Lecture 20

The standard form of a semidefinite programming problem is

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{C} \bullet \boldsymbol{X} \\ \text{subject to} \quad & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i, \quad 1 \leq i \leq m, \\ & \boldsymbol{X} \succeq \boldsymbol{0}, \end{aligned} \tag{SDP-P}$$

where $\boldsymbol{C}$ and $\boldsymbol{A}_i$, $1 \leq i \leq m$, are symmetric $n \times n$ matrices (elements of the vector space SYM_n). At first sight, it is not clear why solving such problem should be of interest. In this lecture we discuss a useful application to *discrete* optimization problems that are usually difficult in practice.

20.1 Semidefinite relaxation

A graph is a pair $G = (V, E)$, where $V = \{1, \ldots, n\}$ consists of the *vertices*, and $E \subseteq V \times V$ consists of *edges* connecting some of the vertices. We will consider undirected edges, i.e., (i, j) will be the same edge as (j, i).

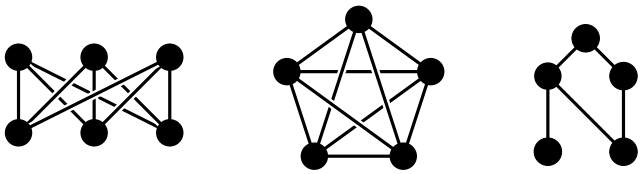


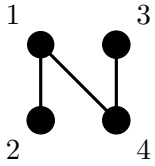
Figure 20.1: Some graphs

Note that for graphs, we only care about the incidences (which node is connected to which other node), the images are just visualizations and the distances and placements of nodes are pure convenience. Graphs are an important tool in areas such as network analysis, and many discrete computational problems can be formulated as graph problems. One such problem is MAXCUT. Given a graph $G = (V, E)$, a *cut* is a pair $(S, V \backslash S)$, where $S \subseteq V$ is a subset of vertices. The *edge set* of the cut is the set

$$E(S, V \backslash S) = \{e \in E : e \text{ connects a vertex in } S \text{ with a vertex in } V \backslash S\}.$$

The size of a cut is the number of edges in it, and the MAXCUT problem is the problem of finding a cut of maximal size.

Example 20.1. Consider the graph with $V = \{1, 2, 3, 4\}$ and $E = \{(1, 2), (3, 4), (1, 4)\}$. There are 7 nontrivial cuts, namely



$$\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}$$

Note that we don't list all the non-empty subsets of $\{1, 2, 3, 4\}$, as that would amount to counting every cut twice (every set would feature as S and as $V \backslash S$). The sizes of these cuts (number of edges joining S with $V \backslash S$) is

$$2, 1, 1, 2, 1, 3, 2.$$

The maximum cut size is therefore 3 (it can't be any bigger, since we only have three edges), and one such cut can be realized by taking the set $S = \{1, 3\}$ and $V \backslash S = \{2, 4\}$.

The problem MAXCUT (like many other graph problems) has the property that it is NP-hard. This means that the decision version of this problem (given a cut, does it have size at least k?) is NP-complete, which in turns means that checking the statement is easy (just count the number of edges in the cut), but *finding* a cut of size at least k appears to be difficult: there does not seem to be a way of doing this that is fundamentally faster than to test all the possible cuts. A famous conjecture, P$\neq$NP, implies that there does not exist an efficient (polynomial time) algorithm that would, in general, be able to find a cut of a given size, or even a maximal cut.

Two common approaches to deal with difficult problems are *approximation* and *randomization*. In approximation, one gives up on finding the best solution and concentrates on finding one that is good enough. In randomization, one takes into account that an algorithm may fail with small probability. Both approximation and randomization may lead to efficient algorithms that can solve a problem well enough for "all practical purposes".

In the case of MAXCUT, an approximation algorithm $\mathcal{A}$ takes as input a graph G and outputs a set $S \subseteq V$, written as $\mathcal{A}(G) = S$. If by $\omega(S)$ we denote the size of the cut induced by S, and by $\mathrm{Opt}(G)$ the largest size of a cut, then the algorithm $\mathcal{A}$ is said to provide an δ-approximation (for some $\delta > 0$), if

$$\omega(\mathcal{A}(G)) \geq \delta \cdot \mathrm{Opt}(G).$$

In words, the algorithms provides a cut that is optimal up to a factor δ. A randomized algorithm is one that is allowed to use internal coin flips. Such an algorithm is a δ-approximation algorithm, if the expected value

$$\mathrm{E}[\omega(\mathcal{A}(G))] \geq \delta \cdot \mathrm{Opt}(G).$$

We are interested in approximation algorithms that run in *polynomial time*. This means that the number of computational steps is a polynomial in the input size. Interior point methods for convex optimization and semidefinite programming are examples of polynomial time algorithms, and it is a remarkable fact that *MaxCut* can be solved approximatively using semidefinite programming.

20.2 The Goemans-Williamson Algorithm

The MAXCUT problem can be formulated as an integer programming problem as follows. Introduce variables $x_1, \ldots, x_n$, where n is the number of vertices. A cut is then an assignment $x_i = 1$ (meaning that $i \in S$) or $x_i = -1$ (meaning that $i \in V \backslash S$). If $e = (i, j)$ is an edge in the cut, then $x_i x_j = -1$, and $x_i x_j = 1$ otherwise. We therefore have

$$\frac{1}{2}(1 - x_i x_j) = \begin{cases} 1 & \text{if } (i,j) \in E(S, V\backslash S) \\ 0 & \text{if } (i,j) \notin E(S, V\backslash S). \end{cases}$$

The size of a cut is then the sum $\sum_{(i,j)\in E} \frac{1}{2}(1 - x_i x_j)$, and the problem of finding the maximum cut can be written as

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - x_i x_j) \\ \text{subject to} \quad & x_i^2 = 1,\ 1 \leq i \leq n. \end{aligned} \tag{20.1}$$

We now look for a *relaxation* of the problem: a new problem with a bigger constraint set that also contains the solutions of (20.1), but possibly more. As a first step, replace $x_i \in \{-1, 1\}$ with vectors $\boldsymbol{u}_i \in \{\pm \boldsymbol{e}_1\}$, where $\boldsymbol{e}_1 = (1, 0, \ldots, 0)^\top$ is a unit vector, and $x_i x_j$ with $\boldsymbol{u}_i^\top \boldsymbol{u}_j$. The optimization is then over sets of vectors $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$, and can be written as

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - \boldsymbol{u}_i^\top \boldsymbol{u}_j) \\ \text{subject to} \quad & \boldsymbol{u}_i \in \{\pm \boldsymbol{e}_1\}. \end{aligned}$$

The relaxation now consists in enlarging the constraint set to allow for *any* unit vectors $\boldsymbol{u}_i \in S^{n-1}$, where $S^{n-1} = \{\boldsymbol{x} : \|\boldsymbol{x}\| = 1\}$ is the unit sphere.

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - \boldsymbol{u}_i^\top \boldsymbol{u}_j) \\ \text{subject to} \quad & \|\boldsymbol{u}_i\| = 1. \end{aligned} \tag{20.2}$$

As the constraint set of the relaxation is bigger, the solution will be at least as large as the solution of (20.1) (ideally, the same, but we can't guarantee this).

For any set of unit vectors $\boldsymbol{u}_i$, let $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$ be the matrix with unit vectors $\boldsymbol{u}_i$ as columns, and consider

$$\boldsymbol{X} = \boldsymbol{U}^\top \boldsymbol{U}.$$

Then $x_{ii} = \boldsymbol{u}_i^\top \boldsymbol{u}_i = 1$ and the matrix $\boldsymbol{X}$ is symmetric and positive definite. Conversely, any symmetric positive definite matrix $\boldsymbol{X}$ can be written as $\boldsymbol{X} = \boldsymbol{U}^\top \boldsymbol{U}$ (Cholesky factorization), and if in addition $x_{ii} = 1$, then the columns of $\boldsymbol{U}$ necessarily have unit length. We conclude that Problem (20.2) is equivalent to the following SDP:

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{1}{2}(1 - x_{ij}) \\ \text{subject to} \quad & x_{ii} = 1 \\ & \boldsymbol{X} \succeq \boldsymbol{0}. \end{aligned} \tag{20.3}$$

Write $\mathrm{SDP}(G)$ for the optimal value of (20.3). From the above discussion we have

$$\mathrm{SDP}(G) \geq \mathrm{Opt}(G).$$

This is the case, because the solution of (20.1) is contained in the feasible set of (20.3), but the latter has more options.

To recover the cut, we proceed as follows.

1. Solve (20.3) to accuracy ε to obtain a matrix $\boldsymbol{X}^*$ such that

   $$\sum_{(i,j)\in E} \frac{1}{2}(1 - x_{ij}^*) \geq \mathrm{SDP}(G) - \varepsilon.$$

   This can be done in polynomial time using, for example, interior point methods.

2. Perform a Cholesky factorization $\boldsymbol{X}^* = (\boldsymbol{U}^*)^\top \boldsymbol{U}^*$ and extract the columns $\boldsymbol{u}_1^*, \ldots, \boldsymbol{u}_n^*$, which are unit vectors and satisfy

   $$\sum_{(i,j)\in E} \frac{1}{2}(1 - (\boldsymbol{u}_i^*)^\top \boldsymbol{u}_j^*) \geq \mathrm{Opt}(G) - \varepsilon.$$

3. Finally, we want a way to recover a partition of the graph without loosing too much. We thus need to assign to each $\boldsymbol{u}_i$ a value $x_i \in \{-1, 1\}$ according so some rule, and declare the result to be our partition. To do so, we *randomly* choose a vector $\boldsymbol{p} \in S^{n-1}$ and then set

   $$x_i = \begin{cases} 1 & \text{if } \boldsymbol{u}_i^\top \boldsymbol{p} \geq 0, \\ -1 & \text{else.} \end{cases}$$

There we have our algorithm: it generates a cut. Let's call the above algorithm, which takes a graph G and produces a set S by the above steps, $\boldsymbol{A}$.

Theorem 20.2. *The Goemans-Williamson algorithms generates a cut $\mathcal{A}(G) = S$ and satisfies*

$$\mathrm{E}[\omega(\mathcal{A}(G))] \geq 0.8785672 \cdot \mathrm{Opt}(G).$$

That is, the algorithm is expected to generate a cut that is at least as 0.8785672 times as good as the optimal cut (that is, could be about 10 percent smaller but not more). By the law of large numbers, repeating the algorithm will likely give a result close to the expectation.