8 Techniques for randomized approximation algorithms

In the first half of the lecture we only looked at deterministic algorithms. In this section we will demonstrate how to use random techniques to achieve good approximate solutions. In recent years, randomized algorithms are more and more used, since they allowed to find good approximate solutions to important optimization problems such as the Euclidean TSP. However, the problem with randomized algorithms is that they may only be good on average. For those that do not really like the view of having algorithms that may just be good on average or with a certain probability, we will also demonstrate how to *derandomize* randomized algorithms, i.e. how to turn them into deterministic algorithms that guarantee the same approximation ratio as the average ratio of the original algorithm.

8.1 The probabilistic method

We start with the presentation of an elementary method that does nothing else than randomly generating a solution, and which is on average surprisingly good. We apply it to the optimization variant of the well-known satisfiability problem (or SAT). In order to define the problem, we introduce some basic notation in logics.

A *Boolean variable* is a variable that can take the two truth values TRUE and FALSE. Boolean variables can be combined to *Boolean expressions* using the Boolean connectives $\vee$ (logical "or"), $\wedge$ (logical "and"), and $\neg$ (logical "not"). Formally, a Boolean expression can be any one of

(a) a Boolean variable,
(b) an expression of the form $\neg\Phi_1$, where Φ_1 is a Boolean expression,
(c) an expression of the form $(\Phi_1 \vee \Phi_2)$, where Φ_1 and Φ_2 are Boolean expressions, or
(d) an expression of the form $(\Phi_1 \wedge \Phi_2)$, where Φ_1 and Φ_2 are Boolean expressions.

In case (b) the expression is called the *negation* of ϕ_1; in case (c), it is the *disjunction* of ϕ_1 and ϕ_2; and in case (d), it is the *conjunction* of ϕ_1 and ϕ_2. An expression of the form x_i or $\neg x_i$, where x_i is a Boolean variable, is called a *literal.* Instead of $\neg x_i$, one often uses the notation $\bar{x}_i$.

A Boolean expression Φ is in *conjunctive normal form* (or CNF) if $\Phi = \bigwedge_{i=1}^n C_i$, where each of the C_is is the disjunction of one or more literals. The C_is are called the *clauses* of the expression. Similarly, we say that an expression ϕ is in *disjunctive normal form* (or DNF) if $\phi = \bigvee_{i=1}^n D_i$, where each of the D_is is the conjunction of one or more literals. The D_is are called the *implicants* of the expression.

A *truth assignment* T is a mapping from a finite set X of Boolean variables to the set of truth values $\{\text{TRUE}, \text{FALSE}\}$. 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 T that satisfies Φ (i.e. gives it a value of TRUE).

In the *satisfiability problem* (SAT) we are given a Boolean expression Φ in CNF and the problem is to determine whether Φ is satisfiable. The optimization variant of the SAT problem is called MAXSAT: given a Boolean expression Φ in CNF, the problem is to find an assignment that maximizes the number of satisfied clauses in Φ. Since SAT is NP-complete (actually, it was the first problem for which NP-completeness was shown [Coo71]), MAXSAT is believed to be hard to solve exactly.

We will show with the help of the following algorithm A that for every Boolean expression in CNF there is an assignment T so that at least half of the clauses are satisfied. Let $\Phi = C_1 \wedge \ldots \wedge C_m$ be an expression in CNF with n Boolean variables.

Algorithm A

for $i = 1$ to n do

$$\text{with probability} \begin{cases} 1/2: & x_i = \textsc{True} \\ 1/2: & x_i = \textsc{False} \end{cases}$$

Under the assumption that the x_i's are selected independently at random we arrive at the following result.

Lemma 8.1 *Let k_j be the number of literals in C_j. It holds:*

$$\Pr[A \textit{ satisfies } C_j] = 1 - \frac{1}{2^{k_j}} .$$

Proof. The probability that a single literal is equal to FALSE is $1/2$. Since the x_i's are independent, this implies that the probability that all k_j literals in C_j are equal to FALSE is $(1/2)^{k_j}$. Hence, the probability that at least one of the literals of C_j (and therefore C_j) is true is equal to $1 - (1/2)^{k_j}$. □

An alternative proof follows directly from elementary counting. There are 2^{k_j} different assignments for the k_j literals in C_j. All up to one contain at least one literal with value TRUE. Hence, apart from this single outcome, all other outcomes satisfy C_j. Since all outcomes have the same probability,

$$\Pr[A \text{ satisfies } C_j] = \frac{2^{k_j} - 1}{2^{k_j}} = 1 - \frac{1}{2^{k_j}} .$$

Theorem 8.2 *For every Boolean expression Φ in CNF that consists of m clauses of at at least k literals each,*

$$\mathrm{E}[A(\Phi)] \geq \left(1 - \frac{1}{2^k}\right) \cdot m .$$

Proof. For $j \in \{1, \ldots, m\}$ let Z_j be the random variable with

$$Z_j = \begin{cases} 1 & \text{if } T_A(C_j) = \textsc{True} \\ 0 & \text{otherwise} \end{cases}$$

for the random assignment T_A selected by A. Since Z_j is a binary random variable, it holds that

$$\mathrm{E}[Z_j] = \Pr[Z_j = 1] = \Pr[A \text{ satisfies } C_j] .$$

Hence, it follows from Lemma 8.1 that

$$\begin{aligned} \mathrm{E}[A(\Phi)] &= \mathrm{E}\left[\sum_{j=1}^{m} Z_j\right] = \sum_{j=1}^{m} \mathrm{E}[Z_j] \\ &= \sum_{j=1}^{m} \left(1 - \frac{1}{2^{k_j}}\right) \geq m \cdot \left(1 - \frac{1}{2^k}\right) . \end{aligned}$$

□

Corollary 8.3 *Since k is at least 1, it holds that for every Boolean expression in CNF at least half of the clauses can be satisfied.*

Corollary 8.3 is actually sharp in the sense that in general it is not possible to fulfill more than half of all clauses. The *probabilistic method* that leads to the proof of Corollary 8.3 simply works as follows: since the expected value of the number of satisfied clauses is at least $m/2$, it follows that there must *exist* an assignment that satisfies at least half of the clauses. Otherwise, Fact 5.5 would be violated.

In order to measure the performance of randomized algorithms, we have to adapt the definition of the approximation ratio to the new situation. We will do this by simply replacing the value $A(I)$ obtained by algorithm A on input I by the *expected value* $\mathrm{E}[A(I)]$, i.e. we consider the *expected* approximation ratio

$$\mathrm{E}[\rho_A(I)] = \max\left\{\frac{\mathrm{OPT}(I)}{\mathrm{E}[A(I)]}, \frac{\mathrm{E}[A(I)]}{\mathrm{OPT}(I)}\right\} .$$

With this new definition and Corollary 8.3 we obtain:

Theorem 8.4 *Algorithm A has for every Boolean expression Φ in CNF an expected approximation ratio of*

$$\mathrm{E}[\rho_A(\Phi)] = \frac{\mathrm{OPT}(\Phi)}{\mathrm{E}[A(\Phi)]} \leq \frac{m}{\mathrm{E}[A(\Phi)]} \leq \frac{m}{m/2} = 2 .$$

If the shortest clause contains at least 2 literals, the expected approximation ratio of A is at most $4/3$.

In order to obtain an upper bound on the approximation ratio it suffices to use the trivial fact that $\mathrm{OPT}(\Phi)$ can be at most m. Of course, the exact value of $\mathrm{OPT}(\Phi)$ might be far away from m. One might ask whether it is possible to prove a bound on $A(\Phi)$ that is much closer related to the exact value of $\mathrm{OPT}(\Phi)$ (as we did this already several times, for example, when we analyzed the performance of Christofides' algorithm). Recently, Chen, Friesen and Zheng [CFZ99] showed that this is possible. Their analysis allows to prove an upper bound on the expected approximation ratio of A of at most $3/2$ for all Φ.

8.2 Randomized rounding

We develop now an algorithm that is gets better with a shorter length of the longest clause. The idea behind the algorithm is to first formulate the problem as an integer program, then to solve the linear relaxation of it in an optimal way by allowing real numbers as solution, and afterwards to use *randomized rounding* to get back from real numbers to integers.

Algorithm A sets every variable x_i equal to True with a probability of $p_i = 1/2$. In the *randomized rounding technique* the probabilities p_i are computed before they are used to perform a random experiment.

Let $\Phi = C_1 \wedge \ldots \wedge C_m$ be a Boolean expression in CNF. Let $S_j^{\oplus}$ be the set of variables that occur in a non-negated form in C_j, and let $S_j^{\ominus}$ be the set of variables that occur in a negated form in C_j. For every Boolean variable x_i we introduce a 0-1-variable $\hat{x}_i$, and for every clause C_j we use a 0-1-variable $\hat{Z}_j$, where 0 represents the value False and 1 represents the value True. The number of satisfied clauses is $\sum_{j=1}^{m} \hat{Z}_j$. This procedure is called *arithmetization* of the problem.

Integer program for MAXSAT:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{j=1}^{m} \hat{Z}_j & \\ \text{subject to} & \displaystyle\sum_{x_i \in S_j^{\oplus}} \hat{x}_i + \sum_{x_i \in S_j^{\ominus}} (1-\hat{x}_i) \geq \hat{Z}_j & \text{for all } j \\ & \hat{x}_i, \hat{Z}_j \in \{0,1\} \qquad \text{for all } i,j & \end{array} \tag{1}$$

An optimal solution to this integer program (or IP) 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 a linear program (or 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:

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

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 solution to a feasible integral solution? Simply by using a randomized rounding approach.

RANDOMIZEDROUNDING(π):

for $i = 1$ to n do

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

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

Algorithm B

1. determine an optimal solution x^* to the linear relaxation for Φ
2. set $\pi_i = x_i^*$ for every i $\quad (*)$
3. select an assignment T_B with RANDOMIZEDROUNDING(π)

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

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

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

We again assume that the x_i's are chosen independently at random in RandomizedRounding. The next lemma establishes a relationship between $T_B(C_j)$ and Z_j^*.

Lemma 8.5 *Let k_j be the number of literals in C_j. Then it holds:*

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

Proof. We need two well-known facts.

Fact 8.6

(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

$$C_j = \left(\bigvee_{x_i \in S_j^{\oplus}} x_i\right) \vee \left(\bigvee_{x_i \in S_j^{\ominus}} \bar{x}_i\right) \ .$$

Hence, it follows from the linear program formulation that

$$Z_j^* \leq \sum_{x_i \in S_j^{\oplus}} x_i^* + \sum_{x_i \in S_j^{\ominus}} (1 - x_i^*) \ . \qquad (2)$$

Obviously, C_j is false if and only if all $x_i \in S_j^{\oplus}$ are set to False and all $x_i \in S_j^{\ominus}$ are set to True. This happens with probability

$$\left[\prod_{x_i \in S_j^{\oplus}} (1 - x_i^*)\right] \cdot \left[\prod_{x_i \in S_j^{\ominus}} x_i^*\right] \ .$$

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

$$\begin{aligned}
&\Pr[B \text{ satisfies } C_j] \\
&\quad = \quad 1 - \left[\prod_{x_i \in S_j^{\oplus}} (1 - x_i^*)\right] \cdot \left[\prod_{x_i \in S_j^{\ominus}} x_i^*\right] \\
&\quad \overset{\text{Fact 8.6(a)}}{\geq} \quad 1 - \left(\frac{\sum_{x_i \in S_j^{\oplus}} (1 - x_i^*) + \sum_{x_i \in S_j^{\ominus}} x_i^*}{k_j}\right)^{k_j} \\
&\quad = \quad 1 - \left(\frac{|S_j^{\oplus}| - \sum_{x_i \in S_j^{\oplus}} x_i^* + |S_j^{\ominus}| - \sum_{x_i \in S_j^{\ominus}} (1 - x_i^*)}{k_j}\right)^{k_j}
\end{aligned}$$

$$
\begin{aligned}
&= \quad 1-\left(\frac{k_j-\left(\sum_{x_i\in S_j^{\oplus}} x_i^* + \sum_{x_i\in S_j^{\ominus}}(1-x_i^*)\right)}{k_j}\right)^{k_j} \\
&\stackrel{(2)}{\geq} \quad 1-\left(1-\frac{Z_j^*}{k_j}\right)^{k_j} \\
&\stackrel{\text{Fact 8.6(b)}}{\geq} \quad \left(1-\left(1-\frac{1}{k_j}\right)^{k_j}\right)\cdot Z_j^*
\end{aligned}
$$

□

Theorem 8.7 *For every Boolean expression Φ in CNF in which there are at most k literals in every clause,*

$$\mathrm{E}[B(\Phi)] \geq \left(1-\left(1-\frac{1}{k}\right)^k\right)\cdot \mathrm{OPT}(\Phi)\ .$$

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

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

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

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

Theorem 8.8 *The expected number of satisfied clauses achieved by algorithm B on input Φ is at least $(1-1/\mathrm{e})\cdot \mathrm{OPT}(\Phi) \approx 0.632 \cdot \mathrm{OPT}(\Phi)$. In other words: algorithm B has an expected approximation ratio of $\mathrm{E}[\rho_B(\Phi)] \leq \frac{1}{1-1/\mathrm{e}} \approx 1.582$.*

We used for the randomized rounding in algorithm B 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 approximation ratio is at most $4/3$. 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.*

8.3 A hybrid approach

In the following we will see that one does not have to come up with cleverly chosen functions for π to obtain an approximation ratio of $4/3$. If one combines the algorithms A and B, then one can obtain the same approximation ratio.

k	$\frac{1}{\mathrm{E}[\rho_A]} = 1 - \frac{1}{2^k}$	$\frac{1}{\mathrm{E}[\rho_B]} \geq 1 - \left(1 - \frac{1}{k}\right)^k$
1	0.5	1.0
2	0.75	0.75
3	0.875	0.704
4	0.938	0.684
5	0.969	0.672

Figure 1: Probabilities of satisfying a clause with k literals

Consider Table 1. It shows the expected fraction of the clauses that can be fulfilled by A and B if all clauses in Φ have a length of exactly k. This problem is called Max-EkSAT, where the "E" stands for "exact". As one can see, algorithm B can cope well with the case that the clauses are short, whereas algorithm A can cope well with the case that the clauses are long. The smallest maximum of their two values is $3/4$.

There are two closely related methods to combine these two algorithms in an automatic way:

1. We can toss a coin to decide which of the two algorithms to use.
2. We can run both algorithms and output the best of the two solutions.

Concerning the first case, consider the following algorithm:

Algorithm C_p:

$$\text{with probability} \begin{cases} p: & \text{run } A \\ 1-p: & \text{run } B \end{cases}$$

Let the algorithm that runs both A and B be called C_{all}. C_{all} is a representative of so-called *hybrid* approximation algorithms. In general, given k algorithms $A_1, \ldots, A_k$ for the same optimization problem Π, let C_{all} be the algorithm that runs all algorithms A_i and outputs the best solution obtained by them. Furthermore, given a probability distribution $p = (p_1, \ldots, p_k)$, let C_p be the algorithm that decides to execute algorithm A_i with probability p_i. The following result shows that C_{all} is at least as good as any algorithm C_p.

Theorem 8.9 *For any $p_1, \ldots, p_k \in [0,1]$ with $\sum_i p_i = 1$ it holds for any instance I of Π:*

- *If Π is maximization problem, then $E[C_{\text{all}}(I)] \geq \mathrm{E}[C_p(I)]$.*
- *If Π is minimization problem, then $E[C_{\text{all}}(I)] \leq \mathrm{E}[C_p(I)]$.*

The proof of the theorem will be an assignment. It directly follows from this result that for the case of algorithms A and B,

$$\mathrm{E}[C_{\text{all}}(\Phi)] \geq \mathrm{E}[C_p(\Phi)] = p \cdot \mathrm{E}[A(\Phi)] + (1-p) \cdot \mathrm{E}[B(\Phi)]$$

for all $p \in [0,1]$. In order to prove a lower bound on $\mathrm{E}[C_{\text{all}}(\Phi)]$, it therefore remains to prove an upper bound on $\mathrm{E}[C_p(\Phi)]$ for any p.

Theorem 8.10 *Algorithm $C_{1/2}$ has an expected approximation ratio of $4/3$.*

Proof. Lemma 8.1 and 8.5 imply that

$$\mathrm{E}[A(\Phi)] = \sum_{k=1}^{n} \sum_{C_j \text{ with } k \text{ literals}} \left(1 - \frac{1}{2^k}\right) \geq \sum_{k=1}^{n} \sum_{C_j \text{ with } k \text{ literals}} \left(1 - \frac{1}{2^k}\right) Z_j^* \tag{3}$$

and

$$\mathrm{E}[B(\Phi)] \geq \sum_{k=1}^{n} \sum_{C_j \text{ with } k \text{ literals}} \left(1 - \left(1 - \frac{1}{k}\right)^k\right) Z_j^* \tag{4}$$

We were able to add Z_j^* to the expression in (3), since $0 \leq Z_j^* \leq 1$ for all j. (3) and (4) imply that

$$\begin{aligned}
\mathrm{E}[C_{1/2}(\Phi)] &= \tfrac{1}{2}(\mathrm{E}[A(\Phi)] + \mathrm{E}[B(\Phi)]) \\
&\geq \sum_{k=1}^{n} \sum_{C_j \text{ with } k \text{ literals}} \frac{1}{2} \underbrace{\left(1 - \frac{1}{2^k} + 1 - \left(1 - \frac{1}{k}\right)^k\right)}_{\geq 3/2 \text{ for any } k \in \mathbb{N}} \cdot Z_j^* \\
&\geq \frac{3}{4} \sum_{j=1}^{m} Z_j^* \geq \frac{3}{4} \cdot \mathrm{OPT}(\Phi)\ .
\end{aligned}$$

□

The randomized rounding technique is due to Raghavan and Thompson [RT87]. The best known approximation algorithm for MaxSAT with a polynomial runtime achieves an (expected) approximation ratio of 1.2004 [AW00]. It uses semidefinite optimization and combines this with a hybrid algorithm based on the algorithms presented above and some other algorithms. We will describe in the following what semidefinite optimization means and how it can be applied to hard optimization problems.

8.4 Semidefinite optimization

In this subsection we will continue to refine our randomized rounding techniques. This time, we will concentrate on a graph problem. We will show how to arithmetize it, to represent it as a quadratic program, and then how to convert it into a semidefinite program. An excellent overview of applications for semidefinite optimization can be found in [WSV00].

8.4.1 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

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

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 8.11 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):

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

OPT(SDP) represents the optimal value achievable for the given SDP.

Fact 8.12

(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 IR^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 8.13

(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 $B = \begin{pmatrix} 1 & -1 \\ 0 & \sqrt{3} \end{pmatrix}$.

8.5 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 8.14 An instance of the MaxCUT problem is an indirected, connected graph $G = (V, E)$. A feasible solution for G is a partition $S = (V_\ominus, V_\oplus)$ of the node set V so that $V_\ominus \cup V_\oplus = V$ and $V_\ominus \cap V_\oplus = \emptyset$. S is called a *cut* through G. An edge $\{u, v\} \in E$ is said to cross the cut S if $u \in V_\ominus$ and $v \in V_\oplus$. The objective is to maximize

$$C(S) = |\{\{u, v\} \in E \mid u \in V_\ominus, \ v \in V_\oplus\}| \, .$$

$C(S)$ is called the *size* of the cut S.

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 8.15 *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 5.7). 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 $A_G = (a_{i,j}) \in M(n, n; \mathrm{IR})$ be the adjacency matrix of the given graph $G = (V, E)$, i.e. for all nodes $u_i, u_j \in V$,

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

In order to perform an arithmetization of the problem, we select a variable x_i for each node u_i that can take the values -1 and $+1$. $x_i = -1$ means that $u_i \in V_\ominus$, and $x_i = +1$ means that $u_i \in V_\oplus$. For an edge $\{u_i, u_j\} \in E$ 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,

$$C(S) = \sum_{\{u_i,u_j\}\in E} \tfrac{1}{2}(1 - x_i x_j) = \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - x_i x_j) \; .$$

($\sum_{i<j}$ 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:

$$\begin{aligned} &\text{maximize} && \tfrac{1}{2}\sum_{i<j} a_{i,j}(1 - x_i x_j) && (5)\\ &\text{subject to} && x_i \in \{-1, +1\} \quad \text{for all } i && (6) \end{aligned}$$

Since in the objective function (5) two variables are multiplied with each other, we talk about a *quadratic* optimization problem. Due to condition (6) it is hard to solve. However, just relaxing (6) to $-1 \le x_i \le 1$ 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:

$$\begin{aligned} &\text{maximize} && \tfrac{1}{2}\sum_{i<j} a_{i,j}(1 - \vec{u}_i^{\,T}\vec{u}_j) && (7)\\ &\text{subject to} && \vec{u}_i^{\,T} \cdot \vec{u}_i = 1 \quad \text{for all } i && (8) \end{aligned}$$

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 $Y = (y_{i,j}) \in M(n, n; \mathrm{IR})$. 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 symetric positive semideinite. Thus, we can formulate now the semidefinite program corresponding to the MaxCUT problem:

Semidefinite program "SD-CUT" for MaxCUT:

$$\begin{aligned} &\text{maximize} && \tfrac{1}{2}\sum_{i<j} a_{i,j}(1 - y_{i,j}) && (9)\\ &\text{subject to} && Y = (y_{i,j}) \text{ is symmetric and positive semidefinite} && (10)\\ & && y_{i,i} = 1 \quad \text{for all } i && (11) \end{aligned}$$

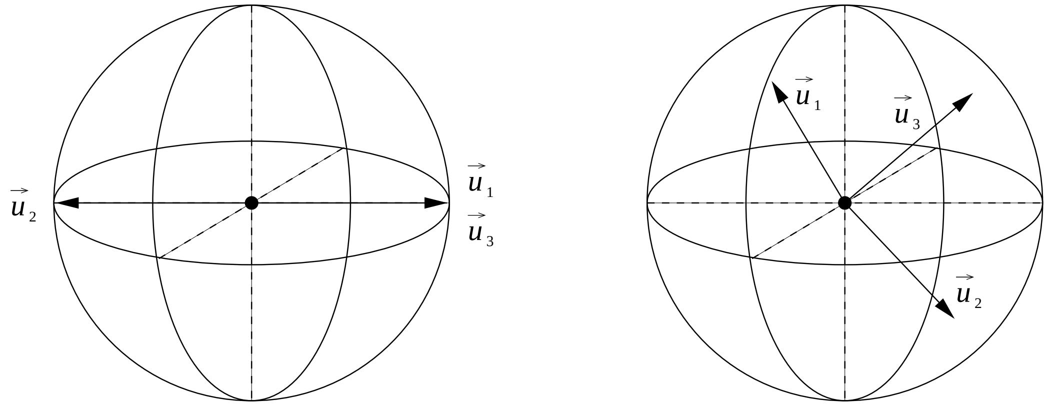

Figure 2: Possible vectors in the unit ball: (a) in the quadratic program and (b) in the semidefinite program

According to Fact 8.12(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

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

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

$$\frac{1}{2}\sum_{i<j} a_{i,j}(1 - y_{i,j}^{(\epsilon)}) \geq \mathrm{OPT}(G) - \epsilon \ . \tag{12}$$

This bound will be useful later.

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 8.12(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 2(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 2(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 original 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 $\mathrm{sgn}(\vec{r}^{\,T} \cdot \vec{u}_i) \neq \mathrm{sgn}(\vec{r}^{\,T} \cdot \vec{u}_j)$. Hence, the complete algorithm looks as follows:

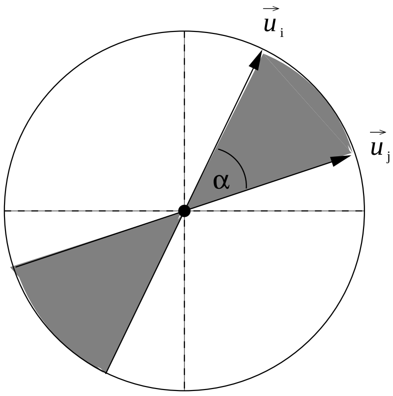

Figure 3: Angle α between $\vec{u}_i$ and $\vec{u}_j$.

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 u_i to $V_\oplus$)
   else $x_i = -1$ (i.e. add u_i to $V_\ominus$)
6. output $(V_\ominus, V_\oplus)$

Theorem 8.16 *Let $G = (V, E)$ be a graph with at least one edge. Then* $\mathrm{E}[\mathrm{SDCut}(G)] \geq 0.878 \cdot \mathrm{OPT}(G)$*, 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

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

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

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

In Figure 3 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 $\cos\alpha = \frac{\vec{u}_i^{\,T} \cdot \vec{u}_j}{|\vec{u}_i| \cdot |\vec{u}_j|} = \vec{u}_i \cdot \vec{u}_j$, we get

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

and therefore

$$\begin{aligned}
\mathrm{E}[\mathrm{SDC}\textsc{ut}(G)] &= \sum_{i<j} a_{i,j} \cdot \frac{\arccos(\vec{u}_i^{\,T} \cdot \vec{u}_j)}{\pi} \\
&\overset{(a)}{\geq} \sum_{i<j} a_{i,j} \cdot \frac{0.8785(1 - \vec{u}_i^{\,T} \cdot \vec{u}_j)}{2} \\
&= 0.8785 \cdot \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - \vec{u}_i^{\,T} \cdot \vec{u}_j) \\
&= 0.8785 \cdot \tfrac{1}{2} \sum_{i<j} a_{i,j}(1 - y_{i,j}^{(\epsilon)}) \\
&\overset{(12)}{\geq} 0.8785 \cdot (\mathrm{OPT}(G) - \epsilon) \\
&\overset{(b)}{\geq} 0.878 \cdot \mathrm{OPT}(G)\ .
\end{aligned}$$

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 sets $V_\ominus$ and $V_\oplus$ can be exchanged 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.

9 Derandomization

Derandomization is the process of converting a randomized algorithm into a deterministic algorithm. It would be nice if one could derandomize the algorithms presented earlier in this section in an automatic way without sacrificing too much of the approximation ratio or the runtime. Unfortunately, no universally applicable method is known so far that can achieve this. However, we will present a fairly general method, called *method of conditional expectations*, that allows to derandomize a large class of randomized algorithms. The method usually converts the expected approximation ratio into a guaranteed approximation ratio while increasing the runtime by a factor that is polynomial in the input size.

In the following, we will restrict ourselves to optimization problems $\Pi = (\mathcal{D}, S, f, \mathrm{opt})$ in which for every instance $I \in \mathcal{D}$ the set of all feasible solutions for I consists of assigning one out of two possible values to each of the variables $x_1, \ldots, x_{n(I)}$. W.l.o.g., these two values are 0 and 1, i.e. for all $I \in \mathcal{D}$, $S(I) = \{0,1\}^{n(I)}$.

A mapping $s : \mathcal{D} \times \{0,1\} \to \mathcal{D}$ is called a *substitution* for Π if for every instance $I \in \mathcal{D}$ and every assignment $x_1 = \alpha$ with $\alpha \in \{0,1\}$ an instance $s(I, \alpha)$ is generated that consists only of variables $\{x_2, \ldots, x_{n(I)}\}$. If there is no confusion about the substitution used, we will simply write $I_{x_1=\alpha}$ instead of $s(I, \alpha)$.

For MAXSAT we will use the following substitution. 0 will take over the role of FALSE and 1 will take over the role of TRUE.

Let $\Phi = C_1 \wedge \ldots \wedge C_m$ be a Boolean expression in CNF with n Boolean variables and let α be a truth value. Then $\Phi_{x_1=\alpha}$ is the Boolean expression that remains when x_1 is given the value α. For this we use the following rules for each clause C_j:

- If x_1 does not occur in C_j, then C_j does not change.
- If setting $x_1 = \alpha$ causes C_j to become true, then C_j will be replaced by the expression TRUE (which we will view here as a clause).
- If setting $x_1 = \alpha$ causes a literal in C_j to become false, then that literal is removed from C_j. If this causes C_j to be empty, the whole C_j is removed from Φ.

Obviously, it takes at most $O(n \cdot m)$ time to compute $\Phi_{x_1=\alpha}$.

Consider, for example, $\Phi = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$. Then the rules above imply that $\Phi_{x_1=\text{FALSE}} = \text{TRUE} \wedge (\bar{x}_2 \vee x_3)$.

Definition 9.1 *Let Π be an optimization problem with a substitution scheme and A be a randomized algorithm for Π with the following properties:*

(a) A assigns values in $\{0,1\}$ to the variables $x_1, \ldots, x_{n(I)}$ independently at random.

(b) For all instances I and all $\alpha \in \{0,1\}$ it holds that $\mathrm{E}[A(I) \mid x_1 = \alpha] \lesseqqgtr \mathrm{E}[A(I_{x_1=\alpha})]$ ("$\leq$": maximization problem, "$\geq$": minimization problem).

(c) The analysis of A allows to determine for every instance I in polynomial time the exact *value of $\mathrm{E}[A(I)]$.*

If these conditions are satisfied, then we call A well analyzed*.*

$\mathrm{E}[A(I) \mid x_1 = \alpha]$ is called the *conditional expectation* of the quality of the output of A when started with I, i.e. the expectation under the assumption that A will select the value α for x_1. In other words, $\mathrm{E}[A(I) \mid x_1 = \alpha]$ is the average value produced by A by computations that set $x_1 = \alpha$.

Algorithm A presented in Section 8.1 for the MAXSAT problem is well analyzed:

- The values for the variables are selected uniformly and independently at random.

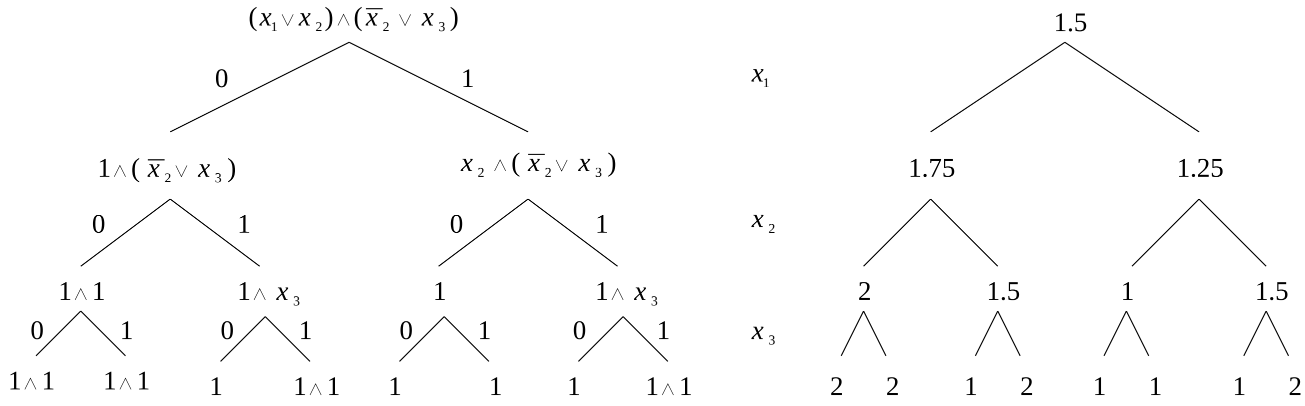

Figure 4: (a) The computation tree for $\Phi = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$ and (b) the expectations for the number of clauses satisfied by A.

- Obviously, $\mathrm{E}[A(\Phi) \mid x_1 = \textsc{False}] = \mathrm{E}[A(I_{x_1=\textsc{False}})]$ and $\mathrm{E}[A(\Phi) \mid x_1 = \textsc{True}] = \mathrm{E}[A(I_{x_1=\textsc{True}})]$ under the assumption that we treat $\textsc{True}$ as a clause.
- From Lemma 8.1 we know the exact expectation for $A(\Phi)$: $\mathrm{E}[A(\Phi)] = \sum_{j=1}^{m}(1 - 1/2^{k_j})$, where $k_j = \infty$ if $C_j = \textsc{True}$ (so that $1 - 1/2^{k_j} = 1$).

Given an instance Φ of MaxSAT, we will view all possibilities of substituting values for the Boolean variables in Φ as a so-called *binary computation tree*. The root of the tree is Φ, and its two children represent the cases $\Phi_{x_1=\textsc{False}}$ and $\Phi_{x_1=\textsc{True}}$. The edges from the root to the two children are labelled accordingly with values $\textsc{False}$ and $\textsc{True}$. The construction is continued with $x_2, x_3, \ldots$ until all x_i have been considered. This results in a complete binary tree of depth n. A leaf ϕ of the tree only consists of clauses of the form $\textsc{True}$. Thus, it certainly holds that $\mathrm{E}[A(\phi)] = A(\phi)$, i.e. at the leaves A is not a randomized algorithm any more.

Now, we also add to each node u in the computation tree the expectation $\mathrm{E}[A(u)]$. Since A is well analyzed, we can compute $\mathrm{E}[A(u)]$ exactly in polynomial time and therefore do not have to look at the whole subtree below u to determine this value (which would need an exponential runtime!).

This construction can be applied to arbitrary optimization problems Π (with a domain of $\{0,1\}$ for each x_i) and arbitrary well analyzed algorithms A. For every leaf v of the tree (which represents an instance of Π), $\mathrm{E}[A(v)] = A(v) = f(\alpha_1, \ldots, \alpha_{n(I)})$, where f is the objective function in Π and $\alpha_1, \ldots, \alpha_{n(I)}$ are the values assigned to the variables $x_1, \ldots, x_{n(I)}$.

For all maximization problems and all approximation algorithms A it holds that

$$\begin{aligned} \mathrm{E}[A(I)] &= \Pr[x_1 = 0] \cdot \mathrm{E}[A(I) \mid x_1 = 0] + \Pr[x_1 = 1] \cdot \mathrm{E}[A(I) \mid x_1 = 1] \\ &\leq \max\{\mathrm{E}[A(I) \mid x_1 = 0],\ \mathrm{E}[A(I) \mid x_1 = 1]\}\ . \end{aligned} \tag{13}$$

(For minimization problems, we simply have to replace "$\leq$" by "$\geq$" and "max" by "min".) Inequality (13) is correct, because the probability values have to sum up to 1.

Now, let A be a well analyzed algorithm for Π. According to (13) and Definition 9.1(b), for any inner node u of the computation tree for A there is a child v of u with $\mathrm{E}[A(u)] \leq \mathrm{E}[A(v)]$. Hence, if we start at the root and always follow the child with the largest expectation, then we will arrive at a leaf w at the end with $A(w) = \mathrm{E}[A(w)] \geq \mathrm{E}[A(I)]$. Such a path is determined by the following algorithm:

Algorithm DERAND_A:

for $i = 1$ to $n(I)$ do
  $e_0 = \mathrm{E}[A(I) \mid x_1, \ldots, x_{i-1} \ \wedge \ x_i = 0]$
  $e_1 = \mathrm{E}[A(I) \mid x_1, \ldots, x_{i-1} \ \wedge \ x_i = 1]$
  if $e_0 \leq e_1$
    then $x_i = 1$
    else $x_i = 0$
output $x_1, \ldots, x_{n(I)}$

The necessary substitutions can be computed in polynomial time and also, since A is well analyzed, the values e_0 and e_1. Applied to Φ in Figure 4, DERAND_A chooses the assignment $x_1 = x_2 = x_3 =$ FALSE. Although DERAND is optimal in this example, there are other examples for which it is not.

Theorem 9.2 *Let* Π *be a maximization problem and* A *be a well analyzed approximation algorithm for* Π *with polynomial runtime and expected approximation ratio* $\mathrm{E}[\rho_A(I)]$*. Then the deterministic algorithm* DERAND_A *is an approximation algorithm for* Π *with worst-case approximation ratio* $\rho_{\text{DERAND\_A}}(I) \leq \mathrm{E}[\rho_A(I)]$.

Where does is it actually used that A has a polynomial runtime, and where do we need to execute something from A? The answers to these closely related questions can be best given by looking at the algorithm B given in Section 8.2. From the proof of Lemma 8.5 we get the exact formula for $\mathrm{E}[B(\Phi)]$. It depends on the solution of the linear relaxation. Thus, in order to compute $\mathrm{E}[B(\Phi)]$ (and $\mathrm{E}[B(v)]$ for all other nodes of the computation tree), we first have to compute an optimal solution of the linear relaxation for Φ. Only then we know the necessary probabilities, and only then we can use the formula in Lemma 8.5 to compute the expected values.

Since A and B can be derandomized, also the algorithm $C_{1/2}$ can be derandomized. However, note that one has to use one additional trick to ensure that the random choice of A or B at the beginning is also derandomized.

Corollary 9.3 *The algorithm* DERAND_$\mathrm{C}_{1/2}$ *has a worst-case approximation ratio of* $4/3$.

Algorithms based on semidefinite programming like the algorithm we presented for MAXCUT are much harder to derandomize. The only random variables are the entries of the vector $\vec{r}$ that defines the hyperplane through the n-dimensional unit ball. Before that and after that the algorithm works in a deterministic way. The problem with applying the approach above to these random variables is that it seems to be quite difficult to determine exact (or sufficiently exact) conditional expectations in this case. Nevertheless, Mahajan and Ramesh [MR99] were able to develop a method that derandomizes the approximation algorithm for MAXCUT, but with an involved method that is not based n substitutions.

Obviously, we can replace the condition that each x_i can only have two possible values by the condition that x_i can have a polynomial number of possible values (and sometimes even more). In this case, the degree of the computation tree will increase accordingly, and in DERAND_A the value with the largest expectation will be taken. One can also relax in many cases the condition to have stochastic independence.

If one cannot compute $\mathrm{E}[A(I)]$ exactly, it is sometimes possible to find at least reasonably good approximations for $\mathrm{E}[A(I)]$ so that the derandomization method still works. Such an approach is, for instance, the *method of pessimistic estimators*.

All of these methods and their relaxations have in common that only nodes along a single path of the computation tree is considered, which ensures that the runtime is polynomial.

The algorithm DERAND_A for MAXSAT was known long before it was recognized as the derandomized version of the algorithm A. It is due to Johnson [Joh74]. In [CFZ99] it was even shown that DERAND_A has an approximation ratio of 3/2, which is also the value of its relative deviation. More approaches and applications of derandomization methods can be found in Chapter 3 of [MPS98].

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.

[CFZ99] J. Chen, D.K. Friesen, and H. Zheng. Tight bound on Johnson's algorithm for maximum satisfiability. *Journal of Computer and System Sciences*, 58:622–640, 1999.

[Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proc. of the 3rd ACM Symp. on Theory of Computing (STOC)*, pages 151–158, 1971.

[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.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

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

[MPS98] E.W. Mayr, H.J. Prömel, and A. Steger. *Lectures on Proof Verification and Approximation Algorithms.* Springer, Berlin, 1998.

[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.

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