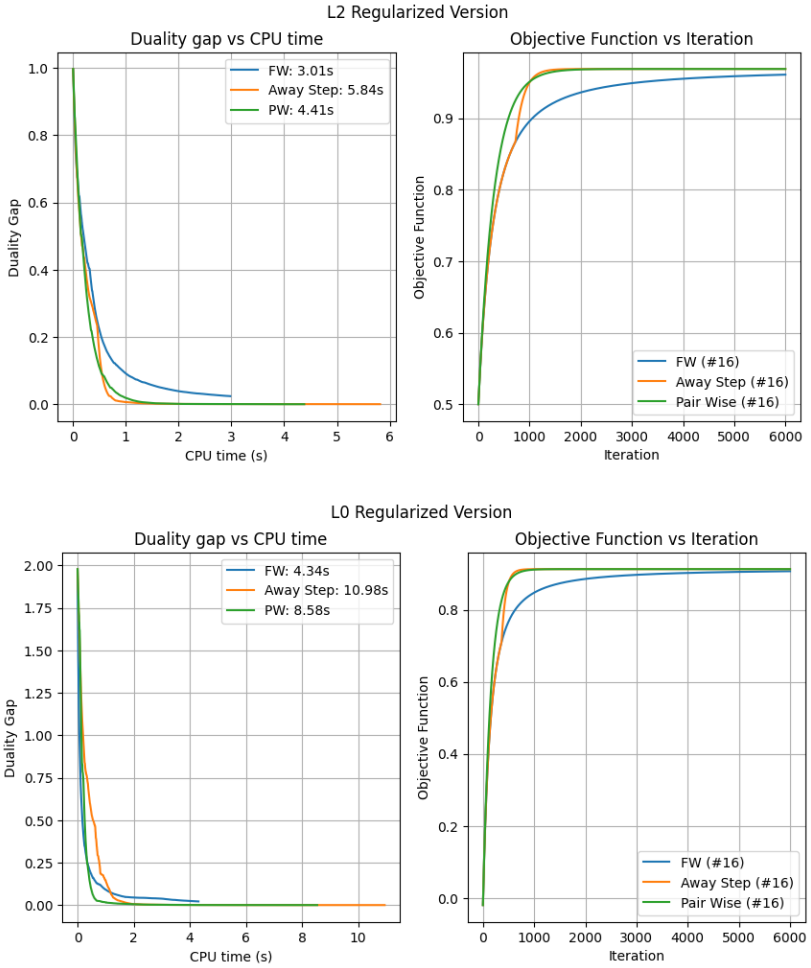


Figura 2: Performance metrics for Hamming8-4 Instance with “Lipschitz constant dependent stepsize”

In section dedicated to the line-search we also list the “Armijo Rule. We won’t discuss it anyway because we find out that the stepsize it computes at each iteratin was too small to be then seen in the improvment of the objective function. We tried several combinations for the parameters “delta” and “gamma” but none of them worked. For this reason and for the fact that the other line-search works well we decide not to dig anymore into this.

		Iterations	Time (s)
L2)	**FW**	943	0.38
	AS	943	0.6
	PW	876	0.62
L0)	**FW**	1865	2.16
	AS	1865	1.76
	PW	1756	1.24

The table presents the performance of the Frank-Wolfe (FW), Away-Step Frank-Wolfe (AS), and Pairwise Frank-Wolfe (PW) algorithms under L2 and L0 regularization terms. The performance is evaluated based on the number of iterations and the time taken to achieve a duality gap threshold of 0.001.

For the L2 regularization, the classic FW algorithm requires 943 iterations and completes in 0.38 seconds, making it the fastest among the three methods. The Away-Step variant also requires 943 iterations but takes longer, with a time of 0.6 seconds. The Pairwise variant, while requiring slightly fewer iterations (876), takes 0.62 seconds. These results indicate that the classic FW algorithm is the most efficient in terms of time when using L2 regularization. The Away-Step variant, despite taking more time, completes in the same number of iterations as the classic FW, suggesting that its additional computational steps do not significantly enhance the solution quality for this regularization. The Pairwise variant, while slower and requiring fewer iterations, still performs reasonably well, indicating its potential utility in scenarios where its specific strengths might be more pronounced.

For the L0 regularization, the performance dynamics change. The classic FW algorithm requires 1865 iterations and takes 2.16 seconds, which is significantly longer than its performance with L2 regularization. The Away-Step variant also requires 1865 iterations but completes more efficiently, with a time of 1.76 seconds. The Pairwise variant, while requiring fewer iterations (1756), is the fastest among the three, taking only 1.24 seconds. These results highlight the differing impacts of regularization on algorithm performance. The classic FW algorithm, while still effective, is outperformed by the Away-Step and Pairwise variants in terms of time efficiency under L0 regularization. The Away-Step variant balances the number of iterations with a reasonable computational time, making it a strong performer. The Pairwise variant, although requiring slightly fewer iterations, achieves this in the shortest time, indicating its suitability for scenarios where speed is critical.

The results underscore the importance of selecting the appropriate algorithm based on the specific regularization term and performance metrics of interest. For L2 regularization, the classic FW algorithm is the most time-efficient while achieving high solution quality. However, for L0 regularization, the Pairwise variant's speed makes it an attractive choice, despite its slightly higher iteration count. The Away-Step variant provides a good balance of speed and solution quality, particularly under L0 regularization making the Pairwise variant is particularly well-suited for applications where rapid convergence is paramount.

2.6 Results: Max Clique Found

In this section, we report the size of the maximum clique found for each dataset, compare it with the best-known solution, and note the initial node used for the search. The results demonstrate varying degrees of success for our algorithm across different datasets. In the Hamming family, our algorithm performed well on the Hamming 8-4 dataset, achieving the best-known solution, but fell short on the Hamming 10-4 dataset. For the Keller family, the algorithm showed modest success, particularly with Keller 4 and Keller 5, though it did not perform as well as the best-known solutions. In the Brock family, the algorithm consistently found smaller cliques compared to the best-known solutions, highlighting the challenging nature of these instances. These results underscore the importance of continuing to refine our algorithm to better handle the complexity and diversity of graph structures presented by the DIMACS benchmark datasets.

Several factors could explain the varying performance of our algorithm across different datasets, particularly in the context of using the Frank-Wolfe algorithm with its classic, away-step, and pairwise variants. The structure and density of the graphs significantly impact the algorithm's efficiency. The Frank-Wolfe algorithm, particularly in its classic form, may struggle with denser graphs where the feasible region's boundaries are highly irregular. This can lead to slow convergence rates, as seen in the Brock and Keller datasets. In contrast, the Hamming 8-4 dataset, with its more regular structure, aligns better with the Frank-Wolfe method's gradient-based optimization, allowing it to achieve the best-known solution.

Another issue could be the specific adaptations of the Frank-Wolfe algorithm used, which have different strengths and weaknesses. The classic Frank-Wolfe method is straightforward but can be inefficient for complex and dense graphs. The away-step variant helps by retracting steps to stay within feasible boundaries more effectively, but might still struggle with highly irregular feasible regions. The pairwise variant aims to balance the classic and away-step methods but might not be as effective for certain graph structures, such as those in the Brock family, which are known for their high density and complexity.

An important osservation is that not all the algorithms arrive at the optimum, infact the Pairwise FW usually does not.

These results emphasize the need for ongoing refinement and development to enhance the Frank-Wolfe algorithm's capability to tackle such a broad spectrum of graph-related challenges, ensuring it can effectively leverage its variants to address the specific nuances of each dataset. Below we can see the results:

Dataset	Max Clique	Best known	Initial Node
Hamming 8-4	16	16	0
Hamming 10-4	32	40	0
Keller 4	8	11	9
Keller 5	16	27	9
Keller 6	32	59	9
Brock 200_2	8	12	17
Brock 200_4	12	17	0
Brock 400_2	17	29	0
Brock 400_4	17	33	0
Brock 800_2	14	24	0
Brock 800_4	15	26	6

3 Conclusion

In conclusion, our exploration of the Frank-Wolfe algorithm and its variants for solving the Maximum Clique Problem (MCP) has provided substantial insights and promising results. The Frank-Wolfe algorithm, known for its projection-free nature and suitability for sparse optimization, has demonstrated its potential when MCP is reformulated as a continuous quadratic programming problem.

Our numerical experiments revealed that while the classic Frank-Wolfe algorithm shows potential, it faces significant limitations when dealing with complex and dense graph structures. The Away-Step and Pairwise variants address some of these challenges by incorporating additional steps to handle boundary solutions more effectively. Notably, the Away-Step variant exhibited consistent improvements in performance, particularly with datasets featuring irregular structures.

The results from various DIMACS benchmark datasets underscore the varying success of our implementations. The Hamming datasets, characterized by regular structures, were more amenable to the Frank-Wolfe approach, whereas the Brock and Keller datasets presented greater challenges due to their density and complexity. These findings highlight the importance of selecting and refining algorithms based on the specific characteristics of the dataset.

Additionally, our experiments with different line search strategies indicated that the "Diminishing Stepsize" method outperformed the "Lipschitz constant dependent stepsize," underscoring the critical role of step size selection in optimizing performance. While the classic Frank-Wolfe algorithm was effective in certain scenarios, it did not perform as well with the "Lipschitz constant dependent stepsize," reaffirming the need for tailored approaches based on the algorithm and problem context.

Future research should focus on several key areas to enhance the effectiveness and efficiency of Frank-Wolfe algorithms in solving the MCP. Investigating additional variants of the Frank-Wolfe algorithm that offer improved convergence rates and robustness, particularly for complex and dense graphs, is crucial. Techniques such as incorporating second-order information or adaptive step sizes could be beneficial. Combining Frank-Wolfe algorithms with other optimization techniques or heuristics, such as integrating metaheuristic approaches like Genetic Algorithms or Simulated Annealing, could improve solution quality and convergence speed. Enhancing the

scalability of the Frank-Wolfe algorithm for very large graphs through parallelization, distributed computing, or efficient data structures is essential. Extending the application of Frank-Wolfe variants to other real-world problems beyond MCP, including domains like machine learning, network analysis, and operations research, could uncover new opportunities and challenges. Furthermore, deeper theoretical investigations into the properties of different Frank-Wolfe variants, particularly in non-convex optimization scenarios, are necessary to inform the design of more effective algorithms. Expanding the benchmarking suite to include more diverse and challenging datasets, ensuring comprehensive testing under varied conditions, and developing evaluation metrics that assess performance beyond convergence rates, such as robustness and computational efficiency, will also be important.