Optimization and Random Rounding

Improvisation certainly is the touch-stone of spirit.

Molière

7.1 Objectives

Integer linear programming (ILP) and 0/1-linear programming (0/1-LP) are known NP-hard optimization problems. On the other hand, the linear programming problem (LP) is solvable in polynomial time. Interestingly, all three problems, ILP, 0/1-LP, and LP, are specified by the same kind of linear constraints and have the same kind of linear objectives. The only difference is in requiring integer solutions for ILP and Boolean solutions for 0/1-LP, while the basic problem of linear programming is considered over real numbers.

How is it possible that there are such large differences in the computational complexities of these problems, while they look so similar? The reason is that the constraints requiring integer or Boolean solutions cannot be formulated as linear equations, and so ILP and 0/1-LP leave the area of linear constraints in this sense. The linearity of LP is really substantial, because the set of feasible solutions of a system of linear inequalities (of an instance of LP) builds a polytype. To search for an optimum with respect to a linear function in a polytype is not so hard, because one can reach an optimum by applying a local search starting from an arbitrary vertex of the polytype and moving along the edges of the polytype.

This strong similarity between the efficiently solvable LP and the NP-hard ILP and 0/1-LP is used in order to design efficient approximation algorithms. First, one expresses an instance of a discrete optimization problem as an instance of ILP or 0/1-LP. Usually, this can be done very easily, because the constraints of many hard optimization problems are naturally expressible in

1i.e., without any restriction on the type of input values
2a convex, multidimensional object
3This is exactly what the famous Simplex algorithm does. The Simplex algorithm runs very fast on almost all instances of LP. But there exist artificial LP instances requiring exponentially many local improvements during the execution of the local search to reach an optimum.
the form of linear equations (or inequalities).\(^4\) The second step is called the relaxation to linear programming. Here, one does not take the constraints requiring integer or Boolean solutions into account, and efficiently solves the given problem instance as an instance of LP. The third and final step is devoted to the conversion of the computed optimal solution\(^5\) for LP to a reasonably good, feasible solution to the original problem instance. One of the possible conversion strategies is random rounding, which we aim to present in this chapter.

This chapter is organized as follows. In Section 7.2 we introduce the method of relaxation to linear programming and show how one can get approximation algorithms by applying this method.

In Section 7.3 we combine the method of relaxation to LP with random rounding in order to design a randomized approximation algorithm for MAX-SAT. We shall see that this algorithm in incomparable with the random sampling algorithm for MAX-SAT presented in Section 2.5 (Exercise 2.5.72), where the incomparability is considered with respect to the quality (approximation ratio) of computed solutions. In Section 7.4 we merge these two randomized approximation algorithms for MAX-SAT and obtain a randomized algorithm whose expected approximation ratio is at most 4/3 for any given formula in CNF.

Finally, Section 7.5 provides a short summary of the most important ideas of the chapter and a survey of related results and recommended sources for more involved study of this topic.

7.2 Relaxation to Linear Programming

The relaxation to linear programming is one of the most frequently applied methods for designing algorithms for NP-hard, discrete optimization problems. The basic schema of this method can be described as follows.

Schema of the Relaxation to Linear Programming

Input: An instance \(I\) of an optimization problem \(U\)

(1) Reduction:
Express \(I\) as an instance \(\text{ILP}(I)\) of ILP (or 0/1-LP).

(2) Relaxation:
Consider \(\text{ILP}(I)\) as an instance \(\text{Rel-LP}(I)\) of LP (i.e., do not take the constraints requiring integer or Boolean solutions into account), and compute an optimal solution \(\alpha\) for \(\text{Rel-LP}(I)\) by a method of linear programming.

\(^4\)Due to this linear programming problems became the paradigmatic problems of combinatorial optimization and operations research.

\(^5\)Which is not necessarily a feasible (integer or Boolean) solution to the original problem instance.
It is important to observe, that the cost of $\alpha$ is a bound on the achievable optimal cost of the original problem instance $I$, i.e., that

$$\text{cost}(\alpha) = \text{Opt}_{\text{LP}}(\text{Rel-LP}(I)) \leq \text{Opt}_U(I)$$

if $U$ is a minimization problem, and

$$\text{cost}(\alpha) = \text{Opt}_{\text{LP}}(\text{Rel-LP}(I)) \geq \text{Opt}_U(I)$$

if $U$ is a maximization problem.

The reason for this is that the constraints of Rel-LP($I$) are a proper subset of the constraints of ILP($I$) (and so of $I$). Therefore the set $\mathcal{M}(\text{ILP}(I))$ of feasible solutions for $I$ is a subset of the set $\mathcal{M}(\text{Rel-LP}(I))$ of feasible solutions for Rel-LP($I$).

(3) Solving the original problem:

Use $\alpha$ in order to compute a feasible solution $\beta$ for $I$ that is of a sufficiently high quality (i.e., optimal or a good approximation of an optimal solution).

{Though one is often unable to efficiently compute an optimal solution for $I$, and so to estimate the cost $\text{Opt}_U(I)$, the approximation ration of $\beta$ can be upper bounded by comparing $\text{cost}(\alpha) = \text{Opt}_{\text{LP}}(\text{Rel-LP}(I))$ with $\text{cost}(\beta)$.}

Parts (1) and (2) of the schema are executable in polynomial time. Therefore, part (3) corresponds to an NP-hard problem if one forces to compute an optimal solution.\(^6\) If one aims to design an approximation algorithm only, one can search for a strategy by executing step (3), which runs in polynomial time and guarantees a reasonable approximation ratio for every problem instance.

To illustrate the method of relaxation to linear programming, we give a few examples of reductions to ILP and then present the design of an approximation algorithm.

The problem of linear programming is one of the fundamental optimization problems in mathematics. Its importance lies especially in the fact that many real situations and frameworks can be well modeled by LP and in the fact that many different optimization problems can be expressed in terms of LP.

A general version of LP that accepts equations as well as inequalities is as follows:

For every problem instance $A = [a_{ji}]_{j=1,\ldots,m, \ i=1,\ldots,n}$, $b = (b_1, \ldots, b_m)^T$, $c = (c_1, \ldots, c_n)$, $M_1, M_2 \subseteq \{1, \ldots, m\}$, $M_1 \cap M_2 = \emptyset$, $n, m \in \mathbb{N} - \{0\}$,

minimize the linear function $f_c(x_1, \ldots, x_n) = \sum_{i=1}^{n} c_i \cdot x_i$

---

\(^6\)This does not exclude the possibility of efficiently computing optimal solutions for several specific instances.
under the linear constraints\textsuperscript{7}
\[\sum_{i=1}^{n} a_{ji} \cdot x_i = b_j \text{ for } j \in M_1,\]
\[\sum_{i=1}^{n} a_{si} \cdot x_i \geq b_s \text{ for } s \in M_2, \text{ and}\]
\[\sum_{i=1}^{n} a_{ri} \cdot x_i \leq b_r \text{ for } r \in \{1, 2, \ldots, m\} - (M_1 \cup M_2).\]

If \(x = (x_1, x_2, \ldots, x_n)^T\) is considered over real numbers, then one can solve this problem in polynomial time.\textsuperscript{8} If one adds the additional nonlinear constraints \(x_i \in \{0, 1\}\) or \(x_i \in \mathbb{Z}\), one obtains the NP-hard problems 0/1-LP and ILP.

In what follows we present a few examples of the reduction and the relaxation of a few discrete optimization problems.

**Example 7.2.1. The minimum vertex cover problem (MIN-VC)**

Remember that an instance of MIN-VC is a graph \(G = (V, E)\). A feasible solution is any vertex set \(U \subseteq V\) such that each edge from \(E\) has at least one end point in \(U\). The objective is to minimize the cardinality of \(U\).

Let \(V = \{v_1, v_2, \ldots, v_n\}\). One can represent a feasible solution \(U\) by a Boolean vector \((x_1, x_2, \ldots, x_n) \in \{0, 1\}^n\), where
\[x_i = 1 \iff v_i \in U.\]

This representation of feasible solutions enables us to express an instance \(G = (V, E)\) of MIN-VC as the following instance ILP(\(G\)) of 0/1-LP:

\[
\text{Minimize } \sum_{i=1}^{n} x_i \tag{7.1}
\]

under the \(|E|\) linear constraints
\[
x_i + x_j \geq 1 \text{ for every edge } \{v_i, v_j\} \in E \tag{7.2}
\]
and the \(n\) nonlinear constraints
\[
x_i \in \{0, 1\} \text{ for } i = 1, 2, \ldots, n. \tag{7.3}
\]

\textsuperscript{7}We know that each LP instance can be reduced to normal forms that allow either only equations or only inequalities. This can be done efficiently by introducing new variables, but it is a topic of operations research, and we omit the presentation of such details here.

\textsuperscript{8}It was an open question for a long time whether or not LP is solvable in polynomial time (see Section 7.5).
If one relaxes (7.3) to the $2 \cdot n$ linear constraints

$$x_i \geq 0 \text{ and } x_i \leq 1 \text{ for } i = 1, 2, \ldots, n,$$

(7.4)

one obtains the instance Rel-LP($G$) of LP. \hfill \square

**Exercise 7.2.2.** Consider the weighted MIN-VC, where every vertex has been assigned a positive integer weight, and the task is to minimize the overall weight of the vertex cover. Express this optimization problem as 0/1-LP.

**Example 7.2.3. The knapsack problem (MAX-KP)**

An instance $I$ of MAX-KP is given by a sequence of $2 \cdot n + 1$ positive integers $I = w_1, w_2, \ldots, w_n, c_1, c_2, \ldots, c_n, b$ for a positive integer $n$. The idea is to consider $n$ objects, where for $i = 1, 2, \ldots, n$, $w_i$ is the weight of the $i$-th object and $c_i$ is the cost of the $i$-th object. One has a knapsack whose weight capacity is bounded by $b$ and the aim is to pack some objects in the knapsack in such a way that the weight of the knapsack content is not above $b$ and the overall cost of objects in the knapsack is maximized. Again, one can describe any feasible solution by a vector $(x_1, x_2, \ldots, x_n) \in \{0, 1\}^n$, where

$$x_i = 1 \Leftrightarrow \text{the } i\text{-th object is in the knapsack.}$$

Then, an instance $I$ of MAX-KP can be expressed as the following instance ILP($I$) of 0/1-LP:

$$\text{Maximize } \sum_{i=1}^{n} c_i \cdot x_i$$

under the linear constraint

$$\sum_{i=1}^{n} w_i \cdot x_i \leq b,$$

and the $n$ nonlinear constraints

$$x_i \in \{0, 1\} \text{ for } i = 1, \ldots, n.$$

If one exchanges the constraints $x_i \in \{0,1\}$ by the following $2 \cdot n$ linear constraints

$$x_i \leq 1 \text{ and } x_i \geq 0 \text{ for } i = 1, 2, \ldots, n,$$

then one obtains the corresponding relaxed instance Rel-LP($I$) of LP. \hfill \square

**Exercise 7.2.4.** The maximum matching problem is to find a matching of maximum cardinality in a given graph $G$. Express any instance of this problem as an instance of 0/1-LP.
Example 7.2.5. The set cover problem (MIN-SC)

An instance of MIN-SC is a pair \((X, \mathcal{F})\), where \(X = \{a_1, \ldots, a_n\}\) and \(\mathcal{F} = \{S_1, S_2, \ldots, S_m\}\), \(S_i \subseteq X\) for \(i = 1, \ldots, m\). A feasible solution is any set \(S \subseteq \mathcal{F}\), such that \(X = \bigcup_{S \in S} S\). This task is to minimize the cardinality of \(S\). Similarly as in Exercise 7.2.1 and Exercise 7.2.3 one can represent a feasible solution \(S\) by a Boolean vector \((x_1, x_2, \ldots, x_m) \in \{0, 1\}^m\), such that

\[
x_i = 1 \iff S_i \in S.
\]

We introduce the notation

\[
\text{Index}(k) = \{d \in \{1, \ldots, m\} \mid a_k \in S_d\}
\]

for \(k = 1, 2, \ldots, n\). Then, one can express \((X, \mathcal{F})\) as the following instance of 0/1-LP:

\[
\text{Minimize } \sum_{i=1}^{m} x_i
\]

under the \(n\) linear constraints

\[
\sum_{j \in \text{Index}(k)} x_j \geq 1 \text{ for } k = 1, 2, \ldots, n
\]

and under the \(m\) nonlinear constraints

\[
x_i \in \{0, 1\} \text{ for } i = 1, 2, \ldots, m.
\]

\hfill \Box

Exercise 7.2.6. What do MIN-VC and MIN-SC have in common? Can one consider one of these problems as a special case of the other?

We have seen that some optimization problems can be expressed as instances of ILP in a very natural way. Hence, part (1) of the schema of the relaxation to LP is usually the simplest one. As already observed, part (2) can be efficiently performed.\(^9\) The development of efficient algorithms for LP is a part of operations research. Since the details of their design are not directly related to our considerations and aims, we omit the details of the execution of part (2). From our point of view, the most creative part of the design of approximation algorithms by the method of the relaxation to LP is part (3), for which one does not have any universal, or at least robust, approach working for a large class of optimization problems. We finish this section by presenting an example that shows that sometimes even simple rounding can help.

Example 7.2.7. Consider the MIN-VC problem that we already expressed as 0/1-LP in Exercise 7.2.1.

\(^9\)Though the corresponding algorithms and their analyses are highly nontrivial.
Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in [0, 1]^n$ be an optimal solution for the relaxed instance Rel-LP ($G$), which is determined by (7.1), (7.2), and (7.4). Clearly, an $\alpha_i \notin \{0, 1\}$ (from $[0, 1]$) does not have any interpretation for MIN-VC. Hence, we have to create a $\beta = (\beta_1, \beta_2, \ldots, \beta_n) \in \{0, 1\}^n$ from $\alpha$. Let us do this by simply rounding the $\alpha_i$’s in the following way:

$$\beta_i = 1 \iff \alpha_i \geq \frac{1}{2}.$$ 

Next, we show that this algorithm following the schema of the relaxation to LP is a 2-approximation algorithm.

From the constraints (7.2) we see that the optimal solution $\alpha$ for Rel-LP ($G$) must satisfy the inequality

$$\alpha_i + \alpha_j \geq 1$$

for every edge $\{v_i, v_j\} \in E$. Hence,

$$\alpha_i \geq \frac{1}{2} \text{ or } \alpha_j \geq \frac{1}{2},$$

and so rounding $\alpha_i$ and $\alpha_j$ one obtains

$$\beta_i = 1 \text{ or } \beta_j = 1.$$ 

Therefore, at least one of the vertices $v_i$ and $v_j$ is in the resulting\(^\text{10}\) vertex cover, and so the edge $\{v_i, v_j\}$ is covered. Thus, we have shown that $\beta$ is a feasible solution for $G$.

Since one rounds to the closest value from $\{0, 1\}$,

$$\beta_i \leq 2 \cdot \alpha_i,$$

and so

$$\text{cost}(\beta) = \sum_{i=1}^{n} \beta_i \leq 2 \cdot \sum_{i=1}^{n} \alpha_i = 2 \cdot \text{cost}(\alpha). \quad (7.5)$$

Since the set of feasible solutions for the instance ILP ($G$) of $0/1$-LP (and so for $G$ as an instance of MIN-VC) is a subset of the set of feasible solutions for Rel-LP ($G$) as an instance of LP,

$$\text{cost}(\alpha) = \text{Opt}_{\text{LP}}(\text{Rel-LP}(G)) \leq \text{Opt}_{\text{MIN-VC}}(G). \quad (7.6)$$

In this way one finally obtains

$$\text{Ratio}(G) = \frac{\text{cost}(\beta)}{\text{def.} \cdot \text{Opt}_{\text{MIN-VC}}(G)} \leq \frac{2 \cdot \text{cost}(\alpha)}{\text{cost}(\alpha)} = 2.$$ 

\(\square\)

\(^{10}\)described by $\beta$. 

Exercise 7.2.8. Let $k$ be a positive integer. Consider MIN-SC($k$) as the following restricted version of MIN-SC. The instances of MIN-SC($k$) are usual instances $(X, \mathcal{F})$ of MIN-SC with the additional restriction that each element $x \in X$ is contained in at most $k$ sets from $\mathcal{F}$. Apply the method of the relaxation to LP in order to design a $k$-approximation algorithm for MIN-SC($k$).

{Hint: Observe that MIN-SC(2) and MIN-VC are the same optimization problems.}

7.3 Random Rounding and MAX-SAT

The aim of this section is to combine the method of the relaxation to linear programming with random rounding in order to design a randomized approximation algorithm for MAX-SAT. At least for formulas with short clauses, the expected number of satisfied clauses should be larger than the number of clauses satisfied by the solutions produced by the naive RSAM algorithm from Exercise 2.3.35 that simply generates a random\textsuperscript{11} assignment.

Before presenting the new algorithm, we show how to express an instance of MAX-SAT as an instance of 0/1-LP. Let

$$
\Phi = F_1 \land F_2 \land \ldots \land F_m
$$

be a formula over the set $\{x_1, x_2, \ldots, x_n\}$ of Boolean variables, where $F_i$ is a clause for $i = 1, 2, \ldots, m$, $n, m \in \mathbb{N} - \{0\}$. Let $\text{Set}(F_i)$ be the set of all literals in $F_i$. We denote by $\text{Set}^+(F_i)$ the set of all variables that occur in $F_i$ in the positive setting (without negation), and by $\text{Set}^-(F_i)$ the set\textsuperscript{13} of all variables whose negations occur in $F_i$. We assume $\text{Set}^+(F_i) \cap \text{Set}^-(F_i) = \emptyset$, because in the opposite case $F_i$ is always satisfied, and we do not need to consider such clauses in the instances of MAX-SAT.

We denote by $\text{In}^+(F_i)$ and $\text{In}^-(F_i)$ the set\textsuperscript{14} of indices of the variables in $\text{Set}^+(F_i)$ and $\text{Set}^-(F_i)$, respectively. Using this notation one can express the instance $\Phi$ of MAX-SAT as the following instance $LP(\Phi)$ of 0/1-LP:

$$
\text{Maximize } \sum_{j=1}^{m} z_j
$$

under the $m$ linear constraints

$$
\sum_{i \in \text{In}^+(F_j)} y_i + \sum_{i \in \text{In}^-(F_j)} (1 - y_i) \geq z_j \text{ for } j = 1, 2, \ldots, m \tag{7.7}
$$

and the $n + m$ constraints

\textsuperscript{11} with respect to the uniform probability distribution

\textsuperscript{12} If, for instance, $F_i = x_1 \lor \overline{x_3} \lor \overline{x_8} \lor x_8$, then $\text{Set}(F_i) = \{x_1, x_3, x_7, x_8\}$.

\textsuperscript{13} For $F_i = x_1 \lor \overline{x_3} \lor \overline{x_7} \lor x_8$, $\text{Set}^+(F_i) = \{x_1, x_8\}$ and $\text{Set}^-(F_i) = \{x_3, x_7\}$.

\textsuperscript{14} For $F_i = x_1 \lor \overline{x_3} \lor \overline{x_7} \lor x_8$, $\text{In}^+(F_i) = \{1, 8\}$ and $\text{In}^-(F_i) = \{3, 7\}$.
\[ y_i \in \{0, 1\} \text{ for } i = 1, \ldots, n, \text{ and } \]
\[ z_j \in \{0, 1\} \text{ for } j = 1, \ldots, m. \]  

(7.8)

For \( i = 1, 2, \ldots, n \), the variable \( y_i \) overtakes the role of the Boolean variable \( x_i \) in this representation. The idea of this representation \( \text{LP}(\Phi) \) of \( \Phi \) is that \( z_j \) can have the value 1 only if\(^\text{15}\) at least one variable from \( \text{Set}^+(F_j) \) has been assigned the value 1 or at least one variable from \( \text{Set}^-(F_j) \) has been assigned the value 0 (i.e., only if \( F_j \) is satisfied). Thus, the objective function \( \sum_{j=1}^{m} z_j \) counts the number of satisfied clauses.

The relaxed version \( \text{Rel-LP}(\Phi) \) of \( \text{LP}(\Phi) \) can be obtained from \( \text{LP}(\Phi) \) by exchanging the \( n + m \) constraints (7.8) by the following \( 2 \cdot n + 2 \cdot m \) linear constraints:

\[ y_i \geq 0, \ y_i \leq 1 \text{ for } i = 1, \ldots, n, \text{ and } \]
\[ z_j \geq 0, \ z_j \leq 1 \text{ for } j = 1, \ldots, n. \]  

(7.9)

Let \( \alpha(u) \) for every \( u \in \{y_1, y_2, \ldots, y_n, z_1, z_2, \ldots, z_m\} \) be the value of \( u \) in an optimal solution for the instance \( \text{Rel-LP}(\Phi) \) of \( \text{LP} \). Since the set of feasible solutions of \( 0/1-\text{LP}(\Phi) \) is a subset\(^\text{16}\) of the feasible solutions of \( \text{Rel-LP}(\Phi) \), the following is true:

\[ \sum_{j=1}^{m} \alpha(z_j) \text{ is an upper bound on the number of clauses that can be satisfied by any assignment to the variables of } \Phi. \]  

(7.10)

To produce an assignment to the variables \( x_1, x_2, \ldots, x_n \), one can round the values \( \alpha(y_1), \alpha(y_2), \ldots, \alpha(y_n) \) of the optimal solution

\( (\alpha(y_1), \ldots, \alpha(y_n), \alpha(z_1), \ldots, \alpha(z_m)) \)

of \( \text{Rel-LP}(\Phi) \). How to round is explained in the following presentation of the designed algorithm.

**Algorithm RRR (Relaxation with Random Rounding)**

*Input:* A formula \( \Phi = F_1 \land F_2 \land \ldots \land F_m \) over \( X = \{x_1, \ldots, x_n\} \) in CNF, \( n, m \in \mathbb{N} - \{0\} \).

*Step 1:* Reduce the instance \( \Phi \) of MAX-SAT to the instance \( 0/1-\text{LP}(\Phi) \) of \( 0/1-\text{LP} \) with the constraints (7.7) and (7.8).

*Step 2:* Relax \( 0/1-\text{LP}(\Phi) \) to the instance \( \text{Rel-LP}(\Phi) \) with the constraints (7.7) and (7.9), and solve \( \text{Rel-LP}(\Phi) \) efficiently.

Let \( (\alpha(y_1), \alpha(y_2), \ldots, \alpha(y_m), \alpha(z_1), \alpha(z_2), \ldots, \alpha(z_m)) \) be the computed optimal solution for \( \text{Rel-LP}(\Phi) \).

\(^{15}\)because of the \( j \)-th constraints of (7.7)

\(^{16}\)The constraints (7.8) strengthen the constraints (7.9).
Step 3:
Choose, uniformly, \( n \) values \( \gamma_1, \ldots, \gamma_n \) from the interval \([0, 1]\) at random.
\[
\text{for } i = 1 \text{ to } n \text{ do }
\quad \text{if } \gamma_i \in [0, \alpha(y_i)] \text{ then }
\qquad \beta_i := 1
\quad \text{else}
\qquad \beta_i := 0.
\]

Output: \( \text{RRR}(\Phi) = (\beta_1, \beta_2, \ldots, \beta_n) \)

Hence, \( \alpha(y_i) \) is the probability that \( x_i \) takes the value 1. The main difference with the random sampling algorithm RSAM is that RSAM takes its random choice with respect to the uniform probability distribution over \( \{0, 1\}^n \) while RRR chooses an assignment with respect to the probability distribution determined by \( \alpha(y_1), \alpha(y_2), \ldots, \alpha(y_n) \).

We start our analysis of RRR by giving a lower bound on the probability that RRR satisfies a clause of \( k \) literals.

**Lemma 7.3.9.** Let \( k \) be a positive integer and let \( F_j \) be a clause of \( k \) literals in \( \Phi \). Let \( \alpha(y_1), \ldots, \alpha(y_n), \alpha(z_1), \ldots, \alpha(z_m) \) be the optimal solution of Rel-LP(\( \Phi \)) computed by RRR(\( \Phi \)).

Then the probability that the assignment RRR(\( \Phi \)) satisfies the clause \( F_j \) is at least
\[
\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \alpha(z_j).
\]

**Proof.** Since one considers the clause \( F_j \) independently of other clauses, one can assume without loss of generality that it contains only uncomplemented variables, and that it can be expressed\(^{17}\) as
\[
F_j = x_1 \lor x_2 \lor \ldots \lor x_k.
\]

From the \( j \)-th constraint in (7.7) of Rel-LP(\( \Phi \)), we have
\[
y_1 + y_2 + \ldots + y_k \geq z_j.
\] (7.11)

The clause \( F_j \) remains unsatisfied if and only if all the variables \( x_1, x_2, \ldots, x_k \) are set to zero. Since the random rounding in step 3 of RRR runs independently for each variable, this occurs with the probability
\[
\prod_{i=1}^{k} (1 - \alpha(y_i)).
\]

Complementary, \( F_j \) is satisfied with the probability

\(^{17}\)This way we avoid double indexing and additional notation for negated variables.
\[ 1 - \prod_{i=1}^{k}(1 - \alpha(y_i)). \]  

(7.12)

Under the constraints (7.11), the function (7.12) is minimized when

\[ \alpha(y_i) = \frac{\alpha(z_j)}{k} \]

for \( i = 1, 2, \ldots, k \). In this way we obtain

\[ \text{Prob}(F_j \text{ is satisfied}) \geq 1 - \prod_{i=1}^{k}\left(1 - \frac{\alpha(z_j)}{k}\right). \]  

(7.13)

The lower bound (7.13) on the probability of satisfying \( F_j \) can be viewed as a function of one variable \( \alpha(z_j) \in [0, 1] \). To complete the proof it suffices to show that, for every positive integer \( k \),

\[ f_k(r) = 1 - \left(1 - \frac{r}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot r = g_k(r) \]

(7.14)

for all \( r \in [0, 1] \) (and so for every \( \alpha(z_j) \)). Next, we show that the relation between the functions \( g(r) \) and \( f(r) \) is as shown in Figure 7.1.

![Fig. 7.1.](image)

Since \( f_k \) is a concave function and \( g_k \) is a linear function, it is sufficient to show (Figure 7.1) that
\[ f_k(0) = g_k(0) \text{ and } f_k(1) = g_k(1). \]

Clearly,
\[ f_k(0) = 1 - (1 - 0)^k = 0 = g_r(0) \text{ and } f_k(1) = 1 - \left(1 - \frac{1}{k}\right)^k = g_k(1). \]

Inserting \( r = \alpha(z_j) \) into (7.14) and combining (7.13) with (7.14) one obtains the assertion of Lemma 7.3.9.

\[ \square \]

**Theorem 7.3.10.** The algorithm RRR runs in polynomial time and it is

(i) a randomized \( E\left[\frac{e}{(e-1)^d}\right] \)-approximation algorithm for MAX-SAT and

(ii) a randomized \( E\left[\frac{k^k}{(k^k - (k^k - 1))^d}\right] \)-approximation algorithm for MAX-EkSAT.

**Proof.** First we analyze the time complexity of the algorithm RRR. The reduction in step 1 can be performed in linear time. The instance Rel-LP(\( \Phi \)) of LP can be solved in polynomial time. Step 3 can be executed in linear time.

Following (7.10), in order to show that RRR is an \( E[d] \)-approximation algorithm it suffices to show that the expected number of satisfied clauses is at least

\[ d^{-1} \cdot \sum_{j=1}^{m} \alpha(z_j). \]

Our probability space\(^{18}\) is \( (\{0,1\}^n, \text{Prob}) \), where

\[ \text{Prob}(\{(\delta_1, \delta_2, \ldots, \delta_n)\}) = \prod_{i=1}^{n} q_i, \]

where

\[ q_i = \alpha(y_i) \quad \text{if } \delta_i = 1 \quad \text{and} \quad q_i = 1 - \alpha(y_i) \quad \text{if } \delta_i = 0. \]

for \( i = 1, 2, \ldots, n \). For \( \delta = (\delta_1, \ldots, \delta_n) \) and \( j = 1, 2, \ldots, m \) we consider the random variable \( Z_j \) defined by

\[ Z_j(\delta) = \begin{cases} 1 & \text{if } \delta \text{ satisfies the clause } F_j \\ 0 & \text{if } \delta \text{ does not satisfy the clause } F_j. \end{cases} \]

Since \( Z_j \) is an indicator variable, \( E[Z_j] \) is the probability that the output RRR(\( \Phi \)) of RRR satisfies the clause \( F_j \). Therefore, Lemma 7.3.9 provides

\[ E[Z_j] \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot \alpha(z_j) \]

\[ \tag{7.15} \]

\(^{18}\)We identify each computation of RRR with its output.
if $F_j$ consists of $k$ literals. Let us consider the random variable

$$Z = \sum_{j=1}^{m} Z_j$$

that counts the number of satisfied clauses. If all clauses consist of exactly $k$ literals, the linearity of expectation provides the following lower bound on $\mathbb{E}[Z]$:

$$\mathbb{E}[Z] = \sum_{j=1}^{m} \mathbb{E}[Z_j] \geq \sum_{j=1}^{m} \left( 1 - \left( 1 - \frac{1}{k} \right)^k \right) \cdot \alpha(z_j) \geq \left( 1 - \left( 1 - \frac{1}{k} \right)^k \right) \cdot \sum_{j=1}^{m} \alpha(z_j). \quad (7.16)$$

Since $\text{Opt}_{\text{MAX-SAT}}(\Phi) \leq \sum_{j=1}^{m} \alpha(z_j)$, we obtain

$$\mathbb{E}[\text{Ratio}_{\text{RRR}}(\Phi)] = \frac{\text{Opt}_{\text{MAX-EkSAT}}(\Phi)}{\mathbb{E}[Z]} \leq \frac{\sum_{j=1}^{m} \alpha(z_j)}{\sum_{j=1}^{m} \alpha(z_j)} = \frac{k^k}{k^k - (k-1)^k}.$$ 

Hence, we have proved claim (ii).

Since $(1 - \frac{1}{k})^k \leq e^{-1}$ for all $k \in \mathbb{N} - \{0\}$, we have

$$1 - \left( 1 - \frac{1}{k} \right)^k \geq 1 - \frac{1}{e} \quad (7.17)$$

for all positive integers $k$. Inserting (7.17) into (7.16), we obtain

$$\mathbb{E}[Z] \geq \left( 1 - \frac{1}{e} \right) \cdot \sum_{j=1}^{m} \alpha(z_j).$$

Therefore

$^{19}$if $\Phi$ is an instance of MAX-E$k$SAT
\[ E[\text{Ratio}_{\text{RRR}}(\Phi)] = \frac{\text{Opt}_{\text{MAX-SAT}}(\Phi)}{E[Z]} \leq \left(1 - \frac{1}{e}\right)^{-1} = \frac{e}{e-1}. \]

We note that the algorithm RRR has a special, very appreciative property with respect to the success probability amplification by repeated runs. One does not need to repeat the complete runs of RRR. Once the first two steps of RRR are executed, and so the probability space \((\{0, 1\}^n, \text{Prob})\) is determined, it suffices to execute several independent random choices in this probability space. Thus, the most expensive step 2 is executed only once, independently of the number of assignments one wants to generate at random.

**Exercise 7.3.11.** Let \(\Phi\) be a formula consisting of \(m = 3 \cdot d\) clauses, where \(d\) clauses are of the length 2, the next \(d\) clauses are of length 3, and the last \(d\) clauses consist of 4 literals. Estimate a lower bound for the expectation \(E[\text{Ratio}_{\text{RRR}}(\Phi)]\).

### 7.4 Combining Random Sampling and Random Rounding

Now, we have two different algorithms for MAX-SAT. On the one hand the algorithm RSAM based on random sampling, and on the other hand the algorithm RRR designed in Section 7.3 by the relaxation method and random rounding. Since \(2 > \frac{e}{e-1}\), the algorithm RSAM with \(E[\text{Ratio}_{\text{RSAM}}(\Phi)] \leq 2\) for every formula \(\Phi\) provides in general a weaker guarantee that the algorithm RRR with \(E[\text{Ratio}_{\text{RRR}}(\Phi)] < \frac{e}{e-1}\).

Surprisingly, if one looks at the behavior of these algorithms more carefully, one sees that the naive algorithm RSAM is better for problem instances with long clauses. Since

\[ \frac{2^k}{2^k - 1} < \frac{k^k}{k^k - (k-1)^k} \]

for \(k \geq 3\), RSAM assures a better upper bound on the expected approximation ratio for MAX-\(E_k\)SAT instances than RRR. Hence, one can consider RSAM and RRR as incomparable algorithms for MAX-SAT, because for some formulas RSAM can provide better results than RRR, and vice versa.

**Exercise 7.4.12.** Find an infinite set of input instances of MAX-SAT for which the expected solutions computed by RSAM are better than the expected solutions computed by RRR.

**Exercise 7.4.13.** Find an infinite set of instances of MAX-SAT such that one can expect better results from RRR than from RSAM.
Exercise 7.4.14. Let $k$ be a positive integer, $k \geq 3$. Find instances $\Phi$ of MAX-$E_k$SAT such that

$$E[\text{Ratio}_{\text{RRR}}(\Phi)] < E[\text{Ratio}_{\text{RSAM}}(\Phi)].$$

Because of the incomparability of RSAM and RRR, a very natural idea is to combine these algorithms into one algorithm by running them independently in parallel and then taking the better of the two solutions computed. Next, we show that the resulting algorithm has an expected approximation ratio of at most $4/3$.

**Algorithm COMB**

*Input:* A formula $\Phi = F_1 \land F_2 \land \ldots \land F_m$ in CNF over a set $X$ of Boolean variables.

*Step 1:* Compute an assignment $\beta$ for $X$ by the algorithm RSAM (i.e., $\beta := \text{RSAM}(\Phi)$).

*Step 2:* Compute an assignment $\gamma$ for $X$ by the algorithm RRR (i.e., $\gamma := \text{RRR}(\Phi)$).

*Step 3:* 

if $\beta$ satisfies more clauses of $\Phi$ than $\gamma$ then 

output ($\beta$)

else 

output ($\gamma$).

Theorem 7.4.15. The algorithm COMB is a polynomial-time, randomized $E\left[\frac{4}{3}\right]$-approximation algorithm for MAX-SAT.

Proof. Since both RSAM and RRR work in polynomial time, COMB is a polynomial-time algorithm, too.

Now, we analyze the expected approximation ratio of COMB. Let $\Phi = F_1 \land \ldots \land F_m$ be a formula over $X = \{x_1, \ldots, x_n\}$. Let $(S_{\text{RSAM},\Phi}, \text{Prob}_{\text{RSAM}})$ be the probability space for the analysis of RSAM, where $S_{\text{RSAM},\Phi}$ is the set of all $2^n$ computations of RSAM on $\Phi$ and $\text{Prob}_{\text{RSAM}}$ is the uniform probability distribution over $S_{\text{RSAM},\Phi}$. Let $(S_{\text{RRR},\Phi}, \text{Prob}_{\text{RRR}})$ be the probability space for the analysis of the work of RRR, where $S_{\text{RRR},\Phi}$ is the set of all $2^n$ computations of RRR on $\Phi$ and $\text{Prob}_{\text{RRR}}$ is the probability distribution determined by the optimal solution for Rel-LP($\Phi$) computed in the second step of RRR. Obviously,

$$(S_{\text{RSAM},\Phi} \times S_{\text{RRR},\Phi}, \text{Prob}),$$

with

$$\text{Prob}((C, D)) = \text{Prob}_{\text{RSAM}}(\{C\}) \cdot \text{Prob}_{\text{RRR}}(\{D\})$$

for all $(C, D) \in S_{\text{RSAM},\Phi} \times S_{\text{RRR},\Phi}$ is the probability space for the analysis of the work of COMB on $\Phi$. Let us consider the following random variables.

---

For simplicity we view the computations of COMB as pairs $(C, D)$, where $C$ is a run of RSAM and $D$ is a run of RRR.
For all \((C, D) \in S_{RSAM, \phi} \times S_{RRR, \phi}\), let
\[ Y((C, D)) \]
be the number of clauses satisfied by \(\beta\) as the output of \(C\)

and let
\[ Z((C, D)) \]
be the number of clauses satisfied by the output \(\gamma\) of \(D\).

Hence, \(Y\) counts the number of clauses satisfied by the output of a run of
RSAM, and \(Z\) counts the number of clauses satisfied by the assignment computed by a run of RRR. We introduce a new random variable \(U = \max\{Y, Z\}\), defined by
\[ U((C, D)) = \max\{Y((C, D)), Z((C, D))\} \]
for all computations \((C, D)\) of COMB.

Hence, \(U\) counts the number of clauses satisfied by the output of a run
\((C, D)\) of the algorithm COMB. Since
\[
\max\{Y((C, D)), Z((C, D))\} \geq \frac{Y((C, D)) + Z((C, D))}{2}
\]
for all \((C, D) \in S_{RSAM, \phi} \times S_{RRR, \phi}\), we have
\[
E[U] \geq \frac{E[Y] + E[Z]}{2}.
\]
(7.18)

From the analysis of RRR, we know that no assignment can satisfy more
than \(\sum_{j=1}^{m} \alpha(z_j)\) clauses.

Following (7.18), it is sufficient to show that
\[
\frac{E[Y] + E[Z]}{2} \geq \frac{3}{4} \cdot \sum_{j=1}^{m} \alpha(z_j).
\]
(7.19)

To show this, we investigate the probability of satisfying every particular clause with respect to its length. For each integer \(k \geq 1\), let \(C(k)\) be the set of clauses from \(\{F_1, F_2, \ldots, F_m\}\) that consist of exactly \(k\) literals. Lemma 7.3.9 implies that
\[
E[Z] \geq \sum_{k \geq 1} \sum_{F_j \in C(k)} \left(1 - \left(1 - \frac{1}{2} \right)^k \right) \cdot \alpha(z_j).
\]
(7.20)

Since \(\alpha(z_j) \in [0, 1]\), the analysis of RSAM provides the following lower bound on \(E[Y]\):
\[
E[Y] = \sum_{k \geq 1} \sum_{F_j \in C(k)} \left(1 - \frac{1}{2^k} \right) \geq \sum_{k \geq 1} \sum_{F_j \in C(k)} \left(1 - \frac{1}{2^k} \right) \cdot \alpha(z_j).
\]
(7.21)

Hence, we obtain
\[ E[U] \geq \frac{E[Y] + E[Z]}{2} \] (7.18)

\[ \geq \frac{1}{2} \cdot \sum_{k \geq 1} F_j \in C(k) \left[ \left( 1 - \frac{1}{2k} \right) + \left( 1 - \left( 1 - \frac{1}{k} \right)^k \right) \right] \cdot \alpha(z_j) \] (7.20) (7.21)

\[ \geq \frac{1}{2} \cdot \frac{3}{2} \cdot \sum_{k \geq 1} F_j \in C(k) \sum \alpha(z_j) \]

\[
\{ \text{Since } (1 - 2^{-k}) + (1 - (1 - k^{-1})^k) \geq \frac{3}{2} \text{ for all positive integers } k. \}\]

\[ = \frac{3}{4} \cdot \sum_{j=1}^{m} \alpha(z_j). \]

Therefore,

\[ E[\text{Ratio}_{\text{COMB}}(\Phi)] = \frac{\text{Opt}_{\text{MAX-SAT}}(\Phi)}{E[U]} \leq \frac{\sum_{j=1}^{m} \alpha(z_j)}{\sum_{j=1}^{m} \alpha(z_j)} \]

\[ = \frac{4}{3}. \]

\[ \square \]

Exercise 7.4.16. Implement the algorithm COMB and test it for real MAX-SAT instances. Try to estimate the average approximation ratio with respect to your input data.

7.5 Summary

The relaxation to linear programming is a robust method for designing approximation algorithms for hard optimization problems as well as for computing bounds on the costs of optimal solutions. The kernel of this method is that many problems can be naturally expressed as instances of the NP-hard integer linear programming and 0/1-linear programming, and that the basic problem of linear programming is efficiently computable. Based on these facts, one obtains the following schema for applying this method.

1. Reduction

A given instance of an optimization problem is expressed as an instance of ILP or 0/1-LP.
2. Relaxation

The instance of ILP or 0/1-LP is relaxed to an instance of LP\(^{21}\) by removing the nonlinear constraints of the domain of the output values. The instance of LP is efficiently solved.

3. Solving the original problem instance

The optimal solution of the relaxed LP instance is used to create a feasible solution for the original problem instance. The cost of the computed optimal solution for the relaxed LP instance is a bound on the achievable cost of the feasible solutions for the original problem instance.

While, today, parts 1 and 2 of this schema can be performed by standard, efficient algorithms, part 3 may require that we apply distinct concepts and methods, and it is a matter of investigation for various concrete optimization problems.

In this book we presented two techniques for implementing part 3 of the schema. First, we used simple rounding for the minimum vertex cover problem, and got a polynomial-time 2-approximation algorithm for MIN-VC in this way.

In Section 7.3 we applied random rounding to design a randomized, polynomial-time \(E[e/(e - 1)]\)-approximation algorithm for MAX-SAT. If one executes this algorithm and the algorithm RSAM\(^{22}\) on the same MAX-SAT instance in parallel, and then takes the better of the two computed assignments, one obtains a \(E[4/3]\)-approximation algorithm for MAX-SAT.

The most involved source for the study of the design of approximation algorithms by the relaxation to linear programming is the textbook by Vazirani [Vaz01]. A detailed introduction to the application of this method is presented in [Hro03], too. There are many good textbooks on linear programming. For computer scientists, we warmly recommend the excellent, classical book by Papadimitriou and Steiglitz [PS82].

The famous Simplex algorithm for solving instances of linear programming was discovered by Dantzig [Dan49] in 1947. Klee and Minty [KM72] were the first researchers who constructed LP instances on which the Simplex algorithm does not work efficiently.\(^{23}\) The long stated open problem about the existence of a polynomial-time algorithm for LP was solved by Khachian [Kha94] in 1979 in a positive sense.

An excellent source for the study of MAX-SAT is the book [MPS98] edited by Mayr, Prömel, and Steger.

\(^{21}\)All constraints of the relaxed instance are linear equations or inequalities.

\(^{22}\)which is based on a simple application of the method of random sampling

\(^{23}\)i.e., in exponential time