Evaluation of Results of Frank Wolfe Algorithms and Variants with the Maximum Clique Problem

Group 31
Nicola Bazzani, ID: 2163835
Sara Pangrazio, ID: 2165777

June 2025

1 The Problem

The goal of this project is to see how well the Frank Wolfe algorithm and two of its variants (Away-Steps Frank Wolfe and Pairwise Frank Wolfe) perform on the *maximum clique problem (MPC)* introduced in [8]. The problem is defined as:

let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a simple undirected graph with vertex set $\mathcal{V} = \{1, \dots, n\}$ and an edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. A clique in $\mathcal{G}$ is a subset $C \subseteq \mathcal{G}$ such that $(i, j) \in \mathcal{E}$ for every $i, j \in C$ with $i \neq j$. The maximum clique problem consists of finding a clique $C \subseteq \mathcal{E}$ such that $|C|$ is maximum.

The following constraint maximization problem was introduced:

$$\max_{x \in \Delta} x^T A x \tag{1}$$

where Δ is the n-dimensional simplex

$$\Delta = \{x \in \mathbb{R}^n : x \geq 0 \text{ and } e^T x = 1\} \quad e = (1 \dots 1)$$

and $A = (a_{ij})_{i,j \in \mathcal{V}}$ is the adjacency matrix relative to $\mathcal{G}$

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

With the following theorem, it was found that this optimization problem is essentially equivalent to MCP.

Theorem 1.1 (Motzkin and Straus [7])**.** *The optimal objective value of (1) is*

$$1 - \frac{1}{\omega(\mathcal{G})}, \quad \textit{where } \omega(G) \textit{ cardinality of max clique in } \mathcal{G}$$

In general, finding the global maximizer can be impractical and costly. However, since iterative optimization methods typically converge to a point that satisfies first-order optimality, a more effective approach seemed to be to use these algorithms to find approximate solutions of the problem (1). It was found that local maxima could satisfy these condition, and that a characteristic vector for a clique satisfied them if and only if the associated clique was maximal.

The drawback of this approach was the presence of infeasible local maxima of (1) which are not characteristic vectors for cliques and from which a clique cannot be recovered. In [2], to account for this, the author introduced the regularization function $\frac{1}{2}\|x\|_2^2$ in the problem, solving

$$\max_{x \in \Delta} x^T A x + \frac{1}{2}\|x\|_2^2 \tag{2}$$

and showed that, unlike in problem (1), the local maxima of (2) had a one-on-one correspondence with the maximal cliques in $\mathcal{G}$. In fact, the optimal value of 2 with $\alpha = 1/2$ is

$$1 - \frac{1}{2\omega(\mathcal{G})}.$$

For this reason, the problem we are going to solve using the Frank-Wolfe algorithms is this one:

$$\max_{x \in \Delta} x^T A x + \frac{1}{2}\|x\|_2^2 \tag{3}$$

But, since the Frank-Wolfe algorithms work as minimization algorithms, our problem becomes

$$\min_{x \in \Delta} -x^T A x - \frac{1}{2}\|x\|_2^2 \tag{4}$$

Where if $f(x^*)$ is the optimal value of problem (4), $-f(x^*)$ will be the optimal value of the maximization problem.

2 The Frank-Wolfe type algorithms

The Frank-Wolfe algorithms and its variants are constraint optimization algorithms and therefore have the following structure, as illustrated in [1]. We consider $F(x, g)$ a set of directions feasible at x using first-order information of f around x. It's important for α_k^{max} to be computed at each iteration, to maintain convexity.

Algorithm 1 Example Algorithm: first order method [1]

1: Choose a point $x_0 \in C$
2: **for** k = 0, ... **do**
3: If x_k satisfies some specific condition, then STOP
4: Choose $d_k \in F(x_k, \nabla f(x_k)$
5: set $x_{k+1} = x_k + \alpha_k d_k$, with $\alpha_k \in (0, \alpha_k^{max}]$ with a suitably chosen stepsize
6: **end for**
7: **return** x

2.1 Frank-Wolfe

Algorithm 2 Frank-Wolfe [1]

1: Choose a point $x_0 \in \Delta$
2: **for** k = 0, ... **do**
3: If x_k satisfies some condition, then STOP
4: Set $s_k = \arg\min_{x \in \Delta} \nabla f(x_k)^T x$
5: Set $d_k^{FW} = s_k - x_k$
6: set $x_{k+1} = x_k + \alpha_k d_k^{FW}$, with $\alpha_k \in (0, 1]$ with a suitably chosen stepsize.
7: **end for**
8: **return** x

To implement this algorithm and solve the problem

$$\min_{x \in \Delta} f(x)$$

we assume that the function f is convex and continuously differentiable. Since our domain is the simplex, we know it's a compact convex subset of the vector space $\mathbb{R}^n$.

At each iteration, the algorithm moves toward an extreme point of the feasible set by minimizing the scalar product with the current gradient. To solve this subproblem, it makes use of a *linear minimization oracle* LMO for the feasible set (step 3).

$$LMO_C(y) = \arg\min_{x \in C} y^T x$$

Computing the LMO can be as complicated as solving the original problem, but, since the polytope we're optimizing over is Δ, we know that, as shown in [3], for

$$\Delta = \{x \in \mathbb{R}^n : x \geq 0 \text{ and } e^T x = 1\} = \text{conv}(\{e_i, i = 1, \ldots, n\}),$$

the solution s_k is

$$s_k = LMO_\Delta(\nabla f(x_k)) = e_{i_k}, \quad i_k = \arg\min_i \nabla_i f(x_k)$$

In this case, $\{e_1, \ldots, e_n\} = \mathcal{S}$ are called atoms.

As for the stopping criterion, we first need to introduce the duality gap [5], a fundamental concept in constraint optimization.

For a constraint optimization problem of the form (4) and a point $x \in C$ we define the duality gap as

$$gap(x) = \max_{s \in C} \langle \nabla f(x), x - s \rangle$$

For the convexity of f, we know that $f(x) + \langle \nabla f(x), x - s \rangle$ always lies below the graph on the function. For this reason, we can consider the duality gap a certificate of the quality of the current approximation, since $gap(x) \geq f(x) - f(x^*)$. With this property, in practice, we implement the duality gap as a stopping criterion, since, given a tolerance ϵ, we know that

$$gap(x) \leq \epsilon \implies f(x) - f(x^*) \leq gap(x) \leq \epsilon$$

without knowing the solution x^*, we'll know that the resulting iterate will have a small error.

2.2 Away-Steps Frank-Wolfe

Algorithm 3 Away-steps Frank-Wolfe algorithm

1: Let $x_0 \in \Delta$, initialize $\mathcal{S}^{(0)}$ and $\alpha_v^{(0)}$
2: **for** $k = 0, \ldots$ **do**
3: Let $s_k^{FW} := \text{LMO}_C(\nabla f(x_k))$ and $d_k^{\text{FW}} := s_k^{FW} - x_k$
4: Let $s_k^{AS} \in \arg\max_{v \in S^{(k)}} \langle \nabla f(x_k), v \rangle$ and $d_k^{\text{AS}} := x_k - s_k^{AS}$
5: **If** s_k^{FW} satisfies some condition, then **STOP**
6: **if** $\langle -\nabla f(x_k), d_k^{\text{FW}} \rangle \geq \langle -\nabla f(x_k), d_k^{\text{AS}} \rangle$ **then**
7: $d_k := d_k^{\text{FW}}$, $\gamma_{\max} := 1$
8: **else**
9: $d_k := d_k^{\text{AS}}$, $\gamma_{\max} := \frac{\alpha_{s_k^{AS}}}{1 - \alpha_{s_k^{AS}}}$
10: **end if**
11: Line-search for a feasible stepsize $\gamma_k \in [0, \gamma_{max}]$
12: Update $x_{k+1} := x_k + \gamma_k d_k$
13: Update $S^{(k+1)}, \alpha_v^{(k+1)}$
14: **end for**

To implement these two variants, we followed the procedure shown in [6].

The Frank-Wolfe Away Steps variant was introduced to deal with a problem that occurred when the solution x^* lied on the boundary of the polytope we're optimizing against. In this case, the original algorithm can take longer as it start to zig-zag on the face of the polytope the solution lies on. This made the algorithm sublinear.

At iteration k, the solution is a convex combination of at most $k+1$ atoms, so, $x = \sum_{v \in \mathcal{S}^{(k)}} \alpha_v^{(k)} v$. We can call $\mathcal{S}^{(k)}$ the active set of atoms for x_k.

To account for the zig-zagging problem, another direction is introduced. The *away direction* d_k^{AS}, introduced to move *away* from the active atom, is defined by finding the atom v_k in $\mathcal{S}_k$ that maximizes the potential of descent $g_k^A = \langle -\nabla f(x^{(k)}), d_k^{\text{AS}} \rangle$. Since this search is done over the (finite and relatively small) set $\mathcal{S}^{(k)}$, it's much easier to implement then the LMO.

Since the duality gap $\langle -\nabla f(x^{(k)}), d_k \rangle$ is an upper bound of the solution's error, we can choose which one of the two direction brings us closer to convergence: if

$$\langle -\nabla f(x_k), d_k^{\text{FW}} \rangle \geq \langle -\nabla f(x_k), d_k^{\text{AS}} \rangle,$$

we choose d_k^{FW}.

The updates in line 13 are of the form

- For a FW step: if $\gamma_k = 1$, $\mathcal{S}^{(k+1)} = \{s_k^{FW}\}$, otherwise, $\mathcal{S}^{(k+1)} = \mathcal{S}^{(k)} \cup \{s_k^{FW}\}$

  Also, we have

$$\alpha_{s_k^{FW}}^{(k+1)} := (1-\gamma_k)\alpha_{s_k^{FW}}^{(k)} + \gamma_k \quad \text{and} \quad \alpha_v^{(k+1)} := (1-\gamma_k)\alpha_v^{(k)} \text{ for } v \in S^{(k)} \backslash \{s_k^{FW}\}.$$

- For a AS step: we have $S^{(k+1)} = S^{(k)} \backslash \{s_k^{AS}\}$ if $\gamma_k = \gamma_{\max}$ (a *drop step*); otherwise $S^{(k+1)} = S^{(k)}$.

  Also, we have

$$\alpha_{s_k^{AS}}^{(k+1)} = (1+\gamma_k)\alpha_{s_k^{AS}}^{(k)} - \gamma_k \quad \text{and} \quad \alpha_v^{(k+1)} = (1+\gamma_k)\alpha_v^{(k)} \text{ for } v \in S^{(k)} \backslash \{s_k^{AS}\}.$$

2.3 Pairwise Frank-Wolfe

Algorithm 4 Pairwise Frank-Wolfe algorithm

1: Let $x_0 \in \Delta$, initialize $\mathcal{S}^{(0)}$ and $\alpha_v^{(0)}$
2: **for** $k = 0, \ldots$ **do**
3: Let $s_k^{FW} := \text{LMO}_C(\nabla f(x_k))$ and $d_k^{\text{FW}} := s_k^{FW} - x_k$
4: Let $s_k^{AS} \in \arg\max_{v \in S^{(k)}} \langle \nabla f(x_k), v \rangle$ and $d_k^{\text{AS}} := x_k - s_k^{AS}$
5: **If** s_k^{FW} satisfies some condition, then **STOP**.
6: Let $d_k^{PFW} = s_k^{FW} - s_k^{AS}$, and $\gamma_{max} = \alpha_{v_k}$
7: Line-search for a feasible stepsize $\gamma_k \in [0, \gamma_{max}]$
8: Update $x_{k+1} := x_k + \gamma_k d_k$
9: Update $S^{(k+1)}, \alpha_v^{(k+1)}$
10: **end for**

For this algorithm, the idea is to move weight away from the atom s_k^{AS} to the FW atom s_k^{FW}. This is equivalent to choosing the search direction $d_k^{PFW} = d_k^{\text{AS}} + d_k^{\text{FW}} = s_k^{FW} - s_k^{AS}$. For this reason, the only weights that are updated are

$$\alpha_{s_k^{FW}}^{(k+1)} = \alpha_{s_k^{FW}}^{(k)} - \gamma \quad \text{and} \quad \alpha_{s_k^{AS}}^{(k+1)} = \alpha_{s_k^{AS}}^{(k)} + \gamma$$

for a stepsize $\gamma \leq \gamma_{max} = \alpha_{v_k}^{(k)}$. When $\gamma = \gamma_{max}$ and $|S^{(k+1)}| = |S^{(k)}|$, we call this a *swap step*.

At each iteration, the FW atom is added to the active atom set, and eventually, to maintain the convex property, the AS atom is removed.

2.4 Key Concepts and Convergence Rates of the Frank Wolfe Algorithms

2.4.1 Frank Wolfe Algorithm

We now present the results of the convergence analysis of the Frank-Wolfe classic algorithm seen in [5].

Since, as we saw, the duality gap is a certificate of the quality of the solution, in constraint optimization we consider **primal-dual convergence**.

It's important to introduce a second element that the convergence analysis of the Frank Wolfe algorithm rely on: the curvature constant C_f of the function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$C_f = \sup_{x,s\in\mathcal{D}} \frac{2}{\gamma^2}(f(y) - f(x) - \langle y - x, \nabla f(x)\rangle), \quad \text{where} \quad \gamma \in [0,1], \quad y = x + \gamma(s - x)$$

The curvature constant of a function defines the deviation f can have from any point.

The following theorem shows that, after $O(1/\epsilon)$ iterations, the iterate x_k satisfies

$$f(x_k) \leq f(x^*) + \epsilon$$

Theorem 2.1 (Primal Convergence)**.** *For each $k \geq 1$, the iterates x_k of the Frank Wolfe algorithm satisfy*

$$f(x_k) - f(x^*) \leq \frac{2C_f}{k+2},$$

where $x^* \in C$ is an optimal solution to the minimization problem.

As for the dual convergence, **Theorem 2.2** states that the algorithm we saw also reaches a small duality gap, $g(x_k) \leq \epsilon$ after $O(1/\epsilon)$ iterations.

Theorem 2.2 (Dual Convergence)**.** *If Algorithm 2 runs for $K \geq 2$ iterations, then it has an iterate $x_{\hat{k}}$, $\quad 1 \leq \hat{k} \leq K$, with duality gap bounded by*

$$g(x_{\hat{k}}) \leq \frac{2\beta C_f}{K+2}$$

where $\beta = 27/8$.

2.4.2 Frank Wolfe Variants

We have different theorems [6] for the convergence of the Frank Wolfe variants, Away Steps and Pairwise (even though they are still valid for the classic algorithm).

These theorem depend also on the *pyramidal width* of the optimizing set (in our case, the simplex).

To generalize, we consider the optimizing set C and set $\mathcal{A}$ s.t. $C = \text{conv}(\mathcal{A})$. Some preliminar definitions are needed:

- A direction r is *feasible* for $\mathcal{A}$ from x if it points inwards $\text{conv}(\mathcal{A})$. FOr this, we write $r \in \text{cone}(\mathcal{A} - x)$
- The *Directional Width* of a set $\mathcal{A}$ with respect to direction r is the minimum directional width over all possible direction in its affine hull and is defined as

$$dirW(\mathcal{A}, r, x) = \max_{s,v\in\mathcal{A}} \langle \frac{r}{\|r\|}, s - v\rangle$$

- The *Pyramidal Direction Width* of a set $\mathcal{A}$ with respect to direction r and a base point $x \in C$ is defined as

$$PdirW(\mathcal{A}, r, x) = \min_{\mathcal{S}\in\mathcal{S}_x} dirW(\mathcal{S} \cup \{s(\mathcal{A}, r), r\}) = \min_{\mathcal{S}\in\mathcal{S}_x} \max_{s,v\in\mathcal{A}} \langle \frac{r}{\|r\|}, s - v\rangle$$

  where $\mathcal{S}_x = \{\mathcal{S} | \mathcal{S} \subseteq \mathcal{A}$ s.t. x is a proper convex combination of all the elements of $\mathcal{S}\}$ and $s(\mathcal{A}, r) = LMO_{\mathcal{A}}(r)$ is the FW atom.

- To define the Pyramidal Width of a set we consider the minimum over the cone of feasible direction r. It's defined as the smallest pyramidal directional width of all its faces

$$PWidth(\mathcal{A}) = \min_{\substack{K\in\text{faces}(\text{conv}(A)) \\ x\in K \\ r\in\text{cone}(K-x)\setminus\{0\}}} PdirW(\mathcal{K} \cap \mathcal{A}, r, x)$$

We can now introduce the primal and dual convergence theorems.

Theorem 2.3. *Suppose that f has L-Lipschitz gradient and is μ-strongly convex over $C = conv(\mathcal{A})$. Let $M = diam(C)$ and $\delta = PWidth(\mathcal{A})$. Then the suboptimality h_k of the iterates of all the variants of the FW algorithm decreases geometrically at each step that is not a drop step nor a swap step, that is*

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

If we define $c(k)$ as the number of "good" steps (non drop and swap steps), then we have $c(k) \geq k/2$ for Away Steps and $c(k) \geq k/(3|\mathcal{A}|! + 1)$ for Pairwise. This yields a global linear convergence rate for all variants of

$$h_k \leq h_0 \exp(-\rho c(k)), \quad \text{if } \mu = 0, \text{ then } h_k = O\left(\frac{1}{c(k)}\right)$$

Theorem 2.4. *Suppose that f has L-Lipschitz gradient over C with $M := diam(C)$. Then the gap gap_k^{FW} for all the algorithms is upper bounded by primal error h_k as follows*

$$gap_k^{FW} \leq \begin{cases} h_k + LM^2/2 & \text{if } h_k > LM^2/2 \\ M\sqrt{2h_k L} & \text{otherwise} \end{cases}$$

3 Implementation of the algorithms

We started by defining the function and its gradient,

$$f : \mathbb{R}^n \to \mathbb{R}, \quad f(x) = -x^T Ax - \frac{1}{2}x^T x = -x^T(A + \frac{1}{2}I)x \tag{5}$$

is a convex function with Lipschitz continuous gradient having constant $L > 0$.

$$\nabla f(x) = g(x) = -2(A + \frac{1}{2}I)x = -(2A + I)x \tag{6}$$

3.1 Frank-Wolfe Algorithm

We defined this function with the following inputs

```
def Frank_wolfe(f,g, A, x0,LMO, checkpoints, max_it, eps = 1e-2)
```

Listing 1:

Where `f` is our loss function and `g` its gradient, `A` is the graph adjacency matrix of dimension $n \times n$ and `x0` the initial vector of dimension n. The list `checkpoints` contains the iterations where the algorithm saves the solution and optimal value, then we have the the maximum number of iterations and the tolerance ϵ needed for the stopping criterion.

The outputs are the same for all the algorithms and are

```
    return x, it, fx, checkpoint_fx, ttot
```

Listing 2:

Here, `x` is the solution vector of dimension $n \times n$, `it` the number of iterations that the algorithm used, `fx` the optimal value, `checkpoint_fx` is a dictionary containing as keys the `checkpoints` list, and as values the saved solutions and `ttot` the total time the algorithm used.

So, we computed the LMO as an external function and calculated the value of the FW direction .

```
def LMO_simplex(g,x_k):
  # D = {x in R^n | x_i >= 0, e.T @ x = 1}
  n = len(A)
  e = np.zeros(n)
  i = np.argmin(g(x_k))
```

```
  e[i] = 1
  return e
```

Listing 3: LMO

As for the stopping criterion, we implemented the duality gap.

```
def duality_gap(x, gradient, s):
  return np.dot(gradient, x - s)
```

Listing 4:

```
gap = duality_gap(x, g_k, s_FW)
if gap < eps:
    stop_cr = 0
    break
```

Listing 5:

For the stepsize, we used the classic diminishing step size $\gamma_k = \frac{2}{k+1}$

3.2 Away Steps Frank-Wolfe

The outputs are the same as the Frank-Wolfe algorithm, but for the inputs we added the option to choose the method of line search `arls = 1,2,3`.

We computed the initial atom set $\mathcal{S}^{(0)}$, we defined the active indices and relative weights with

```
active_indices = np.where(x0 > 1e-10)[0]
active_weights = x0[active_indices]
S = [np.eye(n)[idx] for idx in active_indices]
```

Listing 6:

We implemented the new Away-Step direction $s_k^{AS} = \arg\max_{v \in S^{(k)}} \langle \nabla f(x_k), v \rangle$

```
 active_gradients = g_k[active_indices]
 internal_away_idx = np.argmax(active_gradients)
 away_vertex_original_idx active_indices[internal_away_idx]
 s_AS = S[internal_away_idx]
```

Listing 7:

And we chose the best direction and implemented the different γ_{max} values

```
g_FW = -g_k @ d
d_AS = x - s_AS_FW
g_AS = -g_k @ d_AS

# Choose direction
if g_FW >= g_AS:
    direction = d_FW
    gamma_max = 1.0
    step_type = 'FW'
else:
    direction = d_AS
    gamma_max = active_weights[internal_away_idx] / (1 - active_weights[
        internal_away_idx] + 1e-12)
    step_type = 'AS'
```

Listing 8:

For the line-search, we tried the methods Exact Line search, Armijo Rule and Diminishing Stepsize to see which one performed best.

- **Exact Line Search:** in this case, the stepsize γ_k is the value obtained by minimizing f along d_k, so

$$\gamma_k^* = \arg \min_{\gamma \in [0, \gamma_{max}]} f(x_k + \gamma d_k)$$

We opted for a grid-exact line search: we considered a number of possible values $\gamma \leq \gamma_{max}$ and chose the value that minimized the function $f(x_k + \gamma d_k)$.

```
def exact_line_search(f, x, d, A, gamma_max, steps=20):
  best_gamma = 0.0
  best_value = f(x, A)
  for i in range(1, steps + 1):
      gamma = gamma_max * i / steps
      fx = f(x + gamma*d, A)

      if fx < best_value:
          best_value = fx
          best_gamma = gamma

  return best_gamma
```

Listing 9:

- **Armijo Rule** This case only tries to decrease the objective function. We fix parameters $\delta \in (0, 1)$, $\eta \in (0, 1/2)$ and starting stepsize γ_0. We then update $\gamma = \delta^m \gamma_0$ until the inequality

$$f(x_k + \gamma d_k) \leq f(x_k) + \eta \gamma \nabla f(x_k)^T d_k$$

is satisfied

```
delta = 0.5
eta = 1e-4
gamma = min(1,alpha_max)
while f(x + gamma * d_k, A) > f_k + eta * gamma * g_k.T @ d_k:
 gamma *= delta

x = x + gamma * d_k
```

Listing 10:

- **Diminishing Stepsize.** For this option we used the same value as the Frank-Wolfe method, $\gamma_k = \frac{2}{k+1}$.

Finally, we updated the atom set $\mathcal{S}$ and the weights with the procedure shown in **section 2.2**

3.3 Pairwise Frank-Wolfe

The inputs and outputs of this algorithms are the same as the Frank Wolfe Away-Steps. We calculated the new Pairwise direction as $d_k^{PFW} = s_k^{FW} - s_k^{AS}$ and proceeded with line search. Lastly, we updated the atom set and the weights.

4 Experiments on Datasets

To test the algorithms, we considered three different datasets from three families of maximum cliques instances belonging to the DIMACS benchmark [4].

- From the p_hat family (a set of random graph generated with the p-hat generator, which is a generalization of the classical uniform random graph generator) we used the instance **p_hat300-3**, a graph with 300 nodes, 33390 edged and maximum clique size 36.
- From the C family (a set of random graphs $C_{x.y}$, where x is the number of nodes and y the edge probability), we used the **C125.9** instance, with 125 nodes, 6963 edges and maximum clique size 34.
- From the Hamming family (a hamminga-b is a of graph on a-bit words with an edge if and only if the two words are at least hamming distance b apart), we chose the **hamming8-4** instance, with 256 nodes, 20864 edges and maximum clique size 16.

We uploaded the files in .clq format and extracted the adjacency matrices from them (respectively naming them P,C,H), then, using the package `networkx`, we plotted the relative graphs.

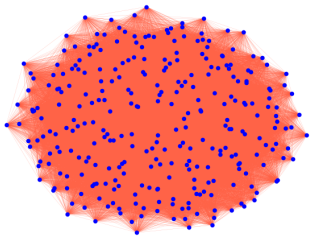

Figure 1: Dataset p_hat300-3

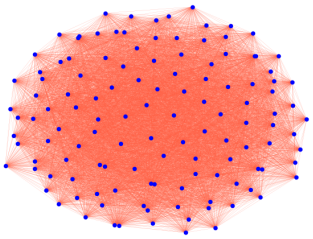

Figure 2: Dataset C125.9

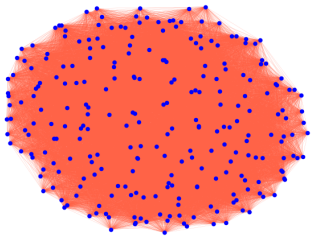

Figure 3: Dataset hamming8-4

Figure 4: Graphs of the instances we used

4.1 Testing the algorithms

When the algorithm is given as input our loss function f and an initial vector we defined on the simplex, the solution vector `x` should look something like

```
[0.    0.    0.03125    0.     0.     0.     0.     0.03125  0.      0.      0.
 ..... 0.      0.03125    0.     0.03125    0.     0.     0.       0.
       0.      0.03125 ]
```

Listing 11:

So, a sparse vector made of, for the most part, values not significantly different from 0, and some values greater than 0. The indices of these values (the support of x) are the nodes present in the maximum clique that the algorithm found. Moreover, to verify whether or not the algorithm actually found the largest clique, we calculated its cardinality using the equation

$$f(x) = 1 - \frac{1}{2\omega} \implies \omega = \frac{1}{2(1 - f(x))}$$

The first part of the following analysis was made using for all the algorithms the line-search that worked best, which is Armijo Line Search. In the second part, we'll see some comparisons between the line-search methods.

- First, we tested on every dataset if the algorithms converged to the right solution.

  The function `clique_accuracy` takes as input the algorithm we want to test and shows whether or not the cardinality of the clique found by the algorithm converges to the right size, calculating the errors for each iteration in `checkpoints`. The errors are calculated as the vector differences between the true cardinality and the cardinality of the clique found by the algorithm at each iteration.

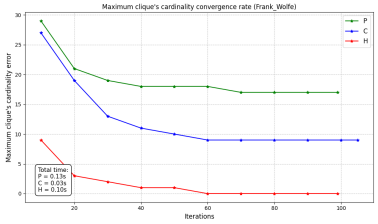

Figure 5: Frank Wolfe

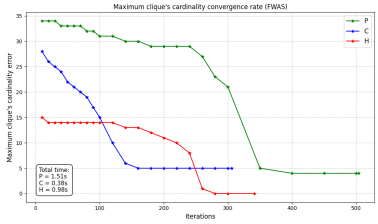

Figure 6: Frank Wolfe Away Steps

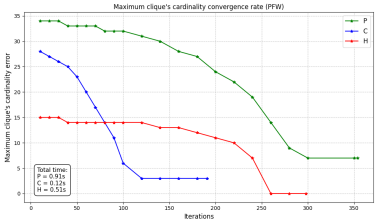

Figure 7: Pairwise Frank Wolfe

Table 1: Clique Size Estimation Performance

Dataset	**FW**	**FWAS**	**PFW**	**Optimal**
p_hat300-3	19 (17)	32 (4)	29 (7)	36
C125.9	25 (9)	29 (5)	31 (3)	34
hamming8-4	16 (0)	16 (0)	16 (0)	16

Parentheses show absolute error vs optimal

For the Frank Wolfe algorithm, we can see that after a few iterations (approximately 40), the results stabilize, and, even though the dataset H produces good results, the other two present high errors. The algorithm, however stops just after 100 iterations.

The Away Steps algorithm works much better, calculating the cardinality of the datasets P and C with a more acceptable error. Here, the algorithm stops after 300 iterations for datasets C and H but reaches the 500 iterations for dataset P

The Pairwise Frank-Wolfe algorithm also works well and shows similar results to the Away Steps algorithm, but uses overall less iterations to do so.

For all the algorithms, the CPU time is negligible.

- We then decided to compare how each algorithm worked for every dataset, using the function `compare_algorithms_on_dataset`.

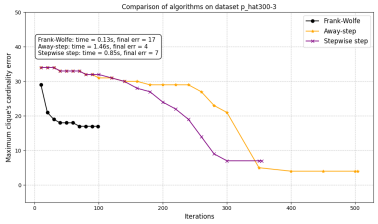

Figure 8: Dataset p_hat300-3

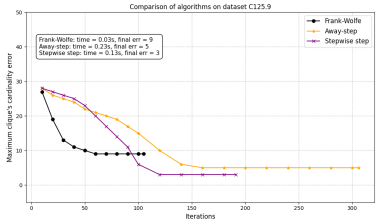

Figure 9: Dataset C125.9

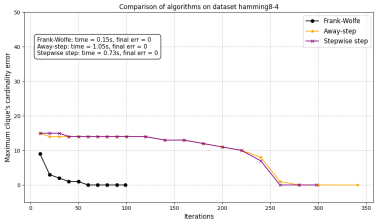

Figure 10: Dataset Hamming8-4

As expected, dataset Hamming8-4 is the one that fits the algorithms best, having 0 error. Dataset C125.9 also works well for the Away Steps and Pairwise algorithm, but doesn't for the Frank Wolfe algorithm. The p_hat300-3 dataset is the one that presents more errors, with a very bad result from the Frank Wolfe algorithm and two acceptable results with the other two.

- We decided to verify that the subset of nodes found by the algorithm was actually a clique C, so if its nodes satisfied the property that $(i,j) \in \mathcal{E}, \forall i,j \in C$ with $i \neq j$. We know that an edge $(i,j) \in \mathcal{E}$ if and only if `A[i,j] = 1`. We used the function `is_clique` and tested it on every algorithm. It's implemented as:

  – Considers the solution vector x^* returned by the algorithm

- Sets a threshold (0) and counts which indices of the vector are greater than the threshold. Those indices represent the nodes belonging to the clique.
- Verifies that the nodes satisfy the property.
- If the subset is not a clique, it returns which edges are missing to define it as one.

As for the results, we obtained that all the algorithms found an actual clique in each dataset.

Finally, we summarized the results we got with two tables.

As we can see, the more effective line search method is, in this case, Armijo Rule. Exact Line Serach also works well, while Diminishing Stepsize, even though it's a good method for the classic FW algorithm, doesn't work well at all for the Away Steps algorithm and also shows medium-bad results for the Pairwise algorithm.

Exact Line Search is also the method that, in some instances, takes the most CPU time (see (Table3b) with the p_hat300-3 dataset and (Table3c) for p_hat300-3 and hamming8-4).

Table 2: Clique Sizes Comparison under Different Step Size Strategies

(a) Exact Line Search

Dataset	Max Clique (ω)	FW Clique	FWAS Clique	PFW Clique
p_hat300-3	36	19	29	30
C125.9	34	25	25	31
hamming8-4	16	16	16	16

(b) Armijo Rule

Dataset	Max Clique (ω)	FW Clique	FWAS Clique	PFW Clique
p_hat300-3	36	19	32	29
C125.9	34	25	29	31
hamming8-4	16	16	16	16

(c) Diminishing Step Size

Dataset	Max Clique (ω)	FW Clique	FWAS Clique	PFW Clique
p_hat300-3	36	19	2	29
C125.9	34	25	5	28
hamming8-4	16	16	1	13

Table 3: Summary of Algorithm Performance

(a) Frank-Wolfe (FW) Summary

Dataset	Clique Error	Time (s)	Iterations	$f(x)$
p_hat300-3	17	0.148	99	-0.973
C125.9	9	0.025	105	-0.980
hamming8-4	0	0.097	99	-0.968

(b) Away-Step Frank-Wolfe (FWAS) Summary

Dataset	Line Search	Clique Error	Time (s)	Iterations	$f(x)$
p_hat300-3	exact	7	7.184	800	-0.983
	armijo	4	1.677	504	-0.984
	diminishing	34	1.233	800	-0.742
C125.9	exact	9	1.145	800	-0.980
	armijo	5	0.333	306	-0.983
	diminishing	29	0.249	800	-0.896
hamming8-4	exact	0	4.392	800	-0.968
	armijo	0	0.974	341	-0.969
	diminishing	15	0.891	800	-0.637

(c) Pairwise Frank-Wolfe (PFW) Summary

Dataset	Line Search	Clique Error	Time (s)	Iterations	$f(x)$
p_hat300-3	exact	6	6.990	800	-0.983
	armijo	7	0.826	354	-0.983
	diminishing	7	1.185	800	-0.983
C125.9	exact	3	1.151	800	-0.984
	armijo	3	0.106	191	-0.984
	diminishing	6	0.239	800	-0.982
hamming8-4	exact	0	4.321	800	-0.969
	armijo	0	0.519	298	-0.969
	diminishing	3	1.118	800	-0.962

References

[1] I. M. Bomze, F. Rinaldi, and D. Zeffiro. *Frank–Wolfe and friends: a journey into projection-free first-order optimization methods*, 4OR 19 (2021).

[2] I.M. Bomze. *Evolution towards the Maximum Clique*, Journal of Global Optimization 10 (1997), pp. 143–164.

[3] *Complexity of linear minimization and projection on some sets*, Operations Research Letters 49.4 (2021), pp. 565–571. ISSN: 0167-6377.

[4] *DIMACS Implementation Challenges - Cliques, Coloring, and Satisfiability*. `https://www.dcs.gla.ac.uk/~pat/jchoco/clique/dimacs/DIMACS_cliques/`. Accessed: June 2025.

[5] Martin Jaggi. *Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization*. Vol. 28. 1. PMLR, 2013, pp. 427–435.

[6] Simon Lacoste-Julien and Martin Jaggi. "On the global linear convergence of Frank-Wolfe optimization variants", Cambridge, MA, USA: MIT Press, 2015.

[7] T.S. Motzkin and E.G. Straus. *Maxima for graphs and a new proof of a theorem of Turàn.*, Canadian Journal of Mathematics 17 (1965), pp. 533–540.

[8] J. T. Hungerford and F. Rinaldi. *A General Regularized Continuous Formulation for the Maximum Clique Problem*, Mathematics of Operations Research 44 (2019), pp. 1161–1173.