3 Randomized Rounding

In this section we will introduce various randomized algorithms that can find good solutions to combinatorial optimization problems. First, we define some important classes of optimization problems, and then we study various randomized rounding techniques for special problems like the MAXSAT problem, the MINCUT problem, the SETCOVER problem, and finally, the *MaxCUT* problem. The randomized rounding idea goes back to an influential work by Raghavan and Thomson [RT87] and is nowadays a standard technique in the area of approximation algorithms.

3.1 Optimization problems

In general, optimization problems are defined as follows. Consider a set S of feasible solutions and an ordered set $(T, \leq)$ of evaluations. Moreover, consider the mapping $f : S \to T$ (the objective function). We are looking for an element $x^* \in S$ with the property that $f(x^*) \geq f(x)$ for all $s \in S$ (maximization problem) or $f(x^*) \leq f(x)$ for all $x \in S$ (minimization problem). It is common to use the following notation:

$$\begin{array}{lll} \max_{x \in S} f(x) & \text{or} & \max\{f(x) \mid x \in S\} \\ \min_{x \in S} f(x) & \text{or} & \min\{f(x) \mid x \in S\} \end{array}$$

In practice, mostly either $\mathbb{R}$ (the set of real numbers), $\mathbb{Q}$ (the set of rational numbers), or $\mathbb{Z}$ (the set of integers) are used for $(T, \leq)$. The formulation above, however, is too general. If S is just given by a list of all of its elements, this problem is either useless or trivial (one simply computes $f(x)$ for all $x \in S$). Thus, S should instead be specified by certain constraints that uniquely determine S without the need to know each of its elements. Also, the function f should have a compact representation. The optimization problems that we are interested in in this section are defined as follows:

- **Linear optimization problem (Linear Program, or short LP):**
  Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{(m,n)}$, and $b \in \mathbb{R}^m$. Then the optimization problem with $S = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ and $f(x) = c^T x$ is called a *linear optimization problem*.
- **Integer linear optimization problem (Integer Linear Program, or short ILP):**
  Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{(m,n)}$, and $b \in \mathbb{R}^m$. Then the optimization problem with $S = \{x \in \mathbb{Z}^n \mid Ax \leq b\}$ and $f(x) = c^T x$ is called an *integer linear optimization problem.*

Integer optimization problems are also called combinatorial optimization problems. Combinatorial problems are usually hard to solve while for linear optimization problems there are efficient polynomial-time algorithms (i.e., the methods by Karmarkar and Kachiyan). One approach is therefore to convert a combinatorial optimization problem into a linear optimization problem, to optimally solve that, and then convert this solution back into an integer solution. A standard technique for the conversion into an integer solution is *randomized rounding*, which will be illustrated in this section by considering various examples.

3.2 The MAXSAT Problem

Let $V = \{x_1, \ldots, x_n\}$ be a set of Boolean variables, i.e., variables that can only take the values TRUE and FALSE. A *literal* is a variable $x_i \in V$ or its negation $\bar{x}_i$. A *clause* $C = \ell_1 \vee \ldots \vee \ell_k$ is a disjunction of literals. A Boolean formula (resp. expression) Φ is in *conjunctive normal form (CNF)* if it is a conjunction of clauses, i.e., $\Phi = C_1 \wedge \ldots \wedge C_m$ for some clauses C_j. We also write in this section $C_j \in \Phi$. For example, the Boolean formula

$$\Phi(x_1, x_2, x_3) = \bar{x}_1 \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_3) \wedge (x_2 \vee x_3)$$

is in CNF.

In the MAXSAT problem we are given a Boolean formula Φ in CNF. The set S of all feasible solutions of Φ is given by the set of all truth assignments $b : V \to \{\text{TRUE}, \text{FALSE}\}$ of the Boolean variables in Φ. Given a truth assignment, a Boolean expression is recursively evaluated in the following way:

Φ_1	$\neg\Phi_1$
TRUE	FALSE
FALSE	TRUE

Φ_1	Φ_2	$\Phi_1 \vee \Phi_2$
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Φ_1	Φ_2	$\Phi_1 \wedge \Phi_2$
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

We say that a Boolean expression Φ is *satisfiable* if there is a truth assignment b that satisfies Φ (i.e., $\Phi(b) = \text{TRUE}$).
The objective function is given by

$$f_\Phi(b) = |\{C_j \in \Phi \mid C_j(b) = \text{TRUE}\}|$$

Thus, $f_\Phi(b)$ is the number of clauses in Φ that are satisfied by b. We are looking for the b^* that maximizes $f_\Phi(b^*)$.

It is straightforward to formulate the MaxSAT problem as an ILP. We replace the domain of each Boolean variable x_i by $\{0, 1\}$ where 0 means FALSE and 1 means TRUE. For each clause C_j we introduce a binary variable z_j that is 1 if and only if C_j is satisfied. In this case, the number of satisfied clauses is given by $\sum_{j=1}^{m} z_j$. In order to set z_j to 1, at least one of the literals in $C_j = \ell_1 \vee \ldots \vee \ell_k$ must be true. Let V_j^+ the set of non-negated variables in C_j and V_j^- be the set of negated variables in C_j. Then z_j can be set to 1 if $\sum_{x_i \in V_j^+} x_i + \sum_{x_i \in V_j^-} (1 - x_i) \geq 1$. Thus, we can formulate the following ILP:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} z_j \\ \text{subject to} \quad & \sum_{x_i \in V_j^+} x_i + \sum_{x_i \in V_j^-} (1 - x_i) \geq z_j \quad \text{for all } j \\ & x_i, z_j \in \{0, 1\} \quad \text{for all } i, j \end{aligned} \tag{1}$$

Here, $x_i, z_j \in \{0, 1\}$ is a short form of the constraints that $x_i \geq 0$, $x_i \leq 1$, $z_j \geq 0$, $z_j \leq 1$, and $x_i, z_j \in \mathbb{Z}$ for all i, j.

An optimal solution to this ILP is obviously a maximum satisfying assignment and therefore not easier to compute than an optimal solution to the original problem. However, when replacing the conditions in (1) by

$$0 \leq \hat{x}_i, \hat{Z}_j \leq 1 \qquad \text{for all } i, j$$

then the resulting optimization problem is an LP and therefore can be solved optimally in time polynomial in the size of the LP [Kar91]. Since the size of the LP is polynomial in the size of Φ, the polynomial time bound also applies to the original instance Φ.

The approach of exchanging a restrictive by a more general condition and thereby obtaining a simpler problem is called *relaxation*. Note that a relaxation usually changes the set of feasible solutions. Hence, after obtaining a feasible solution for the relaxation, one has to use a mechanism to convert it into a solution that is also feasible for the original problem. In the following we assume that the x_i^*, Z_j^* represent the values of an optimal solution for our LP. Since an optimal assignment for Φ fulfills the conditions of the LP, it holds:

$$\sum_{j=1}^{m} Z_j^* \geq \text{OPT}(\Phi) \; .$$

This is the relationship we intend to use when comparing the quality of our solution with the quality of a best possible solution for MAXSAT. It is frequently used when applying linear optimization to approximation algorithms and is called the *superoptimality of the linear relaxation*.

How do we arrive now from a real-valued solution to a feasible integral solution? Simply by using the following randomized rounding approach.

RANDOMIZEDROUNDING(π):

for $i = 1$ to n do

$$\text{with probability} \begin{cases} \pi_i : & x_i = 1 \\ 1 - \pi_i : & x_i = 0 \end{cases}$$

The randomized rounding approach can now be used to construct the following algorithm for the MAXSAT problem.

1. determine an optimal solution x^* to the linear relaxation for Φ
2. set $\pi_i = x_i^*$ for every i $\qquad (*)$
3. use RANDOMIZEDROUNDING(π) to obtain a feasible solution for the ILP

Intuitively, the randomized rounding approach tries to select a feasible solution that is as close to x^* as possible, hoping that this will produce a feasible solution whose value is close to the value of x^*. In fact, when viewing TRUE $= 1$ and FALSE $= 0$, we get

$$\mathbb{E}[x_i] = \Pr[x_i = 1] = x_i^*$$

and therefore $\mathbb{E}[\sum_i x_i] = \sum_i x_i^*$.

If we assume that the x_i's are chosen independently at random in RANDOMIZEDROUNDING, we get the following relationship between $T_B(C_j)$ and Z_j^*.

Lemma 3.1 *Let k_j be the number of literals in C_j. Then it holds:*

$$\Pr[x \textit{ satisfies } C_j] \geq \left(1 - \left(1 - \frac{1}{k_j}\right)^{k_j}\right) \cdot Z_j^* \ .$$

Proof. We need two well-known facts.

Fact 3.2

(a) If $a_i \geq 0$ for all i, then $(\prod_{i=1}^k a_i)^{1/k} \leq \frac{1}{k} \sum_{i=1}^k a_i$, i.e. the geometric mean is at most as large as the arithmetic mean.

(b) If the function $f(x)$ is concave (i.e. $f''(x) \leq 0$) in the interval $[a, b]$ and $f(a) \geq c \cdot a + d$ and $f(b) \geq c \cdot b + d$, then $f(x) \geq c \cdot x + d$ for all $x \in [a, b]$.

C_j can be written as

$$C_j = \left(\bigvee_{x_i \in V_j^+} x_i\right) \vee \left(\bigvee_{x_i \in V_j^-} \bar{x}_i\right) \ .$$

Hence, it follows from the linear program formulation that

$$Z_j^* \leq \sum_{x_i \in V_j^+} x_i^* + \sum_{x_i \in V_j^-} (1 - x_i^*) \ . \tag{2}$$

Obviously, C_j is false if and only if all $x_i \in V_j^+$ are set to 0 (representing FALSE) and all $x_i \in V_j^-$ are set to 1 (representing TRUE). This happens with probability

$$\left[\prod_{x_i \in V_j^+} (1 - x_i^*)\right] \cdot \left[\prod_{x_i \in V_j^-} x_i^*\right] \ .$$

Here it is important that the x_i's are chosen independently at random. The product consists of altogether k_j factors. Thus,

$$\begin{aligned}
\Pr[x \text{ satisfies } C_j] &= 1 - \left[\prod_{x_i \in V_j^+} (1 - x_i^*)\right] \cdot \left[\prod_{x_i \in V_j^-} x_i^*\right] \\
&\overset{\text{Fact 3.2(a)}}{\geq} 1 - \left(\frac{\sum_{x_i \in V_j^+}(1 - x_i^*) + \sum_{x_i \in V_j^-} x_i^*}{k_j}\right)^{k_j} \\
&= 1 - \left(\frac{|V_j^+| - \sum_{x_i \in V_j^+} x_i^* + |V_j^-| - \sum_{x_i \in V_j^-}(1 - x_i^*)}{k_j}\right)^{k_j} \\
&= 1 - \left(\frac{k_j - \left(\sum_{x_i \in V_j^+} x_i^* + \sum_{x_i \in V_j^-}(1 - x_i^*)\right)}{k_j}\right)^{k_j} \\
&\overset{(2)}{\geq} 1 - \left(1 - \frac{Z_j^*}{k_j}\right)^{k_j} \\
&\overset{\text{Fact 3.2(b)}}{\geq} \left(1 - \left(1 - \frac{1}{k_j}\right)^{k_j}\right) \cdot Z_j^*
\end{aligned}$$

□

Theorem 3.3 *For every Boolean expression Φ in CNF in which there are at most k literals in every clause,*

$$\mathbb{E}[f_\phi(x)] \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \mathrm{OPT}(\Phi) \; .$$

Proof. Let $\Phi = C_1 \wedge \ldots \wedge C_m$ and let k be the length of the longest clause.

$$\begin{aligned}
\mathbb{E}[f_\phi(x)] &= \sum_{j=1}^{m} \Pr[x \text{ satisfies } C_j] \\
&\geq \sum_{j=1}^{m} \left(1 - \left(1 - \frac{1}{k_j}\right)^{k_j}\right) \cdot Z_j^* \\
&\overset{(a)}{\geq} \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \sum_{j=1}^{m} Z_j^* \\
&\overset{(b)}{\geq} \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \mathrm{OPT}(\Phi)
\end{aligned}$$

We used in (a) that $1 - (1 - 1/z)^z$ is monotonically decreasing for $z > 0$ and in (b) the superoptimality of the linear relaxation. □

Since $1 - (1 - 1/k)^k \geq 1 - 1/e$ for all $k \in \mathbb{N}$, we obtain:

Theorem 3.4 *The expected number of satisfied clauses achieved on input Φ is at least $(1 - 1/e) \cdot \mathrm{OPT}(\Phi) \approx 0.632 \cdot \mathrm{OPT}(\Phi)$.*

We used for the randomized rounding the assignment $\pi_i = x_i^*$ for all i (see $(*)$). For other assignments like $\pi_i = \frac{1}{2}x_i^* + \frac{1}{4}$ one can even show that the expected objective value is at least $(3/4) \cdot \mathrm{OPT}$. The same holds for arbitrary assignments for π with $1 - 1/4^{x_i^*} \leq \pi_i \leq 4^{x_i^* - 1}$ for all i. This approach is called *non-linear randomized rounding*.

The best known approximation algorithm for MAXSAT with a polynomial runtime achieves an expected objective value of at least $0.833 \cdot \mathrm{OPT}$ [AW00]. It is a combination of various optimization techniques including semidefinite optimization, which is discussed later in this section.

3.3 The MINCUT Problem

In the MINCUT problem we are given an undirected graph $G = (V, E)$ with special nodes $s, t \in V$ and edge costs $c : E \to \mathbb{R}_+$. A partition $(U, \bar{U})$ of V is called an (s,t)-*cut* if $s \in U$ and $t \in \bar{U} = V \setminus U$. The *cost* of a cut $(U, \bar{U})$ is defined as

$$c(U, \bar{U}) = \sum_{\{v,w\} \in E:\ v \in U, w \in \bar{U}} c(v, w)$$

The set S of feasible solutions is the set of all (s,t)-cuts, and the objective function is defined as $f(U, \bar{U}) = c(U, \bar{U})$. Our goal is to find an (s,t)-cut with minimal cost.

The MINCUT problem can be formulated as an ILP. For every node i let the binary variable x_i be 0 if $i \in U$ and 1 otherwise, and for each edge $\{i, j\}$ let the binary variable $z_{i,j}$ be 1 if the edge $\{i, j\}$ crosses the cut $(U, \bar{U})$ and 0 otherwise. Then the goal is to minimize $\sum_{\{i,j\} \in E} c_{i,j} z_{i,j}$. Formally, the ILP looks as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{\{i,j\} \in E} c_{i,j} z_{i,j} \\
\text{subject to} \quad & x_i - x_j \le z_{i,j} \text{ and } x_j - x_i \le z_{i,j} \text{ for all } \{i, j\} \in E \\
& x_s = 0 \text{ and } x_t = 1 \\
& x_i, z_{i,j} \in \{0, 1\} \quad \text{for all } i, j
\end{aligned}$$

If we replace the condition $x_i, z_{i,j} \in \{0, 1\}$ by $0 \le x_i, z_{i,j} \le 1$, we obtain a linear program. For an optimal solution of the ILP and its linear relaxation it has to hold that $z_{i,j} = |x_i - x_j|$. Let x_i^* and $z_{i,j}^*$ be the values of an optimal solution of the linear relaxation, and let the objective value of that solution be y^*. Certainly, the optimal value OPT of the integer program must satisfy $y^* \le \text{OPT}$. We will now prove via randomized rounding that $y^* = \text{OPT}$.

Choose a random $u \in [0, 1)$. For every $i \in V$ we set $x_i = 0$ if $x_i^* \le u$ and otherwise $x_i = 1$. Consider now a fixed edge $\{i, j\} \in E$. The edge crosses the cut if and only if $u \in [\min\{x_i^*, x_j^*\}, \max\{x_i^*, x_j^*\})$, which happens with probability $|x_i^* - x_j^*|$, i.e., $z_{i,j}^*$. Hence, $\mathbb{E}[z_{i,j}] = z_{i,j}^*$ and therefore,

$$\mathbb{E}[\sum_{\{i,j\} \in E} c_{i,j} z_{i,j}] = \sum_{\{i,j\} \in E} c_{i,j} \mathbb{E}[z_{i,j}] = \sum_{\{i,j\} \in E} c_{i,j} z_{i,j}^* = y^*$$

Thus, there must be a u with $\sum_{\{i,j\} \in E} c_{i,j} z_{i,j} \le y^*$, which implies that $y^* = \text{OPT}$. Moreover, since there cannot be a u with $\sum_{\{i,j\} \in E} c_{i,j} z_{i,j} < y^*$, the rounding for *all* u leads to an optimal value. See, for example, [TS97] for more applications of this idea.

3.4 The SETCOVER Problem

In the SETCOVER problem we are given a set $S_1, \ldots, S_m$ of subsets of $\{1, \ldots, n\}$ with costs $c_1, \ldots, c_m \in \mathbb{R}_+$. The set S of feasible solutions is the set of all subsets $C \subseteq \{1, \ldots, m\}$ with the property that for every element $j \in \{1, \ldots, n\}$ there is at least one $i \in C$ with $j \in S_i$, i.e., $S(C) = \{S_i \mid i \in C\}$ is a *(set) cover* of $\{1, \ldots, n\}$. The goal is to minimize $f(C) = \sum_{i \in C} c_i$.

Also the SETCOVER problem can be expressed as an integer linear program. Let the binary random variables $x_1, \ldots, x_m$ have the property that $x_i = 1$ if and only if $i \in C$. Using these variables, we get

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} c_i x_i \\
\text{subject to} \quad & \sum_{i: j \in S_i} x_i \ge 1 \quad \text{for all } j \\
& x_i \in \{0, 1\} \quad \text{for all } i
\end{aligned}$$

This can be turned into a linear program if we replace $x_i \in \{0, 1\}$ by $0 \le x_i \le 1$. Let $(x_1^*, \ldots, x_m^*)$ be an optimal solution of this linear relaxation. We use the following rounding strategy in order to turn this solution into an integer solution.

$(y_1, \ldots, y_m) := (0, \ldots, 0)$
for $r := 1$ **to** t **do**
    $(x_1, \ldots, x_m) :=$ RANDOMIZEDROUNDING($\pi(x) = x^*$)
    $(y_1, \ldots, y_m) := (y_1 \vee x_1, \ldots, y_m \vee x_m)$
output $(y_1, \ldots, y_m)$

The solution obtained by that algorithm may not be feasible, but we will see that for a sufficiently large t it is feasible with high probability. For this we need the following lemma.

Lemma 3.5 *The probability that some fixed element $j \in \{1, \ldots, n\}$ is not covered after t rounds is at most $(1/e)^t$.*

Proof. Consider an arbitrary fixed element j. Let $C_r = \{i \in \{1, \ldots, m\} \mid x_i = 1\}$ be the subset that is randomly generated in round r. Then

$$\Pr[j \notin \bigcup_{i \in C_r} S_i] = \prod_{i:j \in S_i} (1 - x_i^*) \leq \prod_{i:j \in S_i} e^{-x_i^*} = e^{-\sum_{i:j \in S_i} x_i^*}$$

since it holds that $1 - x \leq e^{-x}$ for all $x \in \mathbb{R}$. From the constraints of the LP we know that $\sum_{i:j \in S_i} x_i^* \geq 1$. Hence,

$$\Pr[j \notin \bigcup_{i \in C_r} S_i] \leq 1/e$$

When using t independent randomized roundings, the probability that j is not covered by any of the subsets C_r is at most $(1/e)^t$. □

Let C_r be the subset chosen in round r and let $C = \bigcup_{r=1}^{t} C_r$. Then it holds for $t = \ln(4n)$:

$$\begin{aligned} \Pr[S(C) \text{ is not a cover}] &\leq \sum_{j=1}^{n} \Pr[S(C) \text{ does not cover } j] \\ &\leq \sum_{j=1}^{n} \left(\frac{1}{e}\right)^t = n \left(\frac{1}{e}\right)^{\ln(4n)} = n \cdot \frac{1}{4n} = \frac{1}{4} \end{aligned}$$

The expected cost for every C_r is equal to $\sum_{i=1}^{m} c_i x_i^*$ and therefore equal to the optimal cost of the linear program, which we denote by y^*. The expected cost of C is therefore $y^* \ln(4n)$. According to the Markov inequality it holds for any non-negative random variable X that

$$\Pr[X \geq k \cdot \mathbb{E}[X]] \leq \frac{1}{k}$$

Thus, it holds that

$$\Pr[cost(C) \geq 4y^* \ln(4n)] \leq \frac{1}{4}$$

Thus, the probability that our algorithm produces a feasible solution with an *approximation ratio* of less than $4 \ln(4n)$ (i.e., the objective value is less than a factor of $4 \ln(4n)$ away from the optimal solution)) is equal to

$$\begin{aligned} 1 - \Pr[S(C) \text{ is not a cover or } cost(C) \geq 4y^* \ln(4n)] &\geq 1 - (\Pr[S(C) \text{ is not a cover}] + \Pr[cost(C) \geq 4y^* \ln(4n)]) \\ &\geq 1 - (1/4 + 1/4) = 1/2 \end{aligned}$$

On expectation, it must therefore be executed at most twice to obtain a feasible solution.

The best known approximation algorithm for the SETCOVER problem is, surprisingly, a simple deterministic greedy algorithm, which guarantees an approximation ratio of $\ln n + 1$. On the other hand, Feige [Fei98] has shown that no polynomial time algorithm can have a better approximation ratio than $(1 - o(1)) \ln n$ (unless some complexity classes collapse), so complexity of solving the SETCOVER problem is well-understood.

3.5 Semidefinite optimization

In this subsection we will continue to refine our randomized rounding techniques by using semidefinite optimization. An excellent overview of applications for semidefinite optimization can be found in [WSV00].

3.5.1 Definitions and facts

We start with a short introduction to linear algebra. A sequence $A = (\alpha_{i,j})$ with entries $\alpha_{i,j} \in \mathbb{R}$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$ is called a *matrix* with n columns and m rows and written as

$$A = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & & \vdots \\ \alpha_{m,1} & \alpha_{m,2} & \cdots & \alpha_{m,n} \end{pmatrix}$$

If $n = 1$, A is simply called a *vector*. Let $M(m, n; \mathbb{R})$ be the space of all real-valued matrices with m rows and n columns. Basic operations on matrices are performed as follows:

- For any two matrices $A, B \in M(m, n; \mathbb{R})$:

$$A + B = (\alpha_{i,j}) + (\beta_{i,j}) = (\alpha_{i,j} + \beta_{i,j}) \in M(m, n; \mathbb{R}) \ .$$

- For any $\gamma \in \mathbb{R}$ and any matrix $A \in M(m, n; \mathbb{R})$:

$$\gamma \cdot A = \gamma \cdot (\alpha_{i,j}) = (\gamma \cdot \alpha_{i,j}) \in M(m, n; \mathbb{R}) \ .$$

- For any two matrices $A \in M(m, n; \mathbb{R})$ and $B = M(n, p; \mathbb{R})$:

$$A \cdot B = (\alpha_{i,j}) \cdot (\beta_{i,j}) = (\gamma_{i,j}) \in M(m, p; \mathbb{R})$$

  with $\gamma_{i,j} = \sum_{k=1}^{n} \alpha_{i,k} \cdot \beta_{k,j}$ for all i, j.

The neutral element concerning addition is $0_{m,n} \in M(m, n; \mathbb{R})$, and the neutral element concerning multiplication is $E_n \in M(n, n; \mathbb{R})$, where

$$O_{m,n} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \quad \text{and} \quad E_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

That is, for any matrix $A \in M(m, n; \mathbb{R})$, $A + 0_{m,n} = A$ and $A \cdot E_n = E_m \cdot A = A$. The *transpose* of a matrix $A = (\alpha_{i,j}) \in M(m, n; \mathbb{R})$ is defined as

$$A^T = \begin{pmatrix} \alpha_{1,1} & \alpha_{2,1} & \cdots & \alpha_{m,1} \\ \alpha_{1,2} & \alpha_{2,2} & \cdots & \alpha_{m,2} \\ \vdots & \vdots & & \vdots \\ \alpha_{1,n} & \alpha_{2,n} & \cdots & \alpha_{m,n} \end{pmatrix}$$

A vector $a = (\alpha_i) \in \mathbb{R}^m$ is called a *unit vector* if it is of unit length or

$$\sqrt{\sum_{i=1}^{m} \alpha_i^2} = 1 \ .$$

A matrix $A \in M(n, n; \mathbb{R})$ is called *positive semidefinite* if for all $b \in \mathbb{R}^n$, $b^T \cdot A \cdot b \geq 0$. A is *symmetric and positive semidefinite* if there is a matrix $B \in M(m, n; \mathbb{R})$ with $B^T \cdot B = A$.

Definition 3.6 A *semidefinite program* is a linear program in which we demand that the solution for the variables can be represented as a symmetric and positive semidefinite matrix. That is, a semidefinite program looks as follows:

Semidefinite program (SDP):

$$\begin{aligned} \text{optimize} \quad & \sum_{i,j} c_{i,j} \cdot x_{i,j} \\ \text{subject to} \quad & \sum_{i,j} a_{(i,j),k} \cdot x_{i,j} \leq b_k \quad \text{for all } k \\ & X = (x_{i,j}) \text{ is symmetric and positive semidefinite} \end{aligned}$$

$\mathrm{OPT}(SDP)$ represents the optimal value achievable for the given SDP.

Fact 3.7

(a) If A is symmetric and positive semidefinite, then the matrix B with the property that $B^T \cdot B = A$ can be computed in time $O(n^3)$ with the help of a so-called Cholesky decomposition.

(b) If all diagonal entries in a symmetic and positive semidefinite matrix A are equal to 1, then each column of the corresponding matrix B is a unit vector in $\mathbb{R}^m$.

(c) Semidefinite optimization can be performed in time $O(poly(n, m, \log(1/\epsilon)))$ with an absolute error of at most ϵ for any $\epsilon > 0$. That is, a symmetric and positive semidefinite matrix $X^{(\epsilon)} = (x_{i,j}^{(\epsilon)})$ is computed whose entries fulfill the restrictions and

$$|\mathrm{OPT}(SDP) - \sum_{i,j} c_{i,j} \cdot x_{i,j}^{(\epsilon)}| \leq \epsilon \; .$$

Example 3.8

(a) The matrix $A_1 = \begin{pmatrix} 1 & 4 \\ 4 & 1 \end{pmatrix}$ is not positive semidefinite, since $(1, -1) \cdot A_1 \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = -6$.

(b) The matrix $A_2 = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix}$ is positive semidefinite, since $A_2 = B^T \cdot B$ for $B = \begin{pmatrix} 1 & -1 \\ 0 & \sqrt{3} \end{pmatrix}$.

3.6 The MAXCUT problem

Semidefinite optimization was used with great success for the design of an approximation algorithm for the problem of determining a maximum cut in a graph. The algorithm presented in this section is the best algorithm known so far for this problem.

Definition 3.9 An instance of the MAXCUT problem is an undirected, connected graph $G = (V, E)$. A feasible solution for G is a cut $(U, \bar{U})$, where

$$(U, \bar{U}) = \{\{v, w\} \in E \mid v \in U \text{ and } w \in \bar{U}\}$$

for some subset $U \subseteq V$. The objective is to maximize $|(U, \bar{U})|$, i.e., the number of edges crossing that cut.

The decision variant of the MAXCUT problem is **NP**-complete [GJ79, p. 210]. Similar to the MAXSAT problem, every graph has a relatively large cut.

Theorem 3.10 *Let $G = (V, E)$ be a connected graph. Then G has a cut of size at least $|E|/2$.*

The proof can be done with the help of a probabilistic method (see Theorem 1.6). Hence, it immediately implies a randomized approximation algorithm with an expected approximation ratio of 2 that runs in time $O(|V| + |E|)$. Alternatively, the theorem could also be proved with the help of a simple, deterministic greedy algorithm.

For a long time, an approximation ratio of 2 was the best result known for MAXCUT. It took until 1994 before Goemans and Williamson came up with a better approximation ratio that is based on semidefinite optimization [GW95]. We will present their method in the following.

First, we show how to transform the MAXCUT problem into a so-called quadratic optimization problem. Let $n = |V|$ and $A_G = (a_{i,j}) \in M(n, n; \mathbb{R})$ be the adjacency matrix of the given graph $G = (V, E)$, i.e. for all nodes $i, j \in V$,

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

In order to perform an arithmetization of the problem, we select a variable x_i for each node i that can take the values -1 and $+1$. $x_i = -1$ means that $i \in U$, and $x_i = +1$ means that $i \in \bar{U}$. For an edge $\{i,j\} \in E$ consider the value $\frac{1}{2}(1 - x_i x_j)$. If the edge is crossing the cut, then its value is 1 and otherwise 0. Hence,

$$|(U, \bar{U})| = \sum_{\{i,j\} \in E} \tfrac{1}{2}(1 - x_i x_j) = \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - x_i x_j) \; .$$

($\sum_{i<j}$ is an abbreviation of $\sum_{i=1}^{n} \sum_{j=i+1}^{n}$.) Now we can state the following optimization problem:

Quadratic program for MAXCUT:

$$\begin{aligned} \text{maximize} \quad & \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - x_i x_j) && (3) \\ \text{subject to} \quad & x_i \in \{-1, +1\} \quad \text{for all } i && (4) \end{aligned}$$

Since in the objective function (3) two variables are multiplied with each other, we talk about a *quadratic* optimization problem. Due to condition (4) it is hard to solve. However, just relaxing (4) to $-1 \leq x_i \leq 1$ does not help much, because we still have a quadratic objective function. Instead, our aim will be to convert this program into a semidefinite program. This will be done in several stages.

First, we replace every variable x_i by an n-dimensional vector $\vec{x}_i = (x_i, 0, \ldots, 0)^T$ of length 1 (recall that $n = |V|$). This does not change anything in the objective function, since $\vec{x}_i^{\,T} \cdot \vec{x}_j = x_i \cdot x_j$. We relax now these vectors in a form that we allow all places in them to have values different from 0, as long as they are still of unit length. We call these vectors $\vec{u}_i$. It holds that $\vec{u}_i^{\,T} \cdot \vec{u}_i = 1$. This results in the following relaxation:

Relaxed quadratic program for MAXCUT:

$$\begin{aligned} \text{maximize} \quad & \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - \vec{u}_i^{\,T} \vec{u}_j) && (5) \\ \text{subject to} \quad & \vec{u}_i^{\,T} \cdot \vec{u}_i = 1 \quad \text{for all } i && (6) \end{aligned}$$

Now we are ready to transform the program into a semidefinite program. We introduce n^2 new variables $y_{i,j}$ with $y_{i,j} = \vec{u}_i^{\,T} \cdot \vec{u}_j$. These variables can be written as a matrix $Y = (y_{i,j}) \in M(n, n; \mathbb{R})$. Since the $\vec{u}_i$ have a length of 1, $y_{i,i} = 1$ for all i. Let $B = (\vec{u}_1 \; \vec{u}_2 \; \cdots \; \vec{u}_n)$ whose columns are the vectors $\vec{u}_i$. It is easy to check that $Y = B^T \cdot B$. Thus, we know that Y is symmetric and positive semidefinite. Thus, we can formulate now the semidefinite program corresponding to the MAXCUT problem:

Semidefinite program "SD-CUT" for MAXCUT:

$$\begin{aligned} \text{maximize} \quad & \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - y_{i,j}) && (7) \\ \text{subject to} \quad & Y = (y_{i,j}) \text{ is symmetric and positive semidefinite} && (8) \\ & y_{i,i} = 1 \quad \text{for all } i && (9) \end{aligned}$$

According to Fact 3.7(c), this semidefinite optimization problem can be solved with an absolute error of ϵ (we will determine the ϵ later). Hence, $y_{i,j}^{(\epsilon)}$ can be computed with

$$0 \leq \mathrm{OPT}(\text{SD-CUT}) - \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - y_{i,j}^{(\epsilon)}) \leq \epsilon \; .$$

Since $\mathrm{OPT}(G) \leq OPT(\text{SD-CUT})$, it follows that

$$\tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - y_{i,j}^{(\epsilon)}) \geq \mathrm{OPT}(G) - \epsilon \; . \tag{10}$$

This bound will be useful later.

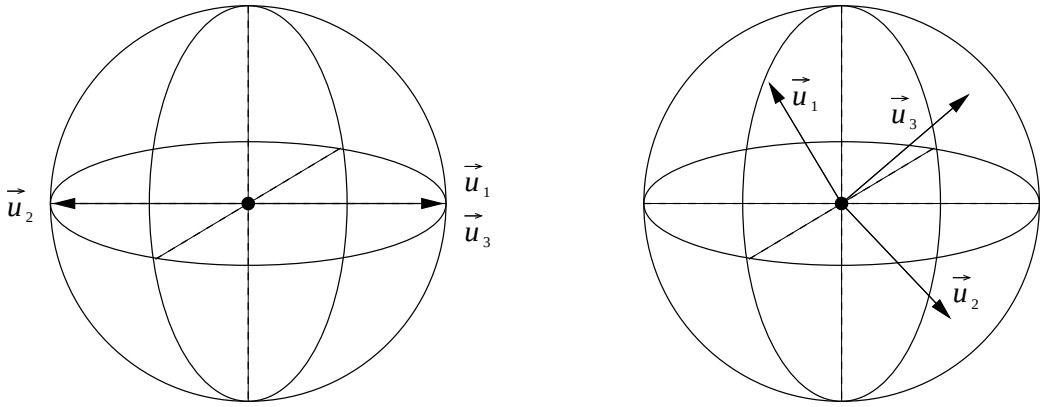

Figure 1: Possible vectors in the unit ball: (a) in the quadratic program and (b) in the semidefinite program

From the obtained symmetric and positive semidefinite matrix $Y^{(\epsilon)}$ we can get via Cholesky decomposition the matrix B and therefore the vectors $\vec{u}_i$. Due to Fact 3.7(b), these vectors are unit vectors. It remains to transform the $\vec{u}_i$'s into the original x_i's. We will do this with the help of a randomized method.

Figure 1(a) shows the situation we would have had, had we been able to obtain an optimal solution for the original quadratic program. The vectors have the values $\vec{u}_1 = (-1, 0, 0)^T$, $\vec{u}_2 = (1, 0, 0)$, and $\vec{u}_3 = (1, 0, 0)$.

Figure 1(b) shows the situation for the optimal solution obtained for the semidefinite program. It may consist of vectors whose endpoints are somewhere on the surface of the ball. The problem is to divide these vectors into two sets (one for the value -1, and one for the value $+1$ in the original problem). We do this with the help of a random hyperplane through the origin of the ball. All vectors that lie on one side of the plane will be mapped to -1, and the vectors that lie on the other side will be mapped to $+1$.

Now we have to determine how to select the random hyperplane. Since we require it to go through the origin of the ball, it will be completely determined by selecting a *normal vector* $\vec{r}$ for the hyperplane, i.e. a vector that is orthogonal to all vectors aligned with the hyperplane. From analytical geometry it is known that $\vec{u}_i$ and $\vec{u}_j$ lie on opposite sides of the hyperplane if $\operatorname{sgn}(\vec{r}^T \cdot \vec{u}_i) \neq \operatorname{sgn}(\vec{r}^T \cdot \vec{u}_j)$. Hence, the complete algorithm looks as follows:

Algorithm SDCUT:

1. $\epsilon = 0.0005$
2. solve the semidefinite program for MAXCUT with absolute error ϵ
3. compute the matrix B with $Y^{(\epsilon)} = B^T \cdot B$ via Cholesky decomposition
   (this will give the vectors $\vec{u}_i$)
4. select uniformly at random an n-dimensional unit vector $\vec{r}$
5. for $i = 1$ to n do
   if $\vec{r}^T \cdot \vec{u}_i \geq 0$
   then $x_i = +1$ (i.e. add node i to $V_\oplus$)
   else $x_i = -1$ (i.e. add node i to $V_\ominus$)
6. output $(V_\ominus, V_\oplus)$

Theorem 3.11 *Let $G = (V, E)$ be a graph with at least one edge. Then $\mathbb{E}[\text{SDCUT}(G)] \geq 0.878 \cdot \text{OPT}(G)$, i.e. algorithm* SDCUT *has an expected approximation ratio of at most $\frac{1}{0.878} = 1.139$.*

Proof. Let the indicator variables $X_{i,j}$ be defined as

$$X_{i,j} = \begin{cases} 1 & \text{if } \{i, j\} \in E \text{ crosses the cut} \\ 0 & \text{otherwise} \end{cases}$$

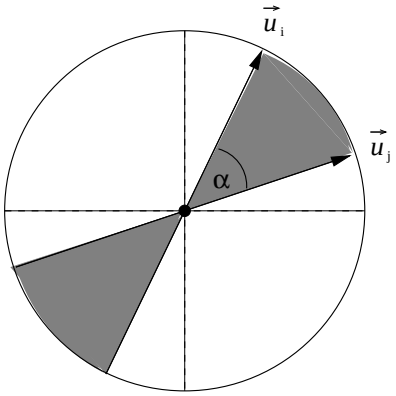

Figure 2: Angle α between $\vec{u}_i$ and $\vec{u}_j$.

Obviously, $\mathbb{E}[X_{i,j}] = a_{i,j} \cdot \Pr[x_i \neq x_j]$ and therefore

$$\begin{aligned}\mathbb{E}[\text{SDCUT}(G)] &= \mathbb{E}\left[\sum_{\{i,j\}\in E} X_{i,j}\right] = \sum_{\{i,j\}\in E} \mathbb{E}[X_{i,j}] = \sum_{i<j} a_{i,j} \cdot \Pr[x_i \neq x_j] \\ &= \sum_{i<j} a_{i,j} \Pr[\text{sgn}(\vec{r}^{\,T} \cdot \vec{u}_i) \neq \text{sgn}(\vec{r}^{\,T} \cdot \vec{u}_j)] \ .\end{aligned}$$

In Figure 2 we see the 2-dimensional plane spanned by $\vec{u}_i$ and $\vec{u}_j$. Due to symmetry reasons, the probability that the random hyperplane separates $\vec{u}_i$ and $\vec{u}_j$ is equal to the ratio of the angle α between $\vec{u}_i$ and $\vec{u}_j$ and π. Since $\cos\alpha = \frac{\vec{u}_i^{\,T} \cdot \vec{u}_j}{|\vec{u}_i| \cdot |\vec{u}_j|} = \vec{u}_i^{\,T} \cdot \vec{u}_j$, we get

$$\Pr[\text{sgn}(\vec{r}^{\,T} \cdot \vec{u}_i) \neq \text{sgn}(\vec{r}^{\,T} \cdot \vec{u}_j)] = \frac{\alpha}{\pi} = \frac{\arccos(\vec{u}_i^{\,T} \cdot \vec{u}_j)}{\pi}$$

and therefore

$$\begin{aligned}\mathbb{E}[\text{SDCUT}(G)] &= \sum_{i<j} a_{i,j} \cdot \frac{\arccos(\vec{u}_i^{\,T} \cdot \vec{u}_j)}{\pi} \\ &\overset{(a)}{\geq} \sum_{i<j} a_{i,j} \cdot \frac{0.8785(1 - \vec{u}_i^{\,T} \cdot \vec{u}_j)}{2} \\ &= 0.8785 \cdot \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - \vec{u}_i^{\,T} \cdot \vec{u}_j) \\ &= 0.8785 \cdot \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - y_{i,j}^{(\epsilon)}) \\ &\overset{(10)}{\geq} 0.8785 \cdot (\text{OPT}(G) - \epsilon) \\ &\overset{(b)}{\geq} 0.878 \cdot \text{OPT}(G) \ .\end{aligned}$$

For (a) we used that for all $z \in [-1, +1]$,

$$\frac{\arccos z}{\pi} \geq \frac{0.8785(1 - z)}{2} \ .$$

(b) holds because of the choice of ϵ. □

Semidefinite optimization was possible, because we were able to describe the MAXCUT problem as a quadratic program. Quadratic programs can often be transformed into semidefinite programs in the way described above. The semidefinite optimization influences at the point the quality of the solution where we used and bounded $\arccos$.

Let α^* be the largest value for α with $\frac{\arccos z}{\pi} \geq \frac{\alpha(1-z)}{2}$. The smaller the ϵ, the closer can the approximation ratio of SDCUT be brought to $1/\alpha^*$.

The graph has n nodes, but we apparently only did a single random experiment, namely in line (4) of SDCUT. Did we separate n objects by a single coin toss? *No*, because we actually selected an n-dimensional vector! In order to determine its components, at least n coin tosses are necessary.

The expected approximation ratio of 1.139 presented here is the best known so far. Since the algorithm can be derandomized (in an involved way) [MR99], the value is also the best known for deterministic approximation algorithms for MAXCUT. Håstad has shown that $\mathbf{P} = \mathbf{NP}$ if there is an approximation algorithm for MAXCUT with an approximation ratio of $\rho < \frac{17}{16} = 1.0625$.

MAXCUT is a so-called symmetric problem since the set U in $(U, \bar{U})$ can be switched to $\bar{U}$ without changing the value of the found solution. In contrast, MAXSAT is an asymmetric problem because in general we cannot reverse the obtained truth assignment for the variables without changing the solution.

Since 1994, the semidefinite optimization approach has been successfully used for many other problems such as graph coloring problems, graph bisection problems, and the maximum independent set problem, where the best algorithms are now based on semidefinite optimization.

References

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for max sat. In *Proc. of the 11th ACM Symp. on Discrete Algorithms (SODA)*, pages 96–105, 2000.

[Fei98] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[Kar91] H. Karloff. *Linear Programming*. Birkhäuser, Boston, 1991.

[MR99] S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. *SIAM Journal on Computing*, 28:1641–1663, 1999.

[RT87] P. Raghavan and C.D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[TS97] C.-P. Teo and J. Sethuraman. LP-based approach to optimal stable matchings. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 710–719, 1997.

[WSV00] H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors. *Handbook of Semidefinite Programming*. Kluwer, 2000.