Approximation algorithm approaches for the Maximum Cut Problem

Bachelor Thesis
presented on 12 April 2020
to
Centre de Mathématiques Appliquées
supervised École Polytechnique Paris

in the context of a Bachelor degree
by

CHAN Chun Hei Michael

Palaiseau, France, 2020

Abstract

This study investigates the different applications of algorithms including approximation algorithms on Maximum Cut problem. Throughout the paper, a thorough review of the existing methods is made. From methods to solve Semi-Definite Programs to probabilistic and deterministic methods of rounding to round solutions of Semi-Definite Programs to MaximumCut problem solutions. Actual implementations of methods in Python is done to carry out performance and time comparisons. On top of comparisons, multiple methods of derandomization algorithms are shown alongside its proofs. In the original paper written by Michel Goemans and David Williamson [8], a potential method of derandomization was depicted but then proved wrong by Sanjeev Mahajan and Hariharan Rames in [13] that is again why we write in our paper multiple methods of derandomization of formulate a more complete set of solving methods to Maximum Cut. One of the goal of the paper was to compare a derandomized rounding to the very well-known Goemans-Williamson 0.878 rounding algorithm: a comparison of performance for a similarly allocated time. We pursue in achieving a high approximation of MaxCut of graphs and manage to reach using derandomization rounding an empirical value of 0.991 of the optimum values for two sets of graphs.

Key words: Convex Optimization, Semi-Definite Programming, Maximum Cut, Derandomization, Rounding

Contents

1 Introduction

Optimizing, More and More: Throughout ages, combinatorics problem gradually got more and more important. Solving such a problem can however become very much difficult. Problems not being solvable in a reasonable time brings upon the idea of a trade-off between approaching enough the optimal solution and not spending overly time. These methods are Approximation Algorithms and our goal in this paper is to present these methods and their performances in different cases. We will be working on two general problems that we define later, Maximum Cut problem
Research has been done on such topics, some related works are the following:

1. Linear Programming relaxations of maxcut [6]
2. Derandomizing Semi-Definite Based Approximation Algorithms [13]

Notations: Here we describe the notations used throughout the paper:

1. $M_{m\times n}$: are matrices of size $m \times n$
2. $C \bullet X = Tr(CX) = |CX|_{Frobenius}$
3. S^n: are symmetric matrices of size $n \times n$
4. $v, M \geq 0$: for a vector v it means $v_i \geq 0$, and for a matrix M it means positive semi-definite.
5. $E_{i,j}$: matrix with 1 at (i, j) and 0 else, $\mathbb{1}$: vector of 1s.
6. L^n: lower triangle matrix of size $n \times n$

2 Approximation Algorithms

Let us introduce the topic through a simple problem.

Problem 1. *"Mike has a backpack that can handle 20 kg. He plans on going to holidays with his friends and wants to bring some snacks along respectively Pumpkin (1.5kg), Nasi Goreng (1kg) Fish Cake (350g). Mike values one Pumpkin at 10, one Nasi Goreng at 6.5, and one Fish Cake at 2.35 (unit: Utility points). How does Mike maximizes its utility?"*

This is a version of the Knapsack Problem [14] that can be effectively solved by discrete optimization methods. Similar problems with more variables can be much more interesting and worth optimizing but unlike the previous question that is rather simple to solve, some require more techniques and happen to be NP-hard, [10] i.e does not possess any exact efficient algorithm that is an algorithm that runs in polynomial time and manage to reach the optimum. These very much interesting problem, being very dear to us, should however still be dealt with. And so appeared a new family of algorithm: **approximation algorithm**. In this part for approximation algorithms, we will be mentioning a couple of algorithms that will be used throughout the paper, bear in mind that it is not an exhaustive description of approximation algorithms.

2.1 Definitions and First Notions

In this short part, we will be giving a definition to approximation algorithms. This definition will be reused multiple times in the paper.

Definition 2.1.1. *An* α-approximation algorithm *is an algorithm that outputs in polynomial time [1], for each case of the same optimization problem, a solution whose value is at most at a* $\alpha-$*factor away from the optimal solution.*

In the situation of a maximization problem with optimal value OPT, it would be for the solution to have a value V with $V \geq \alpha OPT$. And in the situation of a minimization problem to have a value V with $V \leq \alpha OPT$.

2.2 Main Tools and Ideas for Algorithms

In order to solve a combinatorics problem such as Problem 1, multiple solutions can be thought of. We could try some rather naive ideas such as Enumeration for instance, to then be more sophisticate and rely on other listed tools in this part.

2.2.1 Enumeration

An enumeration algorithm that is used to solve a combinatorics problem implies to list all feasible values and then evaluating these values to then pick the optimal one. Despite it always reaching the optimal value when it converges, it does not always converge especially when the number of possible values are exponential. Here is an application of enumeration method on Problem 1. We remind that the problem is equivalent to solving the following maximization problem:

$$\max_{(x,y,z)} 10x + 6.5y + 2.35z$$

$$\text{under the constraint} \quad 1.5x + y + 0.35z \leq 20$$

with $x, y, z \in \mathbb{N}$
Now enumerating all possibilities that are satisfying this constraint gives us 3129 possibilities to pick from. From all these values the highest happens to be 134, that is realized when $x, y, z = (1, 1, 50)$. Again through this example, we see that this method indeed gives us the optimal value but at the price of very demanding computation cost.

2.2.2 Greedy

A greedy algorithms entitles a short-sighted choice making. Given an algorithm for a problem, each iterative step of this specific algorithm will try to pick the maximizing value, accounting only for the current step. While greedy algorithms assure a polynomial-time and a convergence, it does not imply reaching an optimum. Once more we will be using Problem 1 as an example to illustrate this method. We remind that the problem is equivalent to solving the following maximization problem:

$$\max_{(x,y,z)} 10x + 6.5y + 2.35z$$

$$\text{under the constraint} \quad 1.5x + y + 0.35z \leq 20$$

with $x, y, z \in \mathbb{N}$
The way we proceed is the compute the unitary (weight unit) value of each items. And therefore, for a Pumpkin (x) the value is around 6.7 per kg, whereas for a Nasi Goreng (y) the value is 6.5 for 1kg and finally for a Fish Cake (z) it is around 6.71 per kg. Now greedy would then rank these items by these values, so to spend its quota first on the best ratio, until constraints limits it then goes on to the second best ratio. In this specific case, this method yield the 57 Fish Cakes that sums up to 19.95 kg and gives 133.95 unit of benefit. Even though in this case the

optimum was just 0.05 short, we see intuitively that it is possible that this method would waste even more free weights, that is detrimental to reach optimum.

2.2.3 Linear Programming

Despite the previous methods: Enumeration and Greedy being nice to implement. It is known that these perform poorly most of times. Since enumeration, as shown, finds a solution in exponential time while a Greedy algorithm does not necessarily converge toward an optimum. That is why we are required to find another solution. In case Problem 1 the problem can be rephrased as an Integer Program that we define below.

Definition 2.2.1. *Integer Programming (IP) is a family of method to solve a problem with convex and integer constraints, and a linear objective function. Integer programs can be expressed intuitively the following way:*

$$\max c^T x$$

$$\textit{with } Ax \leq b \textit{ and } x \geq 0 \ , x \in \mathbb{Z}^n$$

While our problem is an Integer Program, a general way to solve it is to first apply a relaxation i.e allowing for a bigger set of solution which optimum is at least greater or equal than the optimum value in the original problem. We then solve the specific relaxation using Linear Programming [9] defined as follows.

Definition 2.2.2. *Linear programs (LP) are problems with a linear objective function with feasible space being a polyhedron. Linear programs can be expressed intuitively the following way:*

$$\max c^T x \mid c, x \in \mathbb{R}^n$$

$$\textit{with } Ax \leq b \textit{ and } \mid A \in M_{m \times n}, x, b \in \mathbb{R}^n$$

The constraints are also often written in the following way, called standard form:

$$\max c^T x \mid c, x \in \mathbb{R}^n$$

$$\text{with } Ax + s = b \text{ and } x, s \geq 0 \mid A \in M_{m \times n}, x, s, b \in \mathbb{R}^n$$

Now in terms of methods to solve a Linear Programs, there are a few that we briefly go over as follows:

1. **Simplex Method** [4]: The simplex method was first introduced by Dantzig in 1955. Consider the standard expression of LP. As we previously stated, the feasible solution set is a bounded polyhedron, and therefore optimizing along a direction forces the

optimum to be on the vertices (whereas in a non-bounded polyhedron, there could be no optimum) and simplex algorithm's idea is therefore to parse over the vertices, after visiting one vertex, it goes onto a neighbour vertex that increases the current value.

2. **Interior Point Methods** [11]: Unlike the simplex method, IPMs do not parse the vertices of the polyhedron. It encompasses multiple sub-methods (e.g primal-dual [1]) that all possess a barrier function (usually a log-barrier) in its objective function . The fact that IPMs do not parse edges allow them to avoid degeneracy which is a recurrent problem in simplex algorithms.

3. **Ellipsoid**: [12] Through Ellispoid method, it's been first shown by Khachiyan that it was possible to solve LPs in polynomial-time. However in general graphs its performance is falling behind Simplex and Interior Point Methods'.

2.2.4 Semi-Definite Programming

When Linear Programs are not enough of a relaxation, comes into handy Semi-Definite Programs [17]. Similarly to Linear Programs, there are multiple methods to solve the program, despite some properties and methods not translatable to SDP, such as LP's Simplex not translating to SDP. As we will see later on, SDP is extending LP to more general problems of same nature.

Definition 2.2.3. *Semi-Definite Programs(SDP) are problems with convex constraints, and a linear objective function over the intersection of the affine space defined by the constraints and the cone of positive semidefinite matrices: a spectrahedron. Semi-definite programs can be expressed intuitively the following way:*

$$\max C \bullet X \mid C, X \in \mathbb{S}^n$$

$$\textit{with } A_i \bullet X = b_i \textit{, i=1,...,m}$$

$$X \geq 0 \mid A_i \in \mathbb{S}^n, b \in \mathbb{R}^n$$

We see a lot of similarities between the standard form of an LP and the above expression. Indeed, SDP can be called a relaxation of LP through Vector Programs (VP).

Some possible methods for solving a general SDP is briefly explained below:

1. **Interior Point Methods** [15]: After expressing a problem into a primal and dual, from there we define the Lagrangian which is the sum of the original objective function, the potential function (most of times logarithmic barrier), and the distance between the solution of primal and the solution of dual. The goal is to minimize such a Lagrangian (put a minus before the objective function if it was a maximization problem), and in

order to do so one way is to use first order criteria to find the gradient which leads us to some search direction. The Lagrangian allows for an unconstrained optimization.

2. **Gradient Projection** [3]: Unlike IPM, we do not require dual in this part. Indeed, we compute the gradient of the primal and then project the direction into the feasible space. During the process, we need to make sure the next step is indeed respecting constraints. Even though, it is possible to optimize both objective functions with gradient based method in IPM and Gradient Projection (for IPM the optimization does not necessarily needs to be gradient based) what differs with IPM is also that we optimize here, in a constrained space.

3 Methods on the Maximum cut problem

3.1 Definitions and First Notions

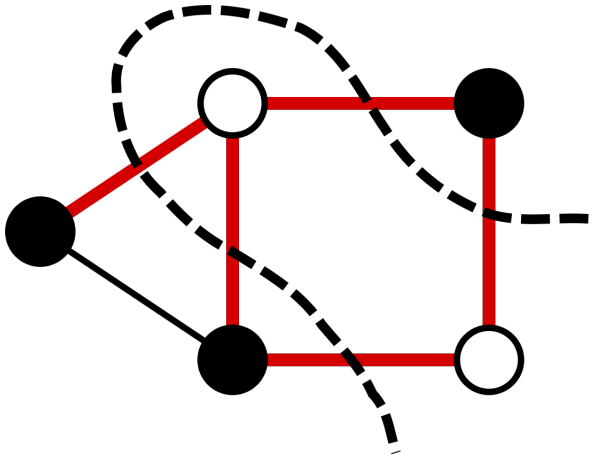

Figure 3.1 – MaxCut graph

Let us consider a weighted graph (3.1). Informally, the maximum cut problem is about finding bi-partition such that the total weights of the edges linking these two sets is maximum. In the picture above (which is an undirected unweighted graph), we have the set of white nodes and the set of black nodes. Supposing that all edges have unitary weight, the weight of the cut defined by the sets of white and black nodes is 4, which happens to be a maximum cut of this graph.

Problem 2 (Maximum Cut problem)**.** *Consider an undirected edge-weighted graph* (G, w)*, where* $G = (V, E)$*. A cut is defined to be a partition into two disjoint sets* S *and* T *of the vertices. The weight of a cut is given by the function* $W : V \times V \to \mathbb{R}$*:*

$$W(S,T) := \sum_{i \in S, j \in T} w_{ij}$$

And the maximum cut is a cut of maximum weight and defined by:

$$M(G,w) := \max_{\forall S \subset V} W(S, V \setminus S)$$

In the following proposition y_i represents the belonging of the vertex i, that is to either set S if $y_i = 1$ else to $V \setminus S$. We also annotate n to be the number of vertices.

Proposition 1. *We can formulate Maximum-Cut as the following integer program:*

$$\max \frac{1}{4} \sum_{1 \le i,j \le n} w_{ij}(1 - y_i y_j) \ ,$$

with $y_i \in \{-1, 1\}$.

Proof. Let's take a look at what is the term to maximize. The edge linking i to j is to be considered in the total weights when $i \in S$ and $j \notin S$, to encode this we have $y_i = 1$ and $y_j = -1$ and indeed $w_{ij}(1 - y_i y_j) = 2w_{ij} \neq 0$ that is counted in the total weights. Now if $i \in S$ and $j \in S$ then $w_{ij}(1 - y_i y_j) = 0$ and this term does not contribute to the cut weight. Note that when an edge belongs to the cut, we add $2w_{ij}$ instead of w_{ij}. Therefore we need to divide a first time the total weight obtained by 2. We divide once more by 2 because we count twice each edge. After these operations we obtain the formulated integer program. □

3.2 First Approaches to Maximum Cut

Having stated the problem 2, let's see some direct ways that we could come up with to solve it.

3.2.1 Enumeration Algorithm

The idea in this algorithm, is to simply try out all combinations of vertices to put in the set S, and among these combinations pick one having the maximum weight. The Combinations are the union of length i element of a bi-partition with $i \in \{1...\lfloor \frac{n-1}{2} \rfloor\}$.

Algorithm 1 Enumerating Algorithm

```
1: Result: MaxCut
2: W ← [ ]
3: for i ∈ {1, ..., ⌊(n−1)/2⌋} do
4:     for S ∈ Combinations(i) do
5:         W append weights(S,V-S)
6:     end for
7:     MaxCut ← max(W)
8: end for
```

Proposition 2. *For any graph S, the enumeration algorithm is terminating and converges in exponential-time.*

Proof. We see that the first and second for loop are iterating over a finite set. Therefore the program does terminate. Now in terms of convergence time, let's fix a number m of vertex we want to put in the set S. The task of picking a set S of size m is therefore to pick m elements among n. The number of sets S of size m is thence $\binom{n}{m}$. Due to symmetry of the sets S and $V - S$, the total number of combinations is the following:

$$\text{Card(Combinations)} = \sum_{i=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n}{i} = 2^{\lfloor \frac{n-1}{2} \rfloor}$$

The number of combinations being exponential of n, we get a convergence in exponential-time. □

3.2.2 Greedy Algorithm

Greedy Algorithm is as said in subsection 2.2.2, maximizing each step's benefit but not necessarily reaching the global maximum. In our case the greedy is used as follows (with S the set we keep track of):

1. Pick a maximum weight edge (i, j) that has not yet been picked
2. if i is already assigned to a set then send j to its complementary
3. if both i and j are not assigned yet then assign $\min(i, j)$ in S and $\max(i, j)$ to its complementary
4. repeat until all vertices have been assigned to either S or $V \setminus S$

Algorithm 2 Greedy Algorithm

Result: MaxCut
Input: E set of edges
$S \leftarrow [\,]$
$T \leftarrow [\,]$
Assigned$\leftarrow [\,]$
$E \leftarrow$ Ordered(E) //By weight
for $(i, j) \in E$ **do**
  if $\min(i, j) \in S$ and $\max(i, j) \notin$ Assigned **then**
    T append $\max(i, j)$
  else if $\min(i, j) \in T$ and $\max(i, j) \notin$ Assigned **then**
    S append $\max(i, j)$
  else if $\max(i, j) \in S$ and $\min(i, j) \notin$ Assigned **then**
    T append $\min(i, j)$
  else if $\max(i, j) \in T$ and $\min(i, j) \notin$ Assigned **then**
    S append $\min(i, j)$
  else
    S append $\min(i, j)$
    T append $\max(i, j)$
  end if
  Assigned append i
  Assigned append j
  if Size(Assigned)=Size(V) **then**
    STOP
  end if
end for
MaxCut $\leftarrow$ weights(S,T)

The program will indeed terminate since we decrease the number of vertices left to assign at each iteration.

Here is a first simple example of an application of greedy algorithm on the following graph 3.2.

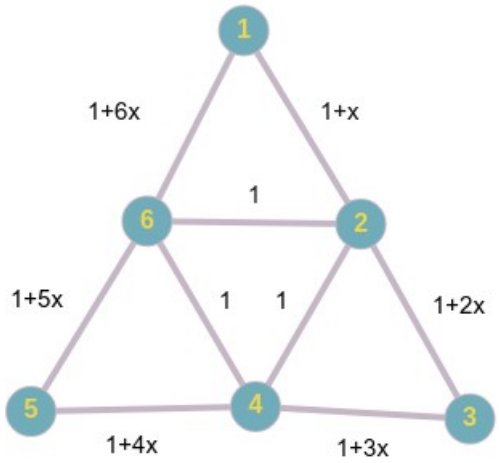

Figure 3.2 – Example of graph

In this situation, we see that the Greedy Algorithm would partition the graph as follows: $S = \{1,3,5\}$ and the value computed by the algorithm would then be $6+21x$. Notice that in this configuration, we managed to leave the paired weights out of the cut, meaning that all the weights linking even nodes that could have been potentially within the MaxCut would be left out. The flaw then comes from there. Intuitively, adding more vertices would then decrease the value found by the Greedy Algorithm.

Proposition 3. *Greedy Algorithm for Maximum Cut is not an approximation algorithm in the sense of definition 2.1.1.*

Proof. More formally, we would define such graphs (3.2) as follows:

$$V = \{1,...,N,...,2N\}$$

$$w_{i,i+1} = 1 + ix \text{ and } w_{1,2N} = 1 + 2xN$$

$$w_{2i,2j} = 1$$

with $i \in \{1,...,2N-1\}$ and $j \in \{1,...,N\}$.
The cut created by the Greedy Algorithm for such a graph would then be: $S = \{1,3,...,2N-1\}$ with

$$W(S, V-S) = \sum_{i=1}^{2N}(1+ix) = 2N + x\frac{2N(2N+1)}{2}$$

Now we decide to set $x = \frac{1}{N(2N+1)}$. To consider the ratio of the value found by Greedy and the optimal, we can consider this other cut that is sub-optimal but behaving better than Greedy.

The cut is namely defined by $S' = \{1,2,3,...,N\}$ where we see that

$$W(S', V-S') = \left\lfloor \frac{N}{2} \right\rfloor^2 + 2 + xN + 2Nx$$

Again with $x = \frac{1}{N(2N+1)}$, the ratio is

$$\frac{W(S,V-S)}{W(S',V-S')} < \frac{2N+1}{\lfloor \frac{N}{2} \rfloor^2 + \frac{3}{2N-1} + 2} < \frac{2N+1}{\lfloor \frac{N}{2} \rfloor^2} < \frac{8}{N}$$

Now if we pick $N = \lfloor \frac{8}{\epsilon} \rfloor + 1$, then we can create whatever ratio and therefore we obtain that: For all ϵ exists N such that

$$\text{Greedy}_{OPT} < \epsilon \text{OPT}$$

So we conclude that Greedy Algorithm is not an approximation algorithm according to 2.1.1. □

3.2.3 Randomized 0.5 Algorithm

Randomized 0.5 Algorithm is a very simple algorithm, but it is a randomized **0.5-approximation algorithm** and its derandomization happens to be a 0.5-approximation algorithm to Max-cut.The algorithm is simply about separating all the vertices with a probability of 0.5 for each vertex. Variance is very high when the graph is small and/or sparse.

Algorithm 3 0.5 Algorithm

```
1: Result: MaxCut
2: S ← [ ]
3: for i ∈ {1,...,n} do
4:     if uniform random(0,1) ≥ 0.5 then
5:         S append i
6:     end if
7: end for
8: MaxCut ← weights(S,V-S)
```

The program will indeed terminate since we decrease the number of vertices left to parse at each iteration.

Proposition 4. *The* 0.5-***Algorithm*** *is a randomized* 0.5-***approximation algorithm***.

Proof. Let X_{ij} a random variable such that X_{ij} is 1 if edge (i, j) is in the cut and 0 otherwise. Let W be a random variable be the total weight of the current cut: $W = \sum_{(i,j)\in E} w_{i,j} X_{ij}$. Let OPT denote the optimum value of the maximum cut. Then, by linearity of expectation, we

know that

$$E[W] = \sum_{(i,j)\in E} w_{i,j} E[X_{ij}] = \sum_{(i,j)\in E} w_{i,j} \Pr[\text{(i,j) in cut}]$$

Note that the probability of two vertices (i, j) to be in different sets, is the sum of the probability of $i \in S \mid j \in V \setminus S$ and $j \in S \mid i \in V \setminus S$.

$$\Pr[\text{(i,j) in cut}] = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$

From there we get,

$$E[W] = \frac{1}{2} \sum_{(i,j)\in E} w_{i,j} \geq \frac{1}{2}\text{OPT}$$

The rightmost inequality comes from the fact that the number of edge in the cut will always be smaller or equal than the total number of edges. □

3.3 The solutions to a relaxation problem, SDP

As we have just shown, the methods presented in section 3.2 may not be efficient: Enumeration method would take far too much time to compute on a sizeable graph, while greedy algorithm does not guarantee any lower-bound since it is not an approximation algorithm 2.1.1. As for LPs, the form in which MaxCut is expressed is not compatible. In this situation, there is a necessity to find other methods. Methods that are built on the following propositions:

Proposition 5. ***Relaxation*** *of previously stated integer program of maximum cut problem is expressed as follows:*

$$\max \frac{1}{4} (W \bullet \mathbb{1}_n - W \bullet Y) \ ,$$

with $W = (w_{ij})_{i,j}$, $Y_{ii} = 1$ *and* $Y \geq 0$. *(We remind that* $A \bullet B = \text{Tr}(A^T B)$*)*

Proof. In order to transform the integer program into a Semi-definite Program, we will go through the relaxation that is to change the IP into Vector Program of the following form:

$$\max \frac{1}{4} \sum_{1\leq i,j\leq n} w_{ij}(1 - v_i^T v_j) \ ,$$

with $v_i \in \mathbb{R}^m$ and $v_i^T v_i = 1$.

This is indeed a relaxation since we notice that by assigning to the first entry of v_i the value y_i we obtain the same value as for IP.

We notice that:

$$\max \frac{1}{4} \sum_{1\leq i,j\leq n} w_{ij}(1 - v_i^T v_j) = \max(\frac{1}{4}(W \bullet \mathbb{1}_n) - \frac{1}{4} \sum_{1\leq i,j\leq n} w_{ij} v_i^T v_j)$$

As we require for Y to have $Y_{ii} = 1$ and positive symmetry definiteness, having the definition of Y being $Y_{ij} = v_i^T v_j$, we do get all the conditions and finally obtain: the required proposition. □

Now the task is to solve 5, and find the maximum argument for Y. Note that

$$\text{argmax}_Y \frac{1}{4}(W \bullet \mathbb{1}_n - W \bullet Y) = \text{argmax}_Y(-W \bullet Y)$$

SDP formulation of Max Cut: Following the relaxation, we now simply have to solve $\max(-W \bullet Y)$, which boils down to:

$$\max C \bullet X$$

$$E_{ii}X = \mathbb{1}_i \mid i = 1, ..., m \ ,$$

with $X_{ii} = 1$ and $X \geq 0$.

3.3.1 Gradient Projection Method

This method was first proposed by Burer and Monteiro in [3]. In our case we use a lower triangular decomposition of X obtained from its Cholesky factorization. Which turns our original SDP into:

$$\max C \bullet (LL^T)$$

$$X = LL^T$$

$$E_{ii}X = \mathbb{1}_i \mid i = 1, ..., m \ ,$$

with $X_{ii} = 1$ and $X \geq 0$. Before we describe and justify the algorithm, let's define some extra notations:

1. $\varphi : L^n \to \mathbb{R}$ and $\varphi(L) = C \bullet (LL^T)$
2. $l : M_{n\times n} \to L^n$ and $l(A) = L'$ with $L'_{ij} = A_{ij}$ if $i \geq j$.
3. $N : L^n \to L^n$ and $N(L)_{ij} = \frac{L_{ij}}{||L_{ij}||_2}$
4. p_i^k is the i-th row of matrix P^k, and $p_i'^k$ is i-th row of matrix P'^k and similarly for L
5. Armijo Rule: $\varphi(N(L + \alpha P)) - \varphi(L) \geq \sigma\alpha(P'^T P)$

The general idea is to compute the direction at which $C \bullet (LL^T)$ is increasing while keeping the next L within the space of Lower Triangular matrices. In order to keep the next L (that is computed by adding it with the gradient), we project the gradient on L. Now for the amplitude of the step, we will calibrate it using a Line Search, Armijo Rule, to pick the distance of αP toward which we move.

Algorithm 4 Gradient Projection Algorithm

1: **Result:** MaxSDP
2: **Input:** $\alpha > 0$, $\sigma \in (0,1)$, L initial feasible solution
3: **for** $i \in \{1,\dots,n\}$ **do**
4: $P' \leftarrow 2l(CL)$
5: P defined by $p_i = p'_i - < p'_i, l_i > l_i$
6: Pick $\alpha < 0$ respecting Armijo rule
7: $L = N(L + \alpha P)$
8: **end for**
9: MaxSDP ← L

3.4 Probabilistic Rounding of relaxation

Consider Y^* being the solution of the SDP for Max Cut, in order to obtain feasible solution to answer the original IP problem, rounding techniques are needed. We will present in this part the Goemans-Williamson probabilistic rounding method. The original proof and lemma can be once again found in Goemans-Williamson's book [8].

Probabilistic rounding: the 0.878 algorithm

Algorithm: having the optimal solution Y^* of the relaxation problem, we consider its column vectors $y_i \in \mathbb{R}^n$. We obtain a vector $r = (r_1, ..., r_n)$ with component picked from $N(0,1)$ distribution. Now the heuristic is to have the vertex $i \in S$ if $y_i^T r \geq 0$ and else $i \in V \setminus S$.

Lemma 3.4.1. *Consider r vector picked with distribution $N(0,1)$ component wise, and let r' the projected vector of r onto a plane. The normalization of r' is uniformly distributed on the unit circle of the plane.*

Lemma 3.4.2. *The probability that edge (i,j) is in the cut is $\frac{1}{\pi}\arccos(y_i^t y_j)$*

Proof. Let r vector previously defined, and r' its projection onto the plane defined by y_i, y_j. We know that $y_i^T r = y_i^T r'$ and similarly $y_j^T r = y_j^T r'$. Reminding that the heuristic is to have $i \in S$ if $y_i^T r \geq 0$, we have the following cases:

1. $y_i^T r \geq 0$ and $y_j^T r \geq 0$ which means $|\angle(r', y_i)|, |\angle(r', y_j)| < \frac{\pi}{2}$ then $i, j \in S$ and (i,j) not in cut
2. $y_i^T r < 0$ and $y_j^T r < 0$ which means $|\angle(r', -y_i)|, |\angle(r', -y_j)| < \frac{\pi}{2}$ then $i, j \in V \setminus S$ and (i,j) not in cut
3. and remains $y_i^T r < 0$ and $y_j^T r \geq 0$ and vice versa, to have (i,j) in cut

Without loss of generality, consider y_i such that $\angle(y_i, y_j) \geq 0$ and $\alpha = \angle(y_i, r')$ (signed radian angle). Bearing in mind the previous cases, we have (i,j) in cut when $\alpha \in [\frac{\pi}{2} - \angle(y_i, y_j), \frac{\pi}{2}] \cup$

$[-\frac{\pi}{2}, -\frac{\pi}{2} - \angle(y_i, y_j)]$. Note that both intervals have length $\angle(y_i, y_j)$. Using Lemma 3.3.1, we have uniform distribution of the r':

$$P[\text{edge (i,j) is in cut}] = \frac{2\angle(y_i, y_j)}{2\pi}$$

and with $\|y_i\|, \|y_j\| = 1$ (since we are working in unit sphere)

$$P[\text{edge (i,j) is in cut}] = \frac{1}{\pi}\arccos(y_i^T, y_j)$$

□

Theorem 3.4.3. *Rounding the semi-definite program the way done in the rounding algorithm gives an .878-approximation to maximum cut.*

Proof. Consider W the random variable that maps a cut to its weight. We then have, using the Lemma 3.3.2,

$$E[W] = \sum_{(i,j)\in S} w_{ij} P[\text{edge (i,j) is in cut}] = \sum_{(i,j)\in S} w_{ij} \frac{1}{\pi}\arccos(v_i^T v_j)$$

Now, knowing that $\frac{1}{\pi}\arccos(x) \geq 0.878\frac{1}{2}(1-x)$:

$$E[W] \geq 0.878\frac{1}{2}\sum_{(i,j)\in S} w_{ij}(1 - v_i^T v_j) = 0.878\ \mathrm{OPT}_{\mathrm{SDP}} \geq \mathrm{OPT}_{\mathrm{IP}}$$

□

3.5 Deterministic Rounding of relaxation

In order to derandomize a solution of SDP, multiple methods exists. We will be giving multiples ones of them here, and their respective complete proof. A first method was proposed by Goemans Williamson around the same time they published the probabilistic rounding. The method consisted in using Conditional Probability on the dimension of the random vector while keeping it "symmetrical" when reducing dimensions i.e keeping the coordinates normally distributed. Unfortunately, this idea was proved wrong by Mahajan and Ramesh in 1999 (see [13]).

3.5.1 Johnson-Lindenstrauss Dimensionality Reduction/Rounding

The method is not entirely mathematical and rely heavily on the pseudorandomness used in implementations. While the Goemmans Williamson method is stochastic, in implementations, the generating of Normal Distribution is pseudorandom, meaning that there is a finite number of random numbers that will be outputted. Now to test all of them, we require $2^{n^{O(1)}}$ operations,

but that will make the algorithm deterministic. Unfortunately, as we can see, the number of candidates vectors are too numerous, and therefore came different ways to reduce the computation time. Let's first start with assumed notions that are unfortunately not proved in this paper.

Theorem 3.5.1 (Johnson-Lindenstrauss Lemma [5])**.** *Let $v_1, ..., v_m$ be a sequence of vectors in $\mathbb{R}^d$ and let $\epsilon, F \in (0,1]$. Then in $O(dm(\log n + 1/\epsilon)^{O(1)})$ deterministic time we can compute linear mapping $A : \mathbb{R}^d \to \mathbb{R}^k$ where $k = O(\log(1/F)/\epsilon^2)$ such that*

$$k\|v_i^2\| \leq \|Av_i\|^2 \leq k(1+\epsilon)\|v_i\|^2$$

for all at least a fraction $1 - F$ of i's.

Lemma 3.5.2. *Suppose that $v_1, ...v_m \in \mathbb{R}^d$ are unit vectors, and $A : \mathbb{R}^d \to \mathbb{R}^k$ is a linear mapping such that $|v|^2 \leq |Av|^2 \leq (1+\epsilon)|v|^2$ for all $v = v_i$ or $v_i - v_j$. Let w_i be the vector Av_i normalized unit length. Then $w_i^T w_j \leq v_i^T v_j + \epsilon$ for all i, j.*

Lemma 3.5.3. *For every dimension k and error parameter $\delta > 0$, there is a distribution X' which can be sampled using $O(\log(k/\delta))$ random bits, $O(\log(k/\delta))$ space, and $O(\frac{1}{\delta}\log^2(k/\delta))$ time. and is close to the $1-$dimension normal distribution in the following sense. Let $v, w \in \mathbb{R}^k$ be unit vectors, and let $c \in \mathbb{R}$. Consider the randomized algorithm $A_{v,c}$ which picks a random vector r in $\mathbb{R}^k$ by choosing each coordinate independently from the 1-dimensional normal distribution, and then outputs 1 or 0 depending on whether $v^T r \geq c$. If $A_{v,c}$ instead uses the approximate distribution X', its output distribution is the same within additive error $\pm\delta$. A similar statement holds for the algorithm $B_{v,w}$, which outputs 1 or 0 depending on whether $sgn(v^T r) = sgn(w^T r)$*

Proof. **Steps:** Generally speaking, these steps consist in using the derandomized Johnson-Lindenstrauss algorithm from Theorem 3.4.4, to reduce dimension, but having the required vectors for reduction. After getting these vectors in dimension k and normalized, we use the now diminished possibilities of normal distributed entries vectors.

Step1: Dimensionality Reduction We define the set $V \cup V'$ to be the sets defined as follows:

1. V a multiset with $\lceil \frac{m}{n} \rceil$ (we don't deal with $m < n$ since that will then be easy to find maxcut) apparition of each vector v_i
2. V' is the set of all vectors $v_{ij} = v_i - v_j$ for $(i, j) \in E$

that we will work from to get the candidates for a dimension reduction.
Using the Derandomized Johnson-Lindenstrauss procedure, we obtain the mapping $f : \mathbb{R}^n \to \mathbb{R}^k$ with $k = O(\log(1/\epsilon)/\epsilon^2)$ such that (we define $F = \epsilon$):

1. for at least a fraction $1 - 2F$ of v_i's we have $1 \leq |f(v_i)|^2 \leq 1 + \epsilon$

2. for at least $1-2F$ of the vectors v_{ij} we have $|v_{ij}|^2 \leq |f(v_{ij})|^2 \leq (1+\epsilon)|v_{ij}|^2$

Denote E' the set of edges (i,j) such that the inequalities stated above works. In this set, for all edge (i,j) we would require it to be among the $1-2F$ vectors v_{ij} and v_i, v_j in the $1-2F$ set that respects the first inequality. This gives us at least $1-6F$ fraction of the edges that satisfies both inequalities.

Define w_i normalization of $f(v_i)$, and using Lemma 2 we have the following inequality for all $(i,j) \in E'$: $w_i^T w_j \leq v_i^T v_j$.
With that let's compute the cut value:

$$\sum_{(i,j)\in E'} \frac{1-w_i^T w_j}{2} \geq \sum_{(i,j)\in E'} \frac{1-v_i^T v_j - \epsilon}{2} \geq -m(\frac{\epsilon}{2}+6F) + \sum_{(i,j)\in E} \frac{1-v_i^T v_j}{2} \tag{3.1}$$

The first inequality is obvious, but the second one is less intuitive. Indeed, (from 3.1) we first pull out the ϵ (we remind that $m = |E|$)

$$\sum_{(i,j)\in E'} \frac{1-v_i^T v_j - \epsilon}{2} \geq -m\frac{\epsilon}{2} + \sum_{(i,j)\in E'} \frac{1-v_i^T v_j}{2}$$

By definition $|E| - |E'| \leq 6F|E|$, and since we are working with unweighted edges (i.e weight 1) in the scope of this proof, we obtain the inequality from (3.1) i.e:

$$\sum_{(i,j)\in E'} \frac{1-v_i^T v_j - \epsilon}{2} \geq -m\frac{\epsilon}{2} - m6F + \sum_{(i,j)\in E'} \frac{1-v_i^T v_j}{2}$$

Since we know that the cut value from the solution of the sdp i.e v_i's acts better than a half approximation algorithm, we then make use of the ineqality $\sum_{(i,j)\in E} \frac{1-v_i^T v_j}{2} \geq \frac{m}{2}$ to obtain (with again $F = \epsilon$)

$$\sum_{(i,j)\in E'} \frac{1-w_i^T w_j}{2} \geq (1-13\epsilon) \sum_{(i,j)\in E} \frac{1-v_i^T v_j}{2} \geq (1-13\epsilon)\text{OPT} \tag{3.2}$$

Step2: Rounding We apply here again the same heuristic than the general Goemans Williamson, but since in implementation, true normal distribution is infeasible, we will use Lemma 2 that gets us close enough.

$$-m\delta + \sum_{(i,j)\in E'} \frac{\arccos(w_i^T w_j)}{\pi} \geq \mu(\sum_{(i,j)\in E'} \frac{1-w_i^T w_j}{2}) - m\delta \geq \mu(1-13\epsilon)\text{OPT} - m\delta \tag{3.3}$$

Using $\mu = 0.87$ and $\delta = \epsilon$ we do get an $(\mu - 13\epsilon)$- approximation algorithm.

Now sampling the normal distribution in Lemma 2 requires $O(\log(k/\delta)$ random bits and $O(\frac{1}{\delta}\log^2(k/\delta))$ time for a k-dimension of the vector. If we use again $\delta = \epsilon$ and with $k =$

$O(\log(1/\epsilon)/\epsilon^2)$ we get $R := O(\log(1/\epsilon))$ random bits and $T := O(\log^2(1/\epsilon)/\epsilon)$ time for each vector. And therefore when enumerating all choices we would have to spend $(2^R T)^k = 2^{O(\log^2(1/\epsilon)/\epsilon^3)}$, in order to obtain the approximation.

□

Remark The dimension reduction can be replaced by a swifter idea given by Goemans Williamson, where we employ conditional probability to obtain a one-dimensional vector where we then simply have to maximize on the choice of the symmetrical vector of dimension 1 and then percolate back to dimension n without loss in the inference heuristic. This idea can be found in again the article written by Mahajan and Ramesh in 1999 (see [13])

3.5.2 Conditional Probability Method

Fortunately a second method exists, and we would want to explore and prove. We will be using the method of Conditional Probabilities to derandomize. We will again require an external Lemma.(proved here [2])

Lemma 3.5.4 (Piecewise Approximation of Conditional Probability)**.** *If $X \in \mathbb{R}^n$ is a standard normal distruted random vector, $a, b \in R^n$, $c_1, c_2, d_1, d_2 \in \mathbb{R}$ and x is a variable, then there exists a constant k and the interval $(-\infty, \infty)$ can be partitioned into intervals $I_1, ..., I_k$, such that $P[a^T X \geq c_1 + c_2 x \text{ and } b^T X \geq d_1 + d_2 x]$ can be expressed as follows,*

$$P[a^T X \geq c_1 + c_2 x \text{ and } b^T X \geq d_1 + d_2 x] = q_k(x) + \delta_k \text{ if } x \in I_k \ ,$$

where $|\delta_1|, ..., |\delta_k|$ are $O(\epsilon)$ and $q_1(x), ..., q_k(x)$ are polynomials of degree $O(-\log\epsilon)$ that can be constructed in $O(\log^2 \epsilon)$ time provided that $|a|, |b|$ and $a^T b$ are known.

Method Instead of working on the conditionality for the choice of vertex vectors to keep the expected value the same, we work on the dimension of the symmetry vector.

The recurrence step is: Given a derandomized vector (acting as the symmetry vector) of dimension $i-1$: $X_1 = x_1, ..., X_{i-1} = x_{i-1}$ we do a maximization task, which is to maximize the following:

$$\max_{x \in \mathbb{R}} \sum_{(u,v) \in E} w_{u,v} P[x_u^T X \geq 0 \text{ and } x_v^T X < 0 \mid X_1 = x_1, X_2 = x_2, ..., X_i, = x] \tag{3.4}$$

We then set X_i to be the argument maximum of equation (3.4). This process goes on until all dimensions (as much dimension as there is in a vertex vector) are derandomized.

Let's firstly rewrite

$$p_{u,v}(x_1) = \sum_{(u,v) \in E} w_{u,v} P[x_u^T X \geq 0 \text{ and } x_v^T X < 0 | X_1 = x_1]$$

We know from Lemma 3.5.4 that $p_{u,v}(x_1)$ can be approximated by $k(u,v)$ polynomials of degree $\log(n/\epsilon)$ (ϵ being the value error between the polynomial and the conditional probability) over different ranges of x that we denote $q^i_{u,v}(x)$ for the i-th polynomial. Starting with a polynomial Q=$q^1_{u,v}(x)$ we then subtract $q^1_{u,v}(x)$ on the interval where $q^2_{u,v}(x)$ is closer to $p_{u,v}(x)$. We repeat this minimization procedure recursively and we get a polynomial Q on which we can maximize to obtain the arguments X_i that achieve the maximizing task.

Remark: Despite this method being a derandomization, we do not have a certification for the lower bound of the performance, unlike the conditional probability used on 0.5 algorithm [13], where we were keeping the expectancy as lower bound. On top of that, we unfortunately did not implement the polynomial approximation of conditional probability and used another implementation that is explained in chapter 4.

4 Implementations of Methods and Performance Comparisons

4.1 Introduction to Benchmark

Benchmark are used to test the efficiency and performance of the different methods we implemented.

4.1.1 Personal Graphs

These personal graphs were made by care by me. They are smaller size graphs, for an easier computation of the maximum cut value through enumeration method. The sample created works as follows (each file represents one graph):

1. x_n_i: For each size n, we generate 100 (indexed by i) unweighted graphs with edge probability of $x \in \{0.5\}$

Remark Unlike the next section where some graphs have fixed density, we warn the user the here we did not fix a specific density even though there is a high chance the number of edges is distributed around the edge probability. Plus, since graphs are smaller in this part, there might be some bigger difference in terms of number of edge compared to the desired x density.

4.1.2 Standard Graphs

These standard graphs were generated by biqmaclib to whom we give full credits for the gaphs [16]. Parts of the instances are not complementary to our study case, where for instance the weight is negative. In the files available we shortlist:

1. g05_n.i: For each size n, we generate ten unweighted graphs with edge probability of 0.5. And there are $n = 60, 80, 100$.
2. pwd_100.i: For each density d we have ten graphs with integer weights varying from

[0,10]. And the densities are $d = 0.1, 0.5, 0.9$ with size $n = 100$ graph.

Remark The density is computed by the following formula: (again we are working in an undirected graph environment)

$$D = \frac{2|E|}{|V|(|V|-1)}$$

with $|E|$: number of edges and $|V|$: number of vertices. Note also, that the optimum values are given with the generated graphs, and therefore these optimum values were not computed by myself.

4.2 Performance Comparisons

In this section, we compare the different methods in terms of time and performance on the handmade graphs and the standard graphs.

4.2.1 Performance on Personal Graphs

NB of Vertices	Standard GW	Grad Proj GW	Greedy	Half_approx
5	0.991	0.482	0.876	0.56
6	0.992	0.503	0.778	0.591
7	0.989	0.56	0.673	0.646
8	0.984	0.529	0.635	0.604
9	0.977	0.591	0.532	0.652
10	0.975	0.606	0.502	0.683
11	0.974	0.597	0.456	0.700
12	0.973	0.621	0.458	0.663
13	0.966	0.615	0.401	0.714
14	0.970	0.631	0.405	0.719

Table 4.1 – Performance (ratio over OPT) on 0.5 probability of edge

Remark: As expected the standard solver (where we employ the cvxpy library [7] to solve SDP before rounding) performs very well. In comparison the gradient projection based GW performs very much poorly. However note that both method uses the same rounding algorithm (i.e GW). The low performance therefore comes from the solving of the SDP, we notice despite all of that , that this performance improves with the number of edges (this can be seen both in this table and in the following examples). A possible explanation is that for smaller graphs (smaller number of edges) the gradient is close or simply a 0 vector that does not give a good enough of direction to improve the SDP value. Now for Greedy, solutions for smaller graphs are indeed close to the optimal value, and similarly for Half Approximation algorithm which behave again better than expected. A potential hint on why Half Approximation is that the

graphs are also generated with 0.5 edge probability.

NB of Vertices	Standard GW	Grad Proj GW	Greedy	Half_approx
5	0.0391	0.137	5.7e-05	3.46e-05
6	0.039	0.1436	5.781e-05	3.260e-05
7	0.043	0.159	5.907e-05	3.207e-05
8	0.0467	0.172	6.73e-05	3.15e-05
9	0.053	0.19	6.247e-05	3.42e-05
10	0.0557	0.208	6.28e-05	3.372e-05
11	0.060	0.2235	6.074e-05	3.56e-05
12	0.074	0.2611	7.336e-05	3.857e-05
13	0.0961	0.318	8.00e-05	4.166e-05
14	0.096	0.329	8.789e-05	3.855e-05

Table 4.2 – Time (s) spent on 0.5 probability of edge

Remark: Unsurpisingly, SDP-based methods do take longer time to find its solution, whereas Greedy and Half Approximation are rather fast. Now between Standard GW and Gradient Projection GW, we have fixed the number of iteration (i.e number of gradient computation) to 100 for Gradient Projection which increased the time inference for Grad Proj.

4.2.2 Performance on Standard Graphs

nb vertices and density	Standard GW	Grad Proj GW	Greedy	Half_approx
60 | 0.5	0.979	0.839	0.11271	0.425
80 | 0.5	0.987	0.8206	0.0859	0.4614
100 | 0.1	0.9397	0.678	0.342	0.3584
100 | 0.5	0.9797	0.879	0.0553	0.448
100 | 0.9	0.988	0.919	0.153	0.467

Table 4.3 – Performance (ratio over OPT) of methods on different types of graphs

Remark: In regards to previous results on smaller graphs we do see a large improvement of Gradient Projection method on SDP. Again through the examples of density 0.1 we see that smaller number of edges seems to impact on its performance.

nb vertices and density	Standard GW	Grad Proj GW	Greedy	Half_approx
60 | 0.5	24.183	2.2304	0.000467	7.62e-05
80 | 0.5	120.9	4.557	0.000507	0.00010
100 | 0.1	603.99	7.667	0.0006519	8.543e-05
100 | 0.5	457.78	6.069	0.000712	9.97e-05
100 | 0.9	482.01	7.15	0.00288	9.95e-05

Table 4.4 – Time (s) spent on different types of graphs

Remark: Since we fixed the number of iteration for Gradient Projection, its time spent is lower than the standard's. Greedy and Half Approximation's time remains consistent and not to far from one another.

4.3 Expected probability VS deterministic

In this part we apply the rounding in two different ways. The two rounding possible are Deterministic (4.3) and GM94 rounding (3.4). The goal is then to figure out the trade off of such a deterministic method and the general GM94 rounding .
As mentioned in chapter 3 the method that we used here for deterministic is different:
Deterministic method implementation: The general idea is that starting from a random vector (that it the same for all graphs of same size) we replace the first coordinate with values between $[-1, 1]$ and keep the value that maximizes the current cut value. We do the following procedure recursively on all coordinates of the originally random vector.

nb vertices and density	Grad Proj GW	Grad Proj Derand
60 | 0.5	0.8436	0.9936
80 | 0.5	0.8489	0.9889

Table 4.5 – Performance (ratio over OPT) of methods on different types of graphs

nb vertices and density	Grad Proj GW	Grad Proj Derand
60 | 0.5	2.78	25.68
80 | 0.5	4.92	129.04

Table 4.6 – Time (s) spent on different types of graphs

Remark To compute these values 10 graphs of 60|0.5 (vertices|density) and 10 graphs of 80|0.5 (vertices|density) were used. We see that while, the time spent on the Gradient Projection Derandomized version is largely longer than the Gradient Projection GW, the derandomized

version compensates its shortcoming with a very high performance.

Seeing that there is such a trade-off, we interested ourselves in seeing whether for the same allocated time, the Gradient Projection GW (by sampling multiple random vectors) would outperform Gradient Projection Derandomized. In the following **two graphs**, Gradient Projection Derandomized were scoring respectively **536** and **921** (with optimum values of those graphs being 536 and 929)

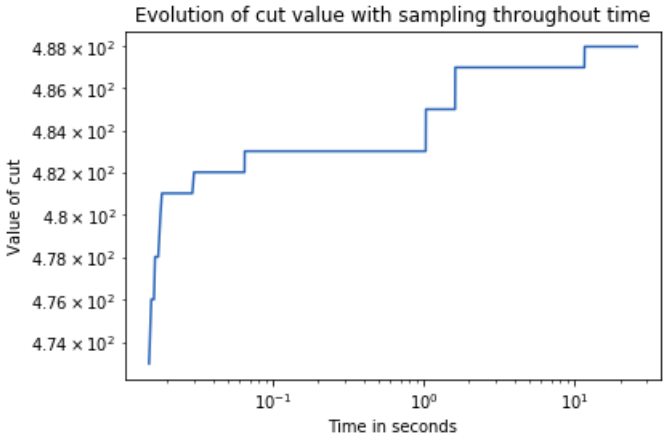

Figure 4.1 – On graph_g05_80.0

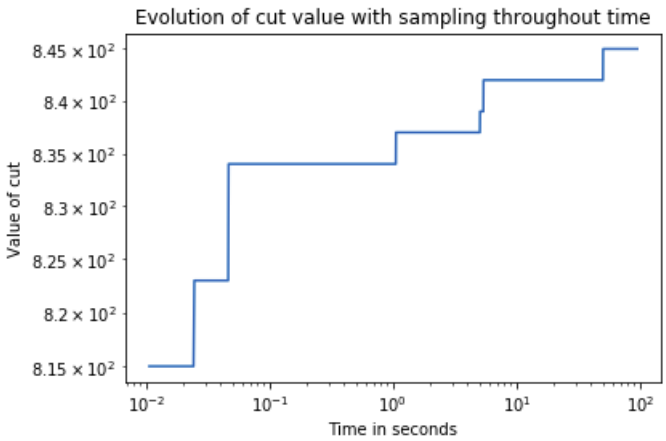

Figure 4.2 – On graph_g05_80.0

Remark We see improvements of cut values found by Gradient Projection GW but even after a same allocated time, Gradient Projection GW gives **0.91** on the first graph and **0.922** on the second graph while Gradient Projection Derandomized scores respectively **1** and **0.991**.

Bibliography

[1] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 227–236, 2007.

[2] Ankur Bhargava and S Rao Kosaraju. Derandomization of dimensionality reduction and SDP based algorithms. In *Workshop on Algorithms and Data Structures*, pages 396–408. Springer, 2005.

[3] Samuel Burer and Renato DC Monteiro. A projected gradient algorithm for solving the maxcut sdp relaxation. *Optimization methods and Software*, 15(3-4):175–200, 2001.

[4] George B Dantzig, Alex Orden, Philip Wolfe, et al. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5(2):183–195, 1955.

[5] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report*, 22(1):1–5, 1999.

[6] Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of Maxcut. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 53–61, 2007.

[7] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[8] Michel X Goemans and David P Williamson. . 879-approximation algorithms for max cut and max 2sat. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 422–431, 1994.

[9] Jean-François Hêche, Thomas M Liebling, and Dominique De Werra. *Recherche opérationnelle pour ingénieurs*, volume 2. PPUR presses polytechniques, 2003.

[10] Dorit S Hochba. Approximation algorithms for NP-hard problems. *ACM Sigact News*, 28(2):40–52, 1997.

[11] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[12] Leonid G Khachiyan and Michael J Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. Technical report, Cornell University Operations Research and Industrial Engineering, 1990.

[13] Sanjeev Mahajan and Hariharan Ramesh. Derandomizing semidefinite programming based approximation algorithms. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 162–169. IEEE, 1995.

[14] Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)*, 22(1):115–124, 1975.

[15] Masayuki Shida, Susumu Shindoh, and Masakazu Kojima. Existence and uniqueness of search directions in interior-point algorithms for the SDP and the monotone SDLCP. *SIAM Journal on Optimization*, 8(2):387–396, 1998.

[16] Angelika Wiegele. Biq mac library—a collection of Max-Cut and quadratic 0-1 programming instances of medium size. *Preprint*, 2007.

[17] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of semidefinite programming: theory, algorithms, and applications*, volume 27. Springer Science & Business Media, 2012.

ÉCOLE POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

Statement of Academic Integrity Regarding Plagiarism

I, the undersigned Chun Hei Michael CHAN [family name, given name(s)], hereby certify on my honor that:

1. The results presented in this report are the product of my own work.

2. I am the original creator of this report.

3. I have not used sources or results from third parties without clearly stating thus and referencing them according to the recommended rules for providing bibliographic information.

Declaration to be copied below:

I hereby declare that this work contains no plagiarized material.

Date 12/04/2020

Signature