KTH:
Royal Institute of Technology

SF2827

Topics in Optimization

Semidefinite programming applied to graph-coloring problems

Author:

Gilles Layon 870328-A199 gilles.layon@student.uclouvain.be

Teacher:

Anders Forsgren

Date of submission: 2009-03-27

Abstract

Semidefinite programs do not always have all the properties that linear programs usually have. For strong duality to hold unless the problem and its dual have to satisfy some conditions, such as the Slater condition. If it does, then the problem can be solved using interior-point methods, ellipsoid method, a cutting plane algorithm,... The interior-point methods are numerous since the complementarity equation the matrices must be symmetrized and there exist many different ways of doing it.

Many applications use SDP-problems relaxations to get an approximate solution of the optimal value. Graph-coloring theory is one of these. A relaxation of a coloring problem can be set under the form of an semidefinite program that can be solved using the methods mentioned above. The optimal solution k obtained is then bounded by

$$\omega(G) \leq k \leq \chi(G),$$

where $\omega(G)$ is the size of the largest clique in G and $\chi(G)$ is the chromatic number of G, i.e. the smallest number of colors necessary to color G properly. After that, different algorithms can be used to round the solution k to a coloration of G. The algorithm presented in this report is quite simple and can color the graph in $O(n^{0.631} \log_2 n)$ colors. Other algorithms, more complicated, can color the graph using only $O(n^{0.387} \log_2 n)$ colors.

Contents

1 Introduction

Semidefinite programs are a quite recent kind of optimization problems that actually have many fields of application. The main difference between SDP-problems and linear programs is that the unknowns are matrices and not simple vectors. Also, these problems do not always have the same properties as LP-problems and the solving methods had to be adapted. Different practical problems that are actually older than semi-definite problems can be relaxed and expressed on the form of a semidefinite program. This can happen when trying to solve a MAX CUT problem or, as it will be detailed in this report, when attempting to vertex-color a graph properly.

This paper first gives several informations about SDP-problems and how they can be solved. Crucial differences compared to LP's and principal difficulties are presented, and some ways of dealing with them are proposed. These sections are not very detailed, but some references are given where further informations (about interior-point methods for example) can be found.

The next section will be about graph-coloring. It will first be shown how a coloring problem can be relaxed to a semidefinite program and then how the solution to that program can be rounded to reach an approximation of the optimal solution to the coloring problem. The rounding step will be done for the particular case of 3-colorable graphs, but a generalization of that method can be applied to any k-colorable graph (see [KMS98] for further informations). Afterwards, the method presented is applied to Petersen's graph, detailing each step of the algorithm. An example of a MATLAB implementation of the algorithm is also appended to this report. Some other algorithms, more efficient but more complicated, will also be mentioned but not detailed here. References to these will be given for interested readers.

Finally, a short conclusion will summarize the main results of this report.

2 Generalities and notations

Semidefinite programming is a particular class of optimization problems. A general semidefinite program can be written as follows:

$$\begin{aligned} \min \quad & \text{trace}(CX) \\ s.t. \quad & \text{trace}(A_i X) = b_i, \quad i = 1, ...m, \\ & X \succeq 0. \end{aligned}$$

In this case, the unknowns are matrices and the inequalities signs are considered in a way different than in normal inequations. If the considered constraint is

$$A \cdot X \geq B,$$

this actually means that the matrix $A \cdot X - B$ must be positive semidefinite. Now and further, this constraint will be written

$$X \succeq 0,$$

to avoid any confusion. Moreover, the constraint

$$X = X^T$$

is often added to the problem. From now until the end of this report, the problem that will be considered will include the fact that X must be symmetric. The general form will then be:

$$\begin{aligned} \min \quad & \text{trace}(CX) \\ s.t. \quad & \text{trace}(A_i X) = b_i, \quad i = 1, ...m, && \text{(1a)} \\ & X = X^T \succeq 0, && \text{(1b)} \end{aligned}$$

and the references to matrices and vectors A_i, C, or b_i will refer to that notation all along the report.

The dual of this problem is given by

$$\begin{aligned} \min \quad & \sum_{i}^{m} b_i y_i \\ s.t. \quad & \sum_{i=1}^{m} A_i y_i \preceq C, \end{aligned}$$

which can be rewritten with equality constraints:

$$\begin{aligned} \min \quad & \sum_{i}^{m} b_i y_i \\ s.t. \quad & \sum_{i=1}^{m} A_i y_i + Z = C, && \text{(2a)} \\ & Z \succeq 0. && \text{(2b)} \end{aligned}$$

From now until the end of this report, the variables X, y and Z will respectively denote the primal solution, the dual solution and the slack variables of the dual.

3 Difficulties of dealing with SDP problems

A first difficulty raised when trying to deal with SDP-problems is due to the constraint

$$X \succeq 0.$$

This is actually equivalent to

$$\lambda(X) \geq 0,$$

where $\lambda(X)$ refers to the eigenvalues of X. This problem of exactly finding the eigenvalues of a matrix being difficult to solve in a polynomial time, one could wonder if solving semidefinite problems is really worth it. In fact, some methods like the ellipsoid method or the interior method claim to find a solution to a semidefinite program in polynomial time, which could seem to be weird. The point is that these methods only give a solution that is exact with a certain tolerance. For example, a MATLAB program that solves an SDP-problem by interior methods with six digits of accuracy (i.e, $error \leq 10^{-6}$) can be found in [HRVW96]. This algorithm is polynomial, but the solution will not be exact.

A second problem raised by semidefinite programming is that strong duality does not always hold. In linear optimization, weak and strong duality are properties that can be used to compute simplex multipliers or the solution to the dual.
In semidefinite optimization, the duality gap is given by

$$\begin{aligned} \mathrm{tr}(CX) - b^T y &= \mathrm{tr}(\sum_{i=1}^{m} A_i y_i + Z)X) - b^T y \\ &= \mathrm{tr}(ZX) + \sum_{i=1}^{m} (\mathrm{tr}(A_i X) - b_i) y_i \\ &= \mathrm{tr}(ZX) \geq 0, \end{aligned}$$

and weak duality thus always hold.
On the opposite, strong duality does not. Some conditions must be satisfied by the constraints of the primal and the dual, as for example the Slater condition: for the problems defined as above, there must exist a feasible solution X such that

$$\mathrm{tr}(A_i X) = b_i \quad \text{and} \quad X \succ 0,$$

and also a feasible dual solution y, Z such that

$$Z = \sum_i A_i y - C \quad \text{and} \quad Z \succ 0.$$

Different examples where strong duality does not always hold can be found in either [Tod01] or [HA04].

One last problem encountered when dealing with semidefinite optimization is related to the solving methods. As it can be seen from the formulation above, the semidefinite requirement of equation (1b) is quite difficult to handle.
The reason is that the constraint $X = X^T$ can not be handled like this by a numerical solver and that it has to be changed by another equation to be sure that the optimal solution will respect that. The mapping most commonly used is the following:

$$H_P(M) = \frac{1}{2}((PMP^{-1}) + (PMP^{-1})^T).$$

It can be shown that with this,

$$H_P(ZX) = \mu I \quad \text{if and only if} \quad ZX = \mu I.$$

Some other ways of symmetrizing this equation have been developed. Discussions on the form that the matrix P above should have or on other ways of symmetrizing this equation can be found in [LR05], [Mon03] or [Zha98].

4 Solution methods

There exist different methods used to solve SDP problems. An old kind of them is called the ellipsoid method. This kind of method where the first one that could be used to solve linear programs in the beginning of the 70's. These were also the first methods which were used to prove that LP problems could be solved in polynomial time (by Khachiyan in 1979).

Nowadays, the simplex method can be used to solve LP problems, while the interior point method is often used to solve LP problems, NLP problems and SDP problems. The interior point method most commonly used is the primal-dual interior point method. These methods tend to use the primal optimilaty conditions as well as the dual ones and the primal-dual complementarity conditions to find an approximate solution to the problem. The goal is to solve a linear system of equations iteratively with a parameter μ that decreases at each iteration to converges toward the optimal solution when $\mu = 0$.
The system that one wants to solve at the beginning is the following

$$\begin{aligned} \sum_i A_i y_i + Z &= C, \\ \operatorname{tr}(A_i X) &= b_i, \quad \forall i, \\ XZ &= 0, \\ X, Z &\succeq 0, \end{aligned}$$

which is usually perturbed to

$$\sum_i A_i y_i + Z = C, \tag{3a}$$

$$\operatorname{tr}(A_i X) = b_i, \quad \forall i, \tag{3b}$$

$$XZ = \mu I. \tag{3c}$$

Equation (3c) is then symmetrized using different techniques and the system is solved approximately for different values of μ tending to zero.
The system uses starting values $X^{(0)}, y^{(0)}$ and $Z^{(0)}$ and then uses Newton's method to find the update step $\Delta X, y$ or Z:

$$\begin{aligned} X^{(n)} &\leftarrow X^{(n-1)} + \Delta X^{(n-1)}, \\ y^{(n)} &\leftarrow y^{(n-1)} + \Delta y^{(n-1)}, \\ Z^{(n)} &\leftarrow Z^{(n-1)} + \Delta Z^{(n-1)}. \end{aligned}$$

When including this into the system of equations (3), the equation (3c) must be linearized with regard to ΔX and ΔZ:

$$\begin{aligned} \mu I &= (X + \Delta X)(Z + \Delta Z) \\ &= XZ + X\Delta Z + \Delta X Z + \Delta X \Delta Z \\ &\approx XZ + X\Delta Z + \Delta X Z. \end{aligned}$$

So that the final system to be solved at each iteration regarding to ΔX, Δy and ΔZ is

$$\begin{aligned} \sum_i A_i \Delta y_i + \Delta Z &= C - \sum_i A_i y_i - Z, \\ \operatorname{tr}(A_i \Delta X) &= b_i - \operatorname{tr}(A_i X), \\ X\Delta Z + \Delta X Z &= \mu I - XZ. \end{aligned}$$

As mentioned in the previous section, there exist many different ways to symmetrize equation (3c) which lead to different systems to be solved. One example could be to use mapping mentioned previously and to replace the equation (3c) before linearization by

$$H_P(ZX) = \mu I.$$

Anyway, this will not be detailed here but further informations about interior-point methods and symmetrization strategies can be found in [HRVW96], [Mon03] or [Zha98].

5 SDP in graph-coloring theory

5.1 Introduction

The problem of graph k-coloring that will be considered here can be defined as follows:

Definition: *For a graph $G = (V, E)$, where V defines a set of n nodes and E defines a set of edges, find an assignment of k colors to the nodes so that two nodes connected by and edge do not share the same color. In this case, the graph is said to be $k-$colorable.*

The lowest number of colors needed to color a graph is called *chromatic number* of the graph and it is denoted as $\chi(G)$.

Another graph-coloring problem that can be considered is the edge-coloring of a graph. In this case, one tries to assign a color to each edge of a graph $G = (V, E)$ such that two edges incident to a same node do not share the same color. Anyway, the problem considered in this report will be the first one: finding a proper vertex-coloring of a graph G.
These coloring problems have many practical applications, such as scheduling, allocation of radio frequencies,etc. Moreover, the problem of finding if a graph G can be colored with k colors is NP-complete (see [Dun08] for a list of NP-complete problems) and the problem of finding the chromatic number of a graph is NP-hard (see [GJ79]). One particular case where a polynomial algorithm could be found is the case where the graph is bipartite, since this implies that the graph does not have any odd loops and that it can then be colored with only 2 colors.

The following section of this report will describe how a relaxation of the coloring problem can be obtained and solved by semidefinite programming. To do that, some theorems, lemmas or properties will be presented. The proofs of these will not always be available but, as often as possible, a reference will be made to a place where a sketch of the proof can be found.

5.2 Semidefinite relaxation

The first concept that needs to be defined here is the concept of a vector k-coloring of a graph.

Definition: *Given a graph $G = (V, E)$ on n vertices, and a real number $k > 1$, a vector k-coloring of G is an assignment of unit vectors v_i from the space $\mathbb{R}^n$ to each vertex $i \in V$, such that for any two adjacent vertices i and j the dot product of their vectors satisfies the inequality*

$$\langle v_i, v_j \rangle \leq \frac{-1}{k-1}.$$

This coloration can be related to another type of graph-coloring: the matrix k-coloring.

Definition: *Given a graph $G = (V, E)$ on n vertices, a matrix k-coloring of the graph is an $n \times n$ symmetric positive semidefinite matrix X, such that*

$$X_{ii} = 1 \quad \text{and} \quad X_{ij} \leq \frac{-1}{k-1} \text{ if } (i,j) \in E.$$

Now, the problem of finding a matrix k-coloring of a graph can be formulated as the following optimization problem:

$$\begin{aligned}
\min \ & \frac{-1}{k-1} \\
\text{s.t. } & X_{ij} \leq \frac{-1}{k-1} \text{ if } (v_i, v_j) \in E, && (4)\\
& X_{ii} = 1 \ \forall i \in \{1...n\}, && (5)\\
& X_{ij} = X_{ji} \ \forall i, \ j, && (6)\\
& X = X^T \succeq 0. && (7)
\end{aligned}$$

From this, one can see that the constraints define in fact a matrix symmetric and positive semidefinite. Moreover, the constraint (4) can be reformulated:

$$\begin{aligned} X_{ij} \leq \frac{-1}{k-1} &\Leftrightarrow e_i^T X e_j \leq \frac{-1}{k-1}, \\ &\Leftrightarrow \mathrm{tr}(X e_j e_i^T) \leq \frac{-1}{k-1}. \end{aligned}$$

From this, the following SDP problem can be set:

$$\begin{aligned} \min \ & \frac{-1}{k-1} \\ \text{s.t. } & \mathrm{tr}(X e_j e_i^T) \leq \frac{-1}{k-1} \text{ if } (v_i, v_j) \in E, \\ & \mathrm{tr}(X e_i e_i^T) = 1, \ \forall i \in \{1...n\}, \\ & X = X^T \succeq 0. \end{aligned}$$

Since the solution to that problem is a positive semidefinite matrix, a Cholesky factorization can be found so that

$$X = UU^T.$$

From this, it can be seen that the vectors v_i forming the vector k-coloring of G are in fact the rows of U. Hence, one can conclude that a graph G will have a k-vector coloring if and only if it has a matrix k-coloring.

Now, the point is to bound the solution of the SDP. On one hand, an upper bound can be found by using some different theorems:

Theorem: *For any integers n and k such that $k \leq n+1$, there exists k unit vectors $v_i \in \mathbb{R}^n$ such that*

$$\langle v_i, v_j \rangle = \frac{-1}{k-1} \quad \forall i \neq j \in \{1...k\}. \tag{8}$$

Proof: (from [KMS98])
Clearly it is sufficient to prove the theorem for $n = k-1$. (For $n > k-1$, we make the coordinates of the vectors 0 in all but the first $k-1$ coordinates.) We begin by proving the claim for $n = k$. We explicitly provide unit vectors $v_1^{(k)}, ...v_k^{(k)} \in \mathbb{R}^k$ such that

$$\langle v_i^{(k)}, v_j^{(k)} \rangle \leq \frac{-1}{k-1} \text{ for } i \neq j.$$

The vector $v_i^{(k)}$ is such that

$$e_j^T v_i^{(k)} = \begin{cases} -\sqrt{\frac{1}{k(k-1)}} & \forall j \neq i, \\ \sqrt{\frac{k-1}{k}} & \text{if } i = j. \end{cases}$$

and one can verify that

$$\|v_i^{(k)}\|_2 = \frac{1}{k(k-1)}(k-1) + \frac{k-1}{k} = \frac{1+k-1}{k} = 1.$$

Moreover, it can also be checked that for any $i \neq j$,

$$\begin{aligned} \langle v_i, v_j \rangle &= (k-2)\frac{1}{k(k-1)} - 2\sqrt{\frac{k-1}{k^2(k-1)}} \\ &= \frac{k-2}{k(k-1)} - \frac{2}{k} \\ &= \frac{k-2-2(k-1)}{k(k-1)} \\ &= \frac{-k}{k(k-1)} \\ &= \frac{-1}{k-1}, \end{aligned}$$

so that the claim is proven for $n = k$.
Moreover, if $e^{(k)}$ denotes a vector full of ones of $\mathbb{R}^k$, then if holds that

$$\begin{aligned}\langle e^{(k)}, v_i^{(k)} \rangle &= -(k-1)\sqrt{\frac{1}{k(k-1)}} + \sqrt{\frac{k-1}{k}} \\ &= -\sqrt{\frac{(k-1)^2}{k(k-1)}} + \sqrt{\frac{k-1}{k}} \\ &= -\sqrt{\frac{k-1}{k}} + \sqrt{\frac{k-1}{k}} \\ &= 0,\end{aligned}$$

so that all the k vectors are in $\mathrm{Ker}(e^{(k)})$ which is of dimension $k-1$. This shows that all k vectors are actually in a $(k-1)$-dimensional hyperplane of $\mathbb{R}^k$. One can then find a basis of that subspace of dimension $k-1$ such that the vectors of that basis satisfy the inequality (8). This then proves the theorem for $n = k-1$ and, as mentioned above, for every n such that $n \geq k-1$. □

Using that, one can demonstrate the following theorem and deduce an upper bound for the solution of the coloration problem:

Theorem: *Any k-colorable graph has a vector k-coloring.*
Proof: This can be proved easily using the previous theorem; if a graph of n nodes is k-colorable, then it holds that $k \leq n+1$ and so there exist k unit vectors in $\mathbb{R}^n$ satisfying the inequality $\langle v_i, v_j \rangle \leq \frac{-1}{k-1}$. Mapping each of these vectors with a color gives a vector k-coloring of the graph. □

From this, it is straightforward to obtain an upper bound on the solution of the coloration problem, namely that

$$k \leq \chi(G).$$

On the other hand, a lower bound can be found by using the following theorem:

Theorem: *Let G be k-vector colorable and let i be a vertex in G. The induced subgraph on the neighbors of i is vector $(k-1)$-colorable.*
Proof: (from [KMS98])
Note that to apply that theorem, $k \geq 3$ must hold. Indeed, if $k = 2$, then there must be no edges in the remaining subgraph since $\langle v_i, v_j \rangle \leq -\infty$ must hold.
Suppose then that $k \geq 3$ and let $v_1, ..., v_n$ be a vector k-coloring of G. Then, it can be assumed without loss of generality that

$$v_i = (1, 0, 0, ..., 0).$$

Associate with each neighbor j of i a vector v'_j obtained by projecting v_j onto coordinates 2 through n and then scaling it up so that v'_j has unit length; i.e:

$$(v'_{j,1}\ v'_{j,2}\ ...\ v'_{j,n}) = \frac{(v_{j,2}\ v_{j,3}\ ...\ v_{j,n})}{\sum_{i=2}^{n} v_{j,i}^2},$$

and observe that

$$\langle v_j, v_i \rangle \leq \frac{-1}{k-1},$$

so that the projection of v_j onto the first coordinate is negative and has magnitude at least $\frac{1}{k-1}$. When scaling v'_j so that is has unit length by dividing it by its own norm, this implies that

$$\frac{1}{\|v'_j\|_2} \geq \frac{k-1}{k(k-1)}.$$

Thus,

$$\begin{aligned}\langle v'_j, v'_{j'}\rangle &\leq \frac{(k-1)^2}{k(k-2)}\left(\langle v_j, v_{j'}\rangle - \frac{1}{(k-1)^2}\right)\\ &\leq \frac{(k-1)^2}{k(k-2)}\left(\frac{-1}{k-1} - \frac{1}{(k-1)^2}\right)\\ &\leq \frac{1}{k-2}\left(\frac{-(k-1)-1}{k}\right)\\ &\leq \frac{-1}{k-2}.\end{aligned}$$

□

Using this theorem recursively, one can show that any graph containing a $k+1$ clique will not be k-vector colorable. In fact, this would lead to the inequality

$$\langle v_i, v_j\rangle \leq \frac{-1}{k-(k+1-1)} = \frac{-1}{0} = -\infty,$$

which is impossible. Here comes then the lower bound of the solution given by the semidefinite program. If the largest clique in G is denoted by $\omega(G)$, then it holds that

$$\omega(G) \leq k \leq \chi(G).$$

6 Rounding the solution given by an SDP relaxation

Finding a feasible coloration for a graph with k colors is a "difficult" problem. [Dun08] presents different NP-complete problems, among which the one of coloring a graph with 3 colors. Actually, the problem of finding a k-coloring of a graph is equivalent to the problem of finding k independent sets in it, which is NP-complete.
Since the problem of finding a 3-coloring of a graph is NP-complete, we will only focus on this one in this part of the report and therefore the algorithm described in this section is applicable to 3-colorable graphs only.
The main point is here to show how a vector 3-coloration can be transformed in a coloration of a graph using a probabilistic algorithm. A generalization of the obtained result to any k-colorable graphs can be found on the paper [KMS98].

The first part of this section will be an explanation of how a simple algorithm using hyperplane relaxation works. We will first give a theoretical and general explanation of the algorithm and then will apply it to Petersen's graph, represented in figure 1. The last part of the report will be a short section dedicated to more efficient but also more complicated algorithms. This will not be described in details here, but some references will be given for further information.

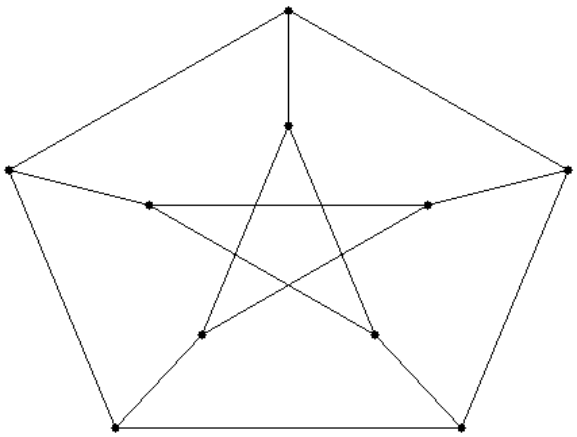

Figure 1: Petersen's graph

6.1 Rounding algorithm via hyperplane partitions

6.1.1 Theoretical explanation

Before describing the algorithm, the concept of semicoloring of a graph $G = (V, E)$ must be defined;

Definition: *A k-semicoloring of a graph G=(V,E) is an assignment of k colors to at least half of the nodes of G such that two adjacent nodes do not share the same color.*

From this, one can say that if it is possible to get a k-semicoloring of any graph, then it is possible to obtain a $(k \log_2 n)$-coloring of that graph. This is useful since the algorithm that will be used further actually uses the solution to the SDP relaxation of the coloring problem to find an $O(n^{0.631})$-semicoloring of the graph and then iterates to finally get a coloration using $O(n^{0.631} \log_2 n)$ colors.

Here follows now the algorithm that is supposed to find a semicoloring of G. It can be found in the paper [PZ]. Let us first describe the different steps and then explain why it

finally gives the results presented above.

Algorithm:

1. Step 1: Solve the vector relaxation problem and obtain a 3-vector coloring of G.
2. Step 2:
   - Let $p = \lceil 2 + \log_3 \Delta(G) \rceil$, where $\lceil . \rceil$ denotes the rounding to the upper integer and $\Delta(G)$ is the maximum degree of any node in G.
   - Generate p hyperplanes $r_1, ...r_p$ and compute $\langle r_i, v_j \rangle \ \forall i \in \{1...p\}, \ \forall j \in \{1...n\}$.
   - Let
$$\begin{aligned} R_1 &= \{v_j | \langle r_i, v_j \rangle \geq 0 \forall\ i\}, \\ R_2 &= \{v_j | \langle r_1, v_j \rangle < 0, \langle r_2, v_j \rangle \geq 0, ... \langle r_p, v_j \rangle \geq 0\}, \\ &\vdots \\ R_{2^p} &= \{v_j | \langle r_i, v_j \rangle < 0 \ \forall\ i\}. \end{aligned}$$
3. Step 3: Assign color i to all the nodes in R_i.

Let us now give some detailed explanation about each step of the algorithm.
In the first step, the value of k given by the algorithm may not be an integer. In this case, it is considered in this implementation that it will have to be rounded to the nearest integer toward plus infinity. If this rounded integer is lower than 3, then the node of maximum degree in the graph has a degree equal to one. Hence, the graph can be colored in two colors. If this integer is strictly larger than 3, then the graph is not 3-colorable (as it has been proven in section (5.2) that any k-colorable graph is vector k-colorable and the algorithm can not be used.

The second step is the most important of the algorithm and uses some properties about hyperplanes and probabilities. The following definitions and lemmas will be useful to demonstrate that a semi-coloring of the graph can be found with probability $\frac{1}{2}$.

Definition: *A hyperplane H is said to separate two vectors if they do not lie on the same side of the hyperplane. For any edge $(i, j) \in E$, the hyperplane is said to cut the edge if it separates the vectors v_i and v_j associated with the vertices i and j.*

Lemma: (from [GW95], without proof) *The probability that a random plane separates an edge whose extreme vectors make an angle of θ is given by*

$$P(r \text{ separates vectors } v_i \text{ and } v_j | angle(v_i, v_j)) = \frac{\theta}{\pi}.$$

Since the vectors given for the k-vector coloration must satisfy

$$\|v_i\|_2 = 1 \ \ \forall\ i,$$

then

$$\langle v_i, v_j \rangle = \|v_i\|_2 \|v_j\|_2 \cos(v_i, v_j) = \cos(v_i, v_j) \leq \frac{-1}{k-1} = \frac{-1}{2},$$

since the considered graph is k-colorable.
Using this, it can be proven the following lemma:

Lemma: *Applying the algorithm to a 3-colorable graph will give a semicoloring of that graph using $O(\Delta^{0.631})$ colors (where Δ is the highest degree of any node in G) with probability of at least $\frac{1}{2}$.*
Proof: (from [PZ]) Let us first see that as $p = 2 + \log_3(\Delta)$, it holds that

$$\begin{aligned} 2^p &= 4 \times 2^{\log_3(\Delta)} \\ &= 4 \times \Delta^{\log_3 2} \\ &= O(\Delta^{\log_3 2}). \end{aligned}$$

Moreover, it also holds that

$$\begin{aligned} P(col(i) = col(j) | (i,j) \in E) &= P(\text{no random plane cuts the edge}) \\ &= \left(1 - \frac{\arccos(\langle v_i, v_j \rangle)}{\pi}\right)^p \\ &\leq \left(1 - \frac{\arccos(\frac{-1}{2})}{\pi}\right)^p \\ &\leq \left(1 - \frac{2}{3}\right)^p \\ &\leq \frac{1}{3^{2+\log_3(\Delta)}} \\ &\leq \frac{1}{9\Delta}. \end{aligned}$$

From there, if the total number of edges in the graph is denoted by m, the average number of uncutted edges is bounded by

$$E(\textit{uncutted edges}) \leq \frac{m}{9\Delta},$$

and, since the number of edges in a graph is bounded by

$$m \leq \frac{n\Delta}{2},$$

it holds that

$$\begin{aligned} E(\textit{uncutted edges}) &\leq \frac{\frac{n\Delta}{2}}{9\Delta} \\ &\leq \frac{n}{18} \\ &\leq \frac{n}{8}. \end{aligned}$$

From there, the following inequality (used without demonstration) can be used:

Markov's inequality: ([MR95], p46) *For the probability space, if X is any random variable and $a > 0$, then*

$$P(|X| \geq a) \leq \frac{E(X)}{a}.$$

It follows that

$$P(\textit{more than } \frac{n}{4} \textit{ uncutted edges}) \leq \frac{\frac{n}{8}}{\frac{n}{4}} \leq \frac{1}{2}.$$

Following the third and last step, it follows that the algorithm finds a semicoloring of G with probability at least $\frac{1}{2}$ and using $O(\Delta^{\log_3 2})$ colors.
Since $\Delta < n$ and $\log_3 2 < 0.631$, the algorithm finds a semicoloring of any three colorable graph with probability 0.5 and using $O(n^{0.631})$ colors. □

If this algorithm is used t times, the probability that it finds a semicoloring of G is given by

$$1 - \frac{1}{2^t}.$$

Removing the nodes that have been colored and iterating the procedure on the remaining subgraph, a final coloration will be found in at most $\log_2 n$ steps and using $O(n^{0.631} \log_2 n)$ colors.

6.1.2 Example: application to Petersen's graph

Petersen's graph is a graph composed with ten nodes, each of them with degree three. This graph can be colored with 3 colors, as shown in figure 2. Moreover, the graph does not contain any clique properly speaking and is not colorable with 2 colors since is contains odd cycles. As it has been shown previously, the first iteration of the algorithm should then give an optimal value of the relaxation problem such that

$$2 \leq k \leq 3.$$

A MATLAB version of the code used to compute the steps described below is appended to the report. This code has not been optimized numerically and could certainly be improved, but as the main purpose here is only to describe visually how the algorithm really works, this is not very important.

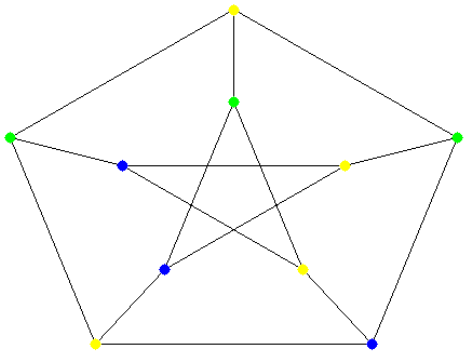

Figure 2: Optimal coloration of Petersen's graph: 3 different colors are used.

For this graph, the nodes have been numbered as shown on figure 2. The adjacency matrix describing the graph is then

$$M = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

This matrix is $n \times n$ where n is the number of nodes in the graph. For each element of the matrix, it holds that

$$M_{ij} = \begin{cases} 1 & \text{iff } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Using that, the relaxed SDP problem to be solved can be set:

$$\begin{aligned} \min \ & \frac{-1}{k-1} \\ \text{s.t. } & \operatorname{tr}(X e_j e_i^T) \leq \frac{-1}{k-1} \text{ if } M_{ij} = 1, \\ & \operatorname{tr}(X e_i e_i^T) = 1 \ \forall i \in \{1...n\}, \\ & X = X^T \succeq 0. \end{aligned}$$

and the optimal k and the final matrix X given by the solver are respectively

$$k = 2.5$$

and

$$X = \begin{pmatrix} 1.000 & 0.167 & -0.667 & -0.667 & 0.167 & -0.667 & 0.167 & 0.167 & 0.167 & 0.167 \\ 0.167 & 1.000 & 0.167 & -0.667 & -0.667 & 0.167 & -0.667 & 0.167 & 0.167 & 0.167 \\ -0.667 & 0.167 & 1.000 & 0.167 & -0.667 & 0.167 & 0.167 & -0.667 & 0.167 & 0.167 \\ -0.667 & -0.667 & 0.167 & 1.000 & 0.167 & 0.167 & 0.167 & 0.167 & -0.667 & 0.167 \\ 0.167 & -0.667 & -0.667 & 0.167 & 1.000 & 0.167 & 0.167 & 0.167 & 0.167 & -0.667 \\ -0.667 & 0.167 & 0.167 & 0.167 & 0.167 & 1.000 & -0.667 & 0.167 & 0.167 & -0.667 \\ 0.167 & -0.667 & 0.167 & 0.167 & 0.167 & -0.667 & 1.000 & -0.667 & 0.167 & 0.167 \\ 0.167 & 0.167 & -0.667 & 0.167 & 0.167 & 0.167 & -0.667 & 1.000 & -0.667 & 0.167 \\ 0.167 & 0.167 & 0.167 & -0.667 & 0.167 & 0.167 & 0.167 & -0.667 & 1.000 & -0.667 \\ 0.167 & 0.167 & 0.167 & 0.167 & -0.667 & -0.667 & 0.167 & 0.167 & -0.667 & 1.000 \end{pmatrix},$$

and using the Cholesky factorization finally gives the following matrix whose rows are the vectors assigned to each node of the graph.

$$V = \begin{pmatrix} 1.000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.167 & 0.986 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.667 & 0.282 & 0.690 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.667 & -0.563 & -0.172 & 0.456 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.167 & -0.704 & -0.517 & -0.456 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.667 & 0.282 & -0.517 & -0.456 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.167 & -0.704 & 0.690 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.167 & 0.141 & -0.863 & 0.456 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.167 & 0.141 & 0.345 & -0.913 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.167 & 0.141 & 0.345 & 0.913 & -0.000 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The next step contains the randomness of the algorithm. As the algorithm will repeat this step until it finds a suitable semicoloring of the graph, the set of random vectors r_i presented in table 1 is a feasible one.

r_1	r_2	r_3
0.087	-0.193	0.356
-0.159	-0.364	0.252
0.284	-0.394	0.343
-0.309	0.059	-0.151
-0.166	0.194	-0.184
-0.321	0.245	0.029
-0.082	0.110	-0.268
-0.133	-0.198	-0.230
0.354	-0.283	-0.226
0.270	0.121	0.121

Table 1: Random vectors used to generate the hyperplanes of the algorithm.

After that, the scalar product of each r_i with one of the vectors v_i has been computed. Since the only important thing is the sign of the result, table 2 only contains a 1 if $\langle v_i, r_j \rangle \geq 0$ and a 0 otherwise. If the notation R_j defined as in the theoretical description of the algorithm is used and if the color j is assigned to every node in R_j, then the coloration represented on the left hand side of figure 3 is obtained. As it can be seen, some nodes are not properly colored since they have the same colors as one of their neighbors. Uncoloring these nodes finally gives a semicoloring of the graph. After that, the different steps above can be repeated on the remaining uncolored subgraph to obtain the final coloration represented on the right hand-side of figure 3.

In this particular case, there is only one node unproperly colored, but there could be several. Moreover, the remaining subgraph contains only one node of degree zero, so that the algorithm does not need to be used again. Since the adjacency matrix (1×1 here) is full of zeros, all the remaining nodes can be colored with only one color.

	r_1	r_2	r_3
v_1	1	0	1
v_2	0	0	1
v_3	1	0	1
v_4	0	1	0
v_5	1	1	0
v_6	0	1	0
v_7	1	0	1
v_8	0	1	0
v_9	1	0	1
v_{10}	0	0	1

Table 2: Positiveness of the scalar product of the v_i's and the r_j's.

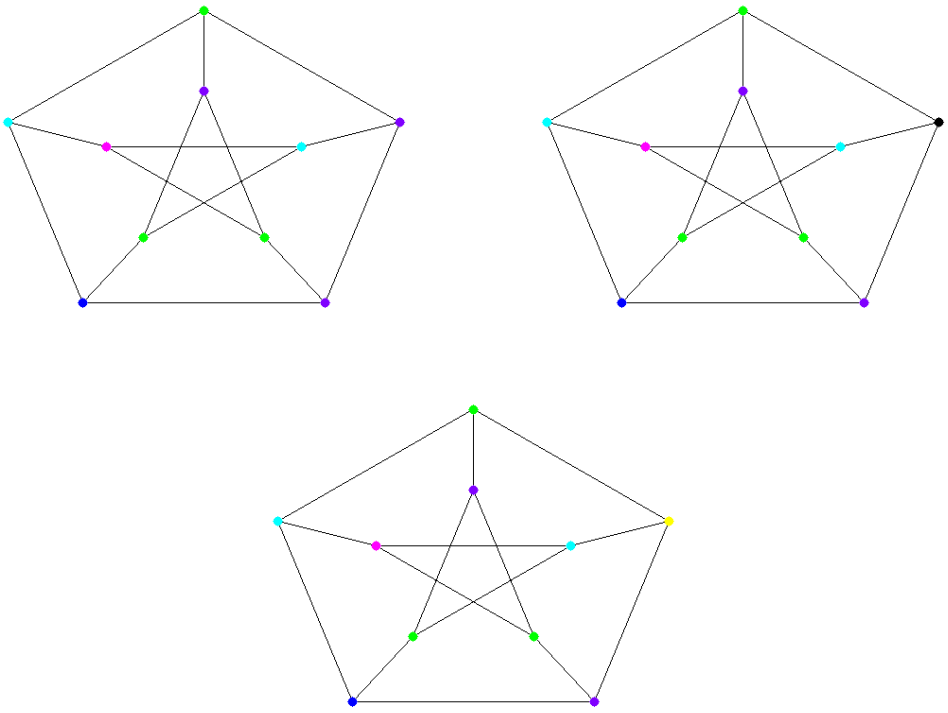

Figure 3: Left & top : preliminary 5-coloring of Petersen's graph. Right & top: feasible 5-semicoloring of Petersen's graph after removing the unproperly colored nodes. Bottom: proper final 6-coloring of Petersen's graph.

6.2 Improved algorithms

Different other algorithms can be used to round the solution given by the SDP-relaxation problem. Some of them actually use the algorithm described above after using an other approach during a certain amount of steps. In fact, a good way to get a better approximation of the coloration is to use Widgerson's algorithm [Wig83] up to a certain bound δ. When all the remaining uncolored nodes have a degree lower than δ, then the algorithm described above is applied. As shown in [KMS98], this leads to a coloration of any 3-colorable graph G with $O(n^{0.387} \log_2 n)$ colors if $\delta \approx n^{0.613}$, which is supposed to be the optimal value for that parameter.

Another rounding algorithm uses vector projections. This algorithm uses only one random vector and some other parameters and transforms the vector k-coloration in a coloration of G. Further theoretical information about this algorithm can be found in [KMS98] and precise description of the different steps of it can be found in [PZ].

7 Conclusion

As it has been seen in this report, SDP problems are special kind of LP's. It has been shown why they are harder to solve than normal linear programs, and the interior-point method that can be used to solve them has been presented. It has also be seen that a semidefinite program can be solved in polynomial time up to a certain precision but not exactly.

After that, it has been shown how semidefinite programming can be applied to a concrete problem: the coloration of a graph. It has then been demonstrated how a relaxation semidefinite problem can be obtained. The solution to that problem has been bounded between the size of the maximal clique of the graph and the chromatic number of the graph.
Afterward, these theoretical results have been applied to three colorable graphs and a rounding algorithm has been presented. The final result is that any 3-colorable graph can be colored using $O(n^{0.631} \log_2 n)$ colors. Finally, more powerful algorithms have also been mentioned to color any 3-colorable graph with, for example, $O(n^{0.387} \log_2 n)$ colors.

A Matlab code

```
function KMS1
close all;
clear all;
clc;

% The matrix coresponding to the Petersen's graph
A = [0 0 1 1 0;0 0 0 1 1;1 0 0 0 1;1 1 0 0 0;0 1 1 0 0];
B = [0 1 0 0 1;1 0 1 0 0;0 1 0 1 0;0 0 1 0 1;1 0 0 1 0];
con = [A eye(5);eye(5) B];

% defining possible colors to color the graph
colors = [0 1 0;0 0 1;1 1 0;0 1 1;1 0 1;1 1 1;1 0.5 0;0 1 0.5;...
      0.5 0 1;0.3 0.3 1;1 0.3 0.3;0.3 1 0.3;0.2 0.2 0.8;0.8 0.2 0.2;...
      0.2 0.8 0.2; 0.5 0.5 1; 0.5 1 0.5;1 0.5 0.5];

n = length(con(:,1));
reminding_nodes = [1:n];
conexions = con;

count = 1;
coloration = zeros(n,3);
while length(reminding_nodes)~= 0
      n = length(con(:,1));
      uncolored_nodes = reminding_nodes;
      [semicol reminding_nodes] = semicoloring(con,colors);
      N = [1:n];
      N(reminding_nodes) = 0;
      indexes = find(N);

      coloration(uncolored_nodes(indexes),:) = colors(semicol(find(semicol)),:);

      % updating the conexion matrix and the colors that can be used for the
      % next iteration:
      nb_colors = length(colors(:,1));
      M = [1:nb_colors];
      for j =1:length(indexes)
            if indexes(j)==n
                  p = [n 1:n-1];
            elseif indexes(j) ==1
                  p = [1:n];
            else
                  p = [indexes(j) 1:indexes(j)-1 indexes(j)+1:n];
            end

            % permuting the rows of "con" to delete the ones corresponding to
            % the colored nodes
            perm = eye(n,n);
            Perm = perm(p,:);
            con = Perm*con*Perm';
            con(1,:) = 0;
            con(:,1) = 0;
            M(semicol(indexes(j))) = 0;
      end
      con = con(length(indexes)+1:end,length(indexes)+1:end);
      remaining_colors = find(M);
      colors = colors(remaining_colors,:);
```

```
        plotcolPeterson(conexions, coloration);
        count = count + 1;
end
end

function [semicol reminding_nodes] = semicoloring(con, colors)
n = length(con(1,:));
if sum(sum(con == zeros(size(con))))~= n^2
        cvx_begin SDP
                variable t
                variable X(n,n) symmetric
                minimize t
                X > 0
                for i = 1 : n
                        ei = zeros(n,1);
                        ei(i) = 1;
                        for j = 1 : n
                                if con(i,j) == 1
                                        ej = zeros(n,1);
                                        ej(j) = 1;
                                        trace(X * ej * ei') <= t
                                end
                        end
                        trace(X * ei * ei') == 1
                end
        cvx_end
        kappr = round(-1/t + 1);
        kfloat = -1/t + 1;
        if kappr - kfloat <= 1e - 4
                k = kappr;
        else
           k = max(3, round((-1)/t + 1));
        end
        V = chol(X)';
else
        semicol = ones(n,1);
        reminding_nodes = [];
        return
end

found = 0;

while found~= 1
        % finding the node of highest degree
        Delta = max(sum(con));
        p = max(3, ceil(2 + log(Delta)/log(3)));

        rp = (rand([n, p]) - 0.5);
        rp(1 : end,:) = rp(1 : end,:)/norm(rp(1 : end,:));

        col = zeros(n,p);
        vect = zeros(n,p);
        for i = 1 : n
              vect(i,:) = V(i,:) * rp;
        end

        Ri = (vect >= zeros(size(vect)));
        compare = zeros(2^p, length(vect(1,:)));
```

```
        count = 0;
        for i = 1 : length(vect(1,:))
                poss = nchoosek([1 : length(vect(1,:))], i);
                for j = 1 : length(poss(:,1))
                        compare(count + j, poss(j,:)) = 1;
                        count = count + 1;
                end
        end
        semicolor = zeros(length(Ri(:,1)),1);
        for i = 1 : length(Ri(:,1))
                for j = 1 : length(compare(:,1))
                        if Ri(i,:) == compare(j,:)
                                semicolor(i) = j;
                                break;
                        end
                end
        end
        [found semicol] = trysemicol(con, semicolor);
        if found == 1
                N = [1 : n];
                colored_nodes = find(semicol);
                N(colored_nodes) = 0;
                reminding_nodes = find(N);
        end
end
end

function [found, semicol] = trysemicol(A, colors)

for s = 1 : length(A(:,1))
      for t = 1 : length(A(1,:))
                if s~= t
                        if A(s,t) == 1
                                if colors(s)~= 0 && (colors(s) == colors(t))
                                        colors(t) = 0;
                                end
                        end
                end
      end
end
semicol = colors;
count = length(find(semicol));
if count >= (length(colors)/2)
       found = 1;
else
       found = 0;
end
end

function plotcolPeterson(con, c)
figure;
axis off

r1 = 5;
r2 = 10;
n = length(con(1,:));

if mod(n,2) == 1
```

```
        numb = (n-1)/2;
        x = [0 r1*cos(pi/2+2*pi/numb*[0:numb-1]) r2*cos(pi/2+2*pi/numb*[0:
numb-1])];
        y = [0 r1*sin(pi/2+2*pi/numb*[0:numb-1]) r2*sin(pi/2+2*pi/numb*[0:
numb-1])];

else numb = n/2;
        x = [r1*cos(pi/2+2*pi/numb*[0:numb-1]) r2*cos(pi/2+2*pi/numb*[0:
numb-1])];
        y = [r1*sin(pi/2+2*pi/numb*[0:numb-1]) r2*sin(pi/2+2*pi/numb*[0:
numb-1])];
end

hold on
for i =1:length(x)
      for j = 1:length(y)
            if con(i,j) ==1
                  plot([x(i) x(j)],[y(i) y(j)],'k')
            end
      end
end

hold on
for i=1:n
      switch sum(c(i,:))
            case 0
                  plot(x(i),y(i),'.k','Markersize',15)
            case 3
                  plot(x(i),y(i),'.','Color',[1 0 0],'Markersize',15)
            otherwise
                  plot(x(i),y(i),'.','Color',c(i,:),'Markersize',15)
      end
end
end
```

References

[Dun08] P.E Dunne. An annotated list of selected np-complete problems, June 2008. Dept. of Computer Science, University of Liverpool, `http://www.csc.liv.ac.uk/ ped/teachadmin/COMP202/annotated`$_n p.html$.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.

[HA04] Didier Henrion and Denis Arzelier. Lagrangian and sdp duality, May 2004. Toulouse, France, *"Course On LMI Optimization with Applications in Control"*.

[HRVW96] Christoph Helmberg, Franz Rendl, Robert J. Vanderbei, and Henry Wolkowicz. An interior-point method for semidefinite programming. *SIAM J. Optim.*, 6(2):342–361, 1996.

[KMS98] David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *J. ACM*, 45(2):246–265, 1998.

[LR05] Monique Laurent and Franz Rendl. Semidefinite programming and integer programming. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*, chapter 8, pages 393–514. Elsevier B.V., 2005.

[Mon03] Renato D. C. Monteiro. First- and second-order methods for semidefinite programming. *Math. Program.*, 97(1-2, Ser. B):209–244, 2003. ISMP, 2003 (Copenhagen).

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

[PZ] Li Pingke and Liu Zhe. A repport on approximate graph coloring by semidefinite programming.

[Tod01] M. J. Todd. Semidefinite optimization. *Acta Numer.*, 10:515–560, 2001.

[Wig83] Avi Wigderson. Improving the performance guarantee for approximate graph coloring. *J. Assoc. Comput. Mach.*, 30(4):729–735, 1983.

[Zha98] Yin Zhang. On extending some primal-dual interior-point algorithms from linear programming to semidefinite programming. *SIAM J. Optim.*, 8(2):365–386 (electronic), 1998.