Success Amplification and Random Sampling

It is not possible to wait for inspiration, and even inspiration alone is not sufficient. Work and more work is necessary. Man blessed by genius can create nothing really great, not even anything mediocre, if he does not toil as hard as a slave.

Piotr Ilyich Tschaikovsky

5.1 Objectives

This chapter is devoted to two paradigms of the design of randomized algorithms, namely the amplification of success probability by repeating runs on the same input and random sampling. The reasons for presenting both these methods in one chapter are their similarity and their equally balanced combination in several applications. Thus, for some randomized algorithms, it is not possible to determine which of these two methods is primarily responsible for success.

In Chapter 2 we called attention to the fact that amplification is a method for reducing the error probability below an arbitrarily given small constant $\epsilon > 0$. We underlined the importance of this observation by classifying randomized algorithms with respect to the speed of error probability reduction with the number of computation repetitions on the same input. In this chapter we aim to present algorithms for which amplification does not only increase the success probability, but directly stamps the process of the algorithm design. Moreover, we do not want only to follow the naive approach of repeating the whole computation on the same input, but also to introduce a more advanced technique that prefers to repeat only some computation parts or to repeat different parts differently many times. The idea is to pay more attention to computation parts in which the probability of making mistakes is greater than in other ones.

With random sampling we aim to document the power of this method by designing efficient randomized algorithms solving problems for which no deterministic polynomial-time algorithm has up to now been discovered.$^1$

This chapter is organized as follows. Section 5.2 introduces the above mentioned generalized version of the amplification method. This method is used

---

$^1$and maybe for which no efficient deterministic algorithms exist at all
to design randomized algorithms solving the minimum cut problem for multigraphs. In Section 5.3 we combine amplification with random sampling in order to design a practicable one-sided-error Monte Carlo algorithm for the well known, NP-hard 3-satisfiability (3SAT) problem. Though this algorithm runs in exponential time, it is much faster than algorithms running in $O(2^n)$ time and it can be successfully applied for relatively large instances of 3SAT. In Section 5.3 we present an application of random sampling that results in a Las Vegas polynomial-time algorithm for a number-theoretic problem, which is not known to be in P. Hence, this randomized algorithm is the only efficient way known for solving this problem. Altogether this chapter presents impressive examples documenting the superiority of randomized algorithms over their best known deterministic counterparts. As usual, we finish the chapter by summarizing the most important ideas and results presented.

### 5.2 Efficient Amplification by Repeating Critical Computation Parts

The aim of this section is to introduce the method of amplification of the success probability as a method for the design of randomized algorithms, and not only (as considered until now) as a technique for error probability reduction of algorithms already designed. For this purpose we consider the following minimization problem MIN-CUT.

**MIN-CUT**

*Input:* A multigraph $G = (V, E, c)$, where $c : E \to \mathbb{N} - \{0\}$ determines the multiplicity of the edges of $G$.

*Constraints:* The set of all feasible solutions for $G$ is the set

$$\mathcal{M}(G) = \{(V_1, V_2) \mid V_1 \cup V_2 = V, V_1 \cap V_2 = \emptyset\}$$

of all cuts of $G$.

*Costs:* For every cut $(V_1, V_2) \in \mathcal{M}(G)$,

$$\text{cost}((V_1, V_2), G) = \sum_{e \in S(V_1, V_2)} c(e),$$

where $S(V_1, V_2) = \{\{x, y\} \in E \mid x \in V_1 \text{ and } y \in V_2\}$

{i.e., cost$((V_1, V_2), G)$ is equal to the number of edges between $V_1$ and $V_2$}

*Goal:* minimum

The best known deterministic algorithm for MIN-CUT runs in time

$$O\left(|V| \cdot |E| \cdot \log \left(\frac{|V|^2}{|E|}\right)\right),$$
which, in the worst case\(^2\), is in \(O(n^3)\) for \(n = |V|\). Our goal is to design an efficient randomized algorithm for MIN-CUT. This algorithm is based on the graph operation \(\text{Contract}(G, e)\) that, for a given multigraph \(G = (V, E)\) and an edge \(e = \{x, y\} \in E\), contracts the edge \(e\). The contraction of \(e = \{x, y\} \in E\) means that

- the vertices \(x\) and \(y\) are replaced by a new vertex \(\text{ver}(x, y)\),
- the multi-edge \(e = \{x, y\}\) is removed (contracted) in this way (we do not allow any self loop),
- each edge \(\{r, s\}\) with an \(r \in \{x, y\}\) and \(s \notin \{x, y\}\) is replaced by a new edge \(\{\text{ver}(x, y), s\}\), and
- all remaining parts of \(G\) remain unchanged.

Visualizing \(\text{Contract}(G, e)\), one simply collapses the vertices \(x\) and \(y\) into one new vertex. We denote the resulting graph by \(G/\{e\}\).

Figure 5.1 shows three contraction operations consecutively executed on the multigraph \(G\) depicted in Figure 5.1(a). First, one executes the operation \(\text{Contract}(G, \{x, y\})\). The resulting multigraph \(G/\{x, y\}\) is depicted in Figure 5.1(b). The next step is the contraction of the edge \(\{u, z\}\), and the resulting multigraph \((G/\{x, y\})/\{u, z\}\) is depicted in Figure 5.1(c). Finally, one contracts the edge \(\{\text{ver}(x, y), v\}\) and obtains the multigraph of two vertices\(^3\) \(\text{ver}(x, y, v)\) and \(\text{ver}(u, z)\) depicted in Figure 5.1(d).

Observe that, given a collection of edges \(F \subseteq E\), the effect of contracting the edges in \(F\) does not depend on the order of contractions. For instance, the multigraph \((G/\{u, z\})/\{x, y\}\) is the same as \((G/\{x, y\})/\{u, z\}\) in Figure 5.1. Therefore we simplify the notation and use \(G/F\) for the resulting multigraph. This way, \(G/\{\{x, y\}, \{u, z\}, \{x, v\}\}\) is a short representation of the multigraph in Figure 5.1(d).

The design idea for the following naive randomized algorithm for MIN-CUT is very simple. One contracts randomly chosen edges until one gets a multigraph with exactly two vertices \(\text{ver}(V_1)\) and \(\text{ver}(V_2)\). Obviously, \(V_1 \cup V_2 = V\) and \(V_1 \cap V_2 = \emptyset\). Hence, \((V_1, V_2)\) is a cut of \(G\), and the number of edges between the two vertices \(\text{ver}(V_1)\) and \(\text{ver}(V_2)\) corresponds to the cost of the cut \((V_1, V_2)\).

In our example, the multigraph in Figure 5.1(d) corresponds to the cut \((\{x, y, v\}, \{u, z\})\) with a cost of 4. In other words, one can view the contraction of an edge \(\{x, y\}\) as saying that the vertices \(x\) and \(y\) must be on the same side of the cut approached and so as restricting the set of all cuts to the cuts with \(x\) and \(y\) on one side. In this way the set of possible cuts is monotonically reduced until it contains only one cut.

\(^2\)if \(|E| \in \Omega(|V|^2)\)

\(^3\)To be precise, one has to write \(\text{ver}(\text{ver}(x, y), v)\) instead of \(\text{ver}(x, y, v)\). But we prefer the shorter representation because, in what follows, the only important matter is that the vertices \(x, y,\) and \(v\) are joined to one vertex. To simplify the notation, we even use \(\text{ver}(V')\) for the vertex created by the union of the vertices in \(V'\).
One can formally describe this approach for solving MIN-CUT as follows. Let $E(G)$ denote the set of edges of a multigraph $G$.

**Algorithm CONTRACTION**

*Input:* A connected\(^4\) multigraph $G = (V, E, c)$

*Step 1:* Set label $(v) := \{v\}$ for every vertex $v \in V$.

*Step 2:*

\[\text{while } G \text{ has more than two vertices do} \]

\[\text{begin} \]

\[\text{choose an edge } e = \{x, y\} \in E(G); \]

\[G := \text{Contract}(G, e); \]

\[\text{Set label } (z) := \text{label} (x) \cup \text{label} (y) \]

\[\text{end} \]

\[\text{endwhile} \]

---

\(^4\)One can determine in time $O(|E|)$ whether or not a multigraph is connected, and so find a minimal cut of cost 0 for any disconnected multigraph.
for the new vertex \( z = \text{ver}(x, y) \);

end

Step 3:

if \( G = (\{u, v\}, E(G)) \) for a multiset \( E(G) \) then

output “(label\((u)\), label\((v)\)” and “cost = \(|E(G)|\)”

Theorem 5.2.1. The algorithm CONTRACTION is a randomized polynomial-time algorithm that computes a minimal cut of a given multigraph \( G \) of \( n \) vertices with probability at least

\[
\frac{2}{n \cdot (n - 1)}.
\]

Proof. Clearly, the algorithm CONTRACTION computes a cut of \( G \), and so a feasible solution for MIN-CUT. The algorithm executes exactly \( n - 1 \) edge contractions for a multigraph of \( n \) vertices, and each contraction can be executed in \( O(n) \) steps. Hence, the time complexity of CONTRACTION is in \( O(n^2) \).

In what follows, we aim to show that the algorithm CONTRACTION finds a minimal cut with probability at least \( \frac{2}{n \cdot (n - 1)} \). Let \( \hat{G} = (V, \hat{E}, c) \) be a multigraph, and let \( \hat{C}_{\text{min}} = (V_1, V_2) \) be a minimal cut of \( G \) with cost \( C_{\text{min}} = k \) for a natural number \( k \). Let \( E(\hat{C}_{\text{min}}) \) denote the set of edges in \( C_{\text{min}} \).

In order to prove the lower bound \( \frac{2}{n \cdot (n - 1)} \) on the success probability, we show that the probability of computing exactly the cut \( C_{\text{min}} \) is at least \( \frac{2}{n \cdot (n - 1)} \).

First, we observe that

\( G \) has at least \( \frac{n \cdot k}{2} \) edges, i.e., \( |E(G)| \geq \frac{n \cdot k}{2} \),

because the minimality of \( C_{\text{min}} \) with cost \( C_{\text{min}} = k \) implies that every vertex of \( G \) has a degree of at least \( k \).

The second important observation is that

the algorithm CONTRACTION computes \( C_{\text{min}} \) if and only if no edge from \( E(C_{\text{min}}) \) has been contracted.

Our aim is to study the probability of this event.

The algorithm consists of \( n - 2 \) contractions. Let \( S_{\text{Con},G} \) be the set of all possible computations of the algorithm CONTRACTION on \( G \). In the probability space \((S_{\text{Con},G}, \text{Prob})\) we investigate the events

\[
\text{Event}_i = \{\text{all computations from } S_{\text{Con},G} \text{ in which no edge of } E(C_{\text{min}}) \text{ is contracted in the } i\text{-th contraction step}\}
\]

for \( i = 1, 2, \ldots, n - 2 \). The event that \( C_{\text{min}} \) is the output of the algorithm is exactly the event

\[
\bigcap_{i=1}^{n-2} \text{Event}_i.
\]
Now, we apply the concept of conditional probabilities (Exercise 2.2.16) in order to calculate the probability of this event.

\[
\text{Prob}\left(\bigcap_{i=1}^{n-2} \text{Event}_i\right) = \text{Prob}(\text{Event}_1) \cdot \text{Prob}(\text{Event}_2 | \text{Event}_1) \\
\quad \cdot \text{Prob}(\text{Event}_3 | \text{Event}_1 \cap \text{Event}_2) \cdot \ldots \\
\quad \cdot \text{Prob}\left(\text{Event}_{n-2} \bigg| \bigcap_{j=1}^{n-3} \text{Event}_j\right)
\]

(5.1)

To prove Theorem 5.2.1, we have to estimate lower bounds on

\[
\text{Prob}\left(\text{Event}_i \bigg| \bigcap_{j=1}^{i-1} \text{Event}_j\right)
\]

for \(i = 1, \ldots, n-2\).

Since \(G\) has at least \(\frac{n \cdot k}{2}\) edges and the algorithm makes a random choice for edge contraction,

\[
\text{Prob}(\text{Event}_1) = \frac{|E| - |E(C_{\min})|}{|E|} = 1 - \frac{k}{|E|} \\
\geq 1 - \frac{k}{k \cdot \frac{n}{2}} = 1 - \frac{2}{n}.
\]

(5.2)

In general, the multigraph \(G/F_i\) created after \(i-1\) random contractions has exactly \(n - i + 1\) vertices. If

\[
F_i \cap E(C_{\min}) = \emptyset \quad \text{(i.e., } \bigcap_{j=1}^{i-1} \text{Event}_j \text{ happens}),
\]

then \(C_{\min}\) is also a minimal cut of \(G/F_i\). Consequently, every vertex in \(G/F_i\) has still to have a degree of at least \(k\), and so \(G/F_i\) has at least

\[
\frac{k \cdot (n - i + 1)}{2}
\]

edges. Therefore,

\[
\text{Prob}\left(\text{Event}_i \bigg| \bigcap_{j=1}^{i-1} \text{Event}_j\right) \geq \frac{|E(G/F_i) - E(C_{\min})|}{|E(G/F_i)|} \\
\geq 1 - \frac{k}{k \cdot \frac{(n-i+1)}{2}} \\
= 1 - \frac{2}{(n - i + 1)}
\]

(5.3)
for $i = 2, \ldots, n - 1$. Inserting (5.3) in (5.1), one obtains

$$\text{Prob}\left(\bigcap_{j=1}^{n-2} \text{Event}_j\right) \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n - i + 1}\right)$$

$$= \prod_{l=n}^{3} \left(\frac{l - 2}{l}\right)$$

$$= \frac{2}{n \cdot (n - 1)} = \frac{1}{\binom{n}{2}}.$$

\[ \square \]

**Exercise 5.2.2.** Modify the algorithm CONTRACTION in the following way. Instead of choosing an edge at random, choose two vertices $x$ and $y$ randomly and join them into one vertex. Construct multigraphs of $n$ vertices, for which the probability that the modified algorithm finds a minimal cut is exponentially small in $n$.

Theorem 5.2.1 assures that the probability of discovering a particular minimal cut in one run is at least

$$\frac{2}{n \cdot (n - 1)} > \frac{2}{n^2}.$$ 

Executing $n^2/2$ independent runs of the algorithm and taking the best output from all computed outputs, one does not obtain a minimal cut with a probability of at most

$$\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}} < \frac{1}{e}.$$ 

Hence, the complementary probability of computing a minimal cut by $n^2/2$ runs of the algorithm is at least

$$1 - \frac{1}{e}.$$ 

The complexity of the algorithm $\text{CONTRACTION}_{n^2/2}$ is in

$$O(n^4).$$

Thus, we need $O(n^4)$ time to compute an optimal solution with a constant probability, and the best known deterministic algorithm computes an optimal solution in time $O(n^3)$ with certainty. The gain of our effort is negative, and one can ask whether it is reasonable to use randomization in this case. But it is not so bad as it seems to be at first glance. The too high complexity is caused by the naive application of the amplification method, in which one increases the success probability by repeating entire runs of the algorithm.
In this case, this standard application of amplification is not clever because the probability of contracting an edge from $C_{\text{min}}$ grows with the number of contractions executed. For the first contractions, this probability\(^5\) is only

\[
\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \frac{2}{n-3}, \frac{2}{n-4}, \ldots,
\]

but for the last edge contractions, it is even as much as $\frac{2}{3}$. A very natural idea is to simply give up the last\(^6\) random contractions because the created multigraph $G/F$, is small enough to be searched for a minimal cut in a deterministic way. In what follows, we describe the algorithm based on this idea. The size of $G/F$ will still remain a free parameter of the algorithm. Let $l : \mathbb{N} \rightarrow \mathbb{N}$ be a monotonic function such that $1 \leq l(n) < n$ for every $n \in \mathbb{N}$.

Algorithm DETRAN($l$)

\begin{itemize}
  \item \textbf{Input:} A multigraph $G = (V, E, c)$ of $n$ edges, $n \in \mathbb{N}, n \geq 3$.
  \item \textbf{Step 1:} Perform the algorithm CONTRACTION on $G$ in order to get a multigraph $G/F$ of $l(n)$ vertices.
  \item \textbf{Step 2:} Apply the best known deterministic algorithm on $G/F$ to compute an optimal cut $D$ of $G/F$.
  \item \textbf{Output:} $D$
\end{itemize}

First, we analyze the influence of the exchange of CONTRACTION for DETRAN($l$) on

(i) the amplification of the success probability, and
(ii) the increase of the complexity\(^7\).

In this analysis we consider $l$ as a free parameter, i.e., the result of the analysis depends on $l$.

\textbf{Lemma 5.2.3.} Let $l : \mathbb{N} \rightarrow \mathbb{N}$ be a monotonic growing function with $1 \leq l(n) < n$. The algorithm DETRAN($l$) works in time

\[O(n^2 + (l(n))^3),\]

and it finds an optimal solution with probability at least

\[
\frac{\binom{l(n)}{2}}{\binom{n}{2}}.
\]

\(^5\)How many contractions have to be considered as the last ones will be analyzed in what follows.

\(^6\)of the events $E_1, E_2, E_3, \ldots$

\(^7\)The increase of time complexity is the cost we pay for the increase of the success probability.
5.2 Efficient Amplification by Repeating Critical Computation Parts

Proof. First we analyze the complexity of DETRAN($l$). In step 1, $n - l(n)$ contractions are performed, and each contraction can be executed in time $O(n)$. Hence, step 1 can be executed in $O((n - l(n)) \cdot n) = O(n^2)$ time. The above mentioned deterministic algorithm can compute an optimal cut of $G/F$ in time $O((l(n))^3)$. Altogether, the complexity of DETRAN($l$) is in $O(n^2 + (l(n))^3)$.

Next we analyze the success probability of DETRAN($l$). As in the proof of Theorem 5.2.1, let $C_{\min}$ be a minimal cut of $G$. We prove a lower bound on the success probability of DETRAN($l$) by proving a lower bound on the probability of having $C_{\min}$ in the multigraph $G/F$ after executing step 1. The corresponding event is

$$\bigcap_{i=1}^{n-l(n)} \text{Event}_i.$$

Applying (5.1), (5.2), and (5.3), one obtains the following lower bound on the probability of this event, and so also on the probability of computing a minimal:

$$\text{Prob} \left( \bigcap_{i=1}^{n-l(n)} \text{Event}_i \right) \geq \prod_{i=1}^{n-l(n)} \left( 1 - \frac{2}{n - i + 1} \right)$$

$$= \frac{\prod_{i=1}^{n-l(n)-2} (1 - \frac{2}{n - i + 1})}{\prod_{j=n-l(n)+1}^{n-l(n)+1} (1 - \frac{2}{n - j + 1})}$$

$$= \frac{1}{\binom{(n)}{2}} = \frac{(l(n))}{2} \cdot \binom{n}{2}.$$

This completes the proof.

Since

$$\frac{n^2}{(l(n))^2} \geq \frac{n^2}{\binom{(l(n))}{2}},$$

$n^2/(l(n))^2$ independent runs of DETRAN($l$) provide a randomized algorithm that works in time

$$O\left((n^2 + (l(n))^3) \cdot \frac{n^2}{(l(n))^2}\right) = O\left(\frac{n^4}{(l(n))^2} + n^2 \cdot l(n)\right)$$

(5.4)

and computes a minimal cut of $G$ with probability at least

$$1 - \frac{1}{e}.$$
The best possible choice of $l$ with respect to the time complexity is

$$l(n) = \left\lfloor \frac{n^{2/3}}{3} \right\rfloor,$$

and making this choice one obtains the following result.

**Theorem 5.2.4.** The algorithm $\text{DETRAN} \left( \left\lfloor \frac{n^{2/3}}{3} \right\rfloor \frac{n^2}{\left\lfloor \frac{n^{2/3}}{3} \right\rfloor} \right)$ works in time

$$O\left( n^{8/3} \right)$$

and computes a minimal cut with probability at least

$$1 - e^{-1}.$$

Now, we have a randomized algorithm that is asymptotically faster than any known deterministic algorithm, and whose error probability can be pushed arbitrarily low. Though we are not satisfied with the designed randomized algorithm. The design strategy is too simple and rough for really capturing the growing error probability of an edge contraction with the number of contractions executed. The algorithm $\text{DETRAN}(l)$ prohibits the execution of the last random contractions with high error probabilities by computing deterministically in its final part, and so increases the success probability. Hence, fewer runs of $\text{DETRAN}(l)$ suffice for getting a constant probability of computing a minimal cut.

If one observes the increase

$$\frac{2}{n}, \frac{2}{n-1}, \frac{2}{n-2}, \frac{2}{n-3}, \ldots, \frac{2}{3}$$

of the error probabilities of the sequence of random contractions, the idea of using fewer run repetitions at the beginning and more at the end of the algorithm $\text{CONTRACTION}$ may appear appropriate. We visualize this idea in what follows. The algorithm $\text{CONTRACTION}$ needs $n^2$ runs of $O(n^2)$ complexity to assure a minimal cut with a constant probability. This corresponds to Figure 5.2, where one lets $n^2$ computations of $\text{CONTRACTION}$ run in parallel. If the complexity of each computation is in $O(n^2)$, then the complexity of the entire work is the area of the rectangle of the size $n^2 \times O(n^2)$, and so $O(n^4)$.

Next, we consider the following implementation of our new idea (Figure 5.3). We start by executing only two independent runs in parallel. After some random edge contractions, when the error probability of choosing a wrong edge has grown substantially, we split each of the two computations

\[ \frac{n^4}{(l(n))^2} = n^2 \cdot l(n). \]

\[ \text{The failure probability tends to 0 with exponential speed with respect to the number of repetitions of } \text{DETRAN} \left( \left\lfloor \frac{n^{2/3}}{3} \right\rfloor \frac{n^2}{\left\lfloor \frac{n^{2/3}}{3} \right\rfloor} \right). \]

\[ \text{an edge of } C_{\min} \]
into two independent runs. After some further contractions we again double the number of runs, and so on. At the very end we also have $O(n^2)$ runs (as in Figure 5.2, too) and we take the best cut of the computed cuts. Hence, the strategy is to use at the end as many runs as in the naive approach of the algorithm CONTRACTION, but to use only a few at the beginning (when the probability of choosing an edge from $C_{\text{min}}$ is small). The crucial point for

Fig. 5.2.

Fig. 5.3.
the complexity analysis of this strategy is estimating the number of random contractions between doubling the number of runs executed in parallel. We double the number of runs always after the number of vertices has been reduced by a factor of $1/\sqrt{2}$ after the last doubling. Since the number of all random contractions is $n - 2$, the number of runs (leaves of the tree in Figure 5.3) is really $2^{\log_\sqrt{2}(n-2)} \in O(n^2)$. Observing Figure 5.3 we see that the complexity of this strategy is the sum of the lengths of all tree edges of the two trees in Figure 5.3. In contrast to the area of the figure rectangle$^{11}$, the trees look light, and so one can expect a substantially smaller complexity. Moreover, if one realizes that the number of edges in the complete binary tree of $O(n^2)$ leaves is in $O(n^2 \cdot \log_2 n^2)$ and that the length of the edges decreases with the number of parallel runs, then one can hope for a complexity in $O(n^2 \log n)$.

To recognize that the success probability of this strategy is large enough is a little bit more complicated. Before doing this, we give a formal presentation of the algorithm.

Algorithm REPTREE($G$)

**Input:** A multigraph $G = (V, E, c)$, $|V| = n$, $n \in \mathbb{N}$, $n \geq 3$.

**Procedure:**

if $n \leq 6$ then
  compute a minimal cut deterministically
else
  begin
    $h := \lceil 1 + \frac{n}{\sqrt{2}} \rceil$;
    Perform two independent runs of CONTRACTION in order to get two multigraphs $G/F_1$ and $G/F_2$ of size $h$;
    REPTREE($G/F_1$);
    REPTREE($G/F_2$)
  end

output the smaller of the two cuts computed by
REPTREE($G/F_1$) and REPTREE($G/F_2$)

**Theorem 5.2.5:** The algorithm REPTREE works in time

$$O(n^2 \cdot \log n)$$

and finds a minimal cut with a probability of at least

$$\frac{1}{\Omega(\log_2 n)}.$$

**Proof.** First we analyze the time complexity of REPTREE. The depths of the binary trees in Figure 5.3 correspond to the number of recursion calls of

$^{11}$which is in $O(n^4)$
5.2 Efficient Amplification by Repeating Critical Computation Parts

REPTREE. Since the size of a given multigraph is reduced by the multiplicative factor \(1/\sqrt{2}\) in each stage of the algorithm, the number of recursion calls is at most

\[ \log_{\sqrt{2}} n \in O(\log_2 n). \]

Since the original algorithm CONTRACTION works in time \(O(n^2)\) on any multigraph of \(n\) vertices, we can roughly bound the time of reducing a multigraph \(G\) of size \(m\) to a multigraph \(G/F\) of size \(\lceil 1 + \frac{m}{\sqrt{2}} \rceil\) by \(O(m^2)\). Hence, we obtain the following recurrence for \(\text{Time}_{\text{REPTREE}}\):

\[
\text{Time}_{\text{REPTREE}}(n) \in O(1) \text{ for } n \leq 6, \text{ and }
\text{Time}_{\text{REPTREE}}(n) = 2 \cdot \text{Time}_{\text{REPTREE}}\left(\left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil\right) + O(n^2). \quad (5.5)
\]

One can easily check that

\[
\text{Time}_{\text{REPTREE}}(n) = \Theta(n^2 \cdot \log_2 n).
\]

In what follows, we estimate a lower bound on the success probability of the algorithm REPTREE. Once again, we do this by analyzing the probability of computing a specific minimal cut \(C_{\min}\) with \(|E(C_{\min})| = k\) for a positive integer \(k\).

First, we pose the following question. Let \(p_l\) be the probability that the multigraph \(G/F_i\) \((i = 1, 2)\) of size \(\lceil 1 + (l/\sqrt{2}) \rceil\) still contains \(C_{\min}\), assuming that the multigraph \(G/F\) of \(l\) vertices has contained \(C_{\min}\) before the computation split into two runs leading to \(G/F_1\) and \(G/F_2\). How large is \(p_l\)? Following the analysis of the algorithm DETRAN, we have

\[
p_l \geq \frac{\left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil\right)}{\left(\frac{l}{2}\right)} = \frac{\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil \cdot \left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil - 1\right)}{l \cdot (l - 1)} \geq \frac{1}{2}.
\]

Let \(\text{Prob}(n)\) be the probability that REPTREE finds a minimal set of a multigraph of \(n\) vertices. Then, for \(i = 1, 2,\)

\[
p_l \cdot \text{Prob}\left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil\right)
\]

is a lower bound on the conditional probability that REPTREE computes \(C_{\min}\) by the reduction from \(G/F\) to \(G/F_i\) and then by the recursive call \(\text{REPTREE}(G/F_i)\), assuming \(G/F\) has contained \(C_{\min}\). Since \(\text{REPTREE}\) starting from \(G/F\) executes two runs of CONTRACTION with outputs \(G/F_1\) and \(G/F_2,\)

\[
\left(1 - p_l \cdot \text{Prob}\left(\left\lceil 1 + \frac{l}{\sqrt{2}} \right\rceil\right)\right)^2
\]

is an upper bound on the conditional probability that REPTREE does not find \(C_{\min}\) assuming \(G/F\) has contained \(C_{\min}\). Hence, we obtain the following recurrence for \(\text{Prob}(n)\):
\[
\text{Prob}(2) = 1, \text{ and } \\
\text{Prob}(l) \geq 1 - \left(1 - p_l \cdot \text{Prob}\left(\left\lceil \frac{l}{\sqrt{2}} \right\rceil \right)\right)^2 \\
\geq 1 - \left(1 - \frac{1}{2} \cdot \text{Prob}\left(\left\lceil \frac{l}{\sqrt{2}} \right\rceil \right)\right)^2 
\] (5.6)

One can show that each function \( \text{Prob} \) satisfying the recurrence (5.6) is in \( \Theta\left(\frac{1}{\log_2 n}\right) \).

\[\square\]

**Exercise 5.2.6.** Prove that the solution of the recurrence (5.5) is a function in \( \Theta(n^2 \cdot \log n) \).

**Exercise 5.2.7.** Prove that each function \( \text{Prob} \) satisfying the recurrence (5.6) is a function that fulfills \( \text{Prob}(n) \geq \frac{1}{\Theta(\log_2 n)} \).

**Exercise 5.2.8\*:** Analyze the success probability and the time complexity of the versions of the algorithm REPTREE, for which one takes the following size reduction between two splits of the computation:

(i) from \( l \) to \( \left\lfloor \frac{l}{2} \right\rfloor \) 
(ii) from \( l \) to \( \sqrt{l} \) 
(iii) from \( l \) to \( \frac{l}{\log_2 l} \) 
(iv) from \( l \) to \( l - \sqrt{l} \).

A consequence of Theorem 5.2.5 is that \( O(\log_2 n) \) repetitions of the algorithm REPTREE are sufficient in order to compute a minimal cut with a constant probability. Already \( O((\log_2 n)^2) \) repetitions suffice to reduce the non-success probability to a function tending to 0 with growing \( n \) and one can consider this algorithm applicable. The complexity of \( \text{REPTREE}_{(\log_2 n)^2} \) is in

\[ O(n^2 \cdot (\log_2 n)^3) \]

which is substantially better than the complexity \( O(n^3) \) of the best deterministic algorithm and the complexity \( O(n^{8/3}) \) of the randomized algorithm \( \text{DETRAN} \left(\left\lceil n^{2/3} \right\rceil \right) / n^{2/3} \).

Hence, the idea of repeating different parts differently many times can be very fruitful.

### 5.3 Repeated Random Sampling and Satisfiability

Here we combine amplification and random sampling with local search in order to design a randomized algorithm that can solve the 3-satisfiability problem\(^{12}\)

---

\(^{12}\)Remember that the instances of 3SAT are formulas in 3CNF, and the task is to decide whether or not a given formula is satisfiable.
(3SAT) for instances of nontrivial size in acceptable time. 3SAT is a known NP-hard problem, and we call attention to the fact that until now no randomized polynomial-time algorithm with a bounded error has been discovered for any NP-hard problem. Our experience with general polynomial-time computations point to the commonly accepted conjecture that NP-hard problems are also hard for randomized polynomial time and so one does not hope for exponential gaps in the time complexity between determinism and randomization. Therefore in the case of the NP-hard 3SAT problem, we aim to design a practicable exponential randomized algorithm. Table 5.1 shows that the design of exponential algorithms can be a reasonable and worthy objective. The values 10, 50, 100, and 300 in the first row represent the input sizes, and the first column contains the running times considered. The particular items in the table give the number of corresponding computer operations. If a number is too large, we provide its number of digits only.

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$n$</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n!$</td>
<td></td>
<td>$\approx 3.6 \cdot 10^6$</td>
<td>(65 digits)</td>
<td>(158 digits)</td>
<td>(625 digits)</td>
</tr>
<tr>
<td>$2^n$</td>
<td></td>
<td>1024</td>
<td>(16 digits)</td>
<td>(31 digits)</td>
<td>(91 digits)</td>
</tr>
<tr>
<td>$2^{n/2}$</td>
<td></td>
<td>32</td>
<td>$\approx 33 \cdot 10^6$</td>
<td>(16 digits)</td>
<td>(46 digits)</td>
</tr>
<tr>
<td>$(1.2)^n$</td>
<td></td>
<td>$\approx 6.19$</td>
<td>9100</td>
<td>$\approx 8.2 \cdot 10^7$</td>
<td>(24 digits)</td>
</tr>
<tr>
<td>$10 \cdot 2^{\sqrt{n}}$</td>
<td></td>
<td>$\approx 30$</td>
<td>$\approx 1345$</td>
<td>10240</td>
<td>$\approx 1.64 \cdot 10^6$</td>
</tr>
<tr>
<td>$n^2 \cdot 2^{\sqrt{n}}$</td>
<td></td>
<td>895</td>
<td>$\approx 336158$</td>
<td>$1.024 \cdot 10^7$</td>
<td>$\approx 1.48 \cdot 10^{11}$</td>
</tr>
<tr>
<td>$n^6$</td>
<td></td>
<td>$10^6$</td>
<td>$1.54 \cdot 10^{10}$</td>
<td>$10^{12}$</td>
<td>$\approx 7.29 \cdot 10^{14}$</td>
</tr>
</tbody>
</table>

If one considers approximately $10^{16}$ operations as the boundary of practically doable, then one observes that algorithms with a running time like $n!$ or $2^n$ are already not applicable for small input sizes. But algorithms with $2^{n/2}$ time complexity are useful for problem instances of size 100 and, for a time complexity of $(1.2)^n$, one can go still further. Algorithms with an exponential time complexity such as $10 \cdot 2^{\sqrt{n}}$ or $n^2 \cdot 2^{\sqrt{n}}$ can be successfully applied for relatively large problem instances (Table 5.1), and are even faster than polynomial-time algorithms with a time complexity of $n^6$ for $n \leq 300$. Moreover, these exponential algorithms run on inputs of size less than 300 in a few

---

13 General is meant in the sense of unrestricted computing models that represent algorithms.

14 The most impressive presentation of the computational power of randomization relative to determinism can be found by restricted frameworks of simple models of computation such as finite automata, pushdown automata, communication protocols, etc.
seconds, something that cannot be said about an \( n^6 \)-algorithm on inputs of size 100.

The aim of this section is to design a 1MC algorithm for 3CNF that works in time

\[
O\left(|F| \cdot n^2 \cdot \left(\frac{4}{3}\right)^n\right)
\]

for each formula in 3CNF over \( n \) variables.\(^{15}\) This running time is not achievable by pure random sampling. If \( F \) is satisfiable, but there are only a few assignments satisfying \( F \), then one would need in average \( \Omega(2^n) \) random samples to find one of these satisfying assignments.

**Exercise 5.3.9.** Let \( F \) be a formula of \( n \) variables that is satisfied by exactly \( k \) assignments to its variables. How many random samples from the set \( \{0, 1\}^n \) are necessary in order to find an assignment satisfying \( F \) with a probability of at least 1/2?

The idea of the design of the algorithm is simple. We repeat at most

\[
O\left(\sqrt{n} \cdot \left(\frac{4}{3}\right)^n\right)
\]

times the following procedure. Choose one of the \( 2^n \) assignments from \( \{0, 1\}^n \) at random and perform at most \( 3 \cdot n \) steps of local search in order to find an assignment satisfying \( F \). One step of local search consists of flipping one bit of the current assignment. Which particular bit is flipped is partially decided at random. A detailed description of the algorithm follows.

**Algorithm SCHÖNING**

**Input:** A formula \( F \) in 3CNF over \( n \) Boolean variables.

**Step 1:**

\[
\begin{align*}
\text{NUMBER} & := 0; \\
\text{ATMOST} & := \left\lceil 20 \cdot \sqrt{3\pi n} \cdot \left(\frac{4}{3}\right)^n \right\rceil; \\
\text{FOUND} & := \text{FALSE}; \\
\end{align*}
\]

{The variable \( \text{NUMBER} \) counts the number of random samples executed. \( \text{ATMOST} \) gives the upper bound on the number of random samples we are willing to perform. \( \text{FOUND} \) indicates whether or not an assignment satisfying \( F \) was already found.}

**Step 2:**

\[
\begin{align*}
\text{while } \text{NUMBER} < \text{ATMOST} \text{ and } \text{FOUND} = \text{FALSE} \text{ do} \\
\quad \text{begin} \\
\quad \quad \text{NUMBER} := \text{NUMBER} + 1; \\
\quad \quad \text{Generate at random an assignment } \alpha \in \{0, 1\}^n; \\
\quad \quad \text{if } F \text{ is satisfied by } \alpha \text{ then } \text{FOUND} := \text{TRUE}; \\
\quad \text{end}
\end{align*}
\]

\(^{15}|F|\) denotes the length of the formula \( F \), i.e., the input length. Clearly, \( n \leq |F| \).
\[ M := 0; \]
\[ \text{while } M < 3 \cdot n \text{ and } \text{FOUND} = \text{FALSE do} \]
\[ \text{begin} \]
\[ M := M + 1; \]
\[ \text{Find a clause } C \text{ that is not satisfied by } \alpha. \]
\{ Since \( \alpha \) does not satisfy \( F \), such a clause must exist. If there are several such clauses, it does not matter which one is chosen. \}
\[ \text{Pick one of the literals of } C \text{ at random, and flip the value of its variable in order to get a new assignment } \alpha. \]
\{ Observe that \( \alpha \) now satisfies the clause \( C. \) \}
\[ \text{if } \alpha \text{ satisfies } F \text{ then } \text{FOUND} := \text{TRUE;} \]
\[ \text{end} \]

**Step 3:**

\[ \text{if } \text{FOUND} = \text{TRUE then} \]
\[ \text{output } "F \text{ is satisfiable}" \]
\[ \text{else} \]
\[ \text{output } "F \text{ is not satisfiable}". \]

The main difficulty is to show that random sampling with an enclosed short, randomized local search has a substantially higher probability of success (of finding an assignment satisfying \( F \), assuming \( F \) is satisfiable) than the pure random sampling.

**Theorem 5.3.10** The algorithm SCHÖNING is a 1MC algorithm for the 3SAT-problem that runs in time
\[ O\left( |F| \cdot n^{3/2} \cdot \left( \frac{4}{3} \right)^n \right) \]
for any instance \( F \) of \( n \) variables.

**Proof.** First, we analyze the time complexity in the worst case. Step 1 can be performed in time \( O(n^2 \cdot \log_2 n) \) by repeated squaring\(^{16}\), and step 3 requires only \( O(1) \) operations. The random sampling accompanied with the local search described above is executed at most
\[ \left\lceil 20 \cdot \sqrt{3\pi n} \cdot \left( \frac{4}{3} \right)^n \right\rceil \]
times. Each local search consists of at most \( 3 \cdot n \) steps, and each step can be performed in time\(^{17}\) \( O(|F|) \). Hence, the time complexity of the second step (and so the time complexity of the whole algorithm) is in

\(^{16}\)see Appendix, Section A.2

\(^{17}\)A formula \( F \) can be evaluated in time \( O(|F|) \) for every given assignment. During the evaluation procedure, one fixes which clauses are satisfied and which ones are not.
In order to show that SCHÖNING is a 1MC algorithm for 3SAT, we analyze its failure probability. As usual, we distinguish two cases with respect to the satisfiability of $F$.

Let $F$ be not satisfiable. Then, the algorithm SCHÖNING does not find any assignment that satisfies $F$ and outputs the correct answer “$F$ is not satisfiable” with certainty.

Let $F$ be satisfiable. To prove a lower bound on the success probability of SCHÖNING, we proceed in a way similar to that in Section 5.2; namely we analyze the probability of finding a certain assignment $\alpha^*$ that satisfies $F$. Let $p$ be the probability that the algorithm SCHÖNING finds $\alpha^*$ by a random sample followed by at most $3 \cdot n$ steps of the local search described above. Our first aim is to show that

$$p \geq \frac{1}{2} \cdot \sqrt{3\pi n} \cdot \left(\frac{3}{4}\right)^n.$$  \hspace{1cm} (5.7)

We start by partitioning the set $\{0, 1\}^n$ of all assignments of $F$ into $n + 1$ classes with respect to their distances to $\alpha^*$. Let $\alpha$ and $\beta$ be two assignments. We define the distance $\text{Dist}(\alpha, \beta)$ between $\alpha$ and $\beta$ as the number of bits in which they differ (i.e., as the number of flip operations necessary to get from $\alpha$ to $\beta$, or vice versa). For every $j = 0, 1, 2, \ldots, n$ we define the $j$-th class as

$$\text{Class}(j) = \{\beta \in \{0, 1\}^n | \text{Dist}(\alpha^*, \beta) = j\}.$$  

These $n + 1$ classes are pairwise disjoint and so they build a partitioning of $\{0, 1\}^n$. Clearly,

$$\text{Class}(0) = \{\alpha^*\},$$

and

$$|\text{Class}(j)| = \binom{n}{j}$$

for $j = 0, 1, \ldots, n$. The kernel of our analysis is to investigate the execution of the local search as a run (movement) between these classes, i.e., to mimic the local search by a sequence of corresponding classes. To visualize this we use the graph in Figure 5.4. The vertex $i$ of this graph corresponds to the class $\text{Class}(i)$ for $i = 0, 1, \ldots, n$, and we say that the algorithm is situated in the vertex $i$ of the current assignment, $\alpha$ is in $\text{Class}(i)$. Our goal is to achieve the vertex 0. Every local step\footnote{performed by flipping a bit of the current assignment} changes exactly one bit of the assignment, and so corresponds to the movement to one of the two neighboring vertices. Hence, executing a local step one can approach the vertex 0 by decreasing the distance by exactly 1, or one increases the distance to the vertex 0 by 1. The only exceptions are vertices 0 and $n$. One can move from vertex $n$ only in the
5.3 Repeated Random Sampling and Satisfiability

Fig. 5.4.

... direction of vertex 0 and we do not want to leave vertex 0 anymore.\(^{19}\) Now, we are claiming that, for \(j = 1, 2, \ldots, n - 1\),

\[
\text{the probability of moving from vertex } j \text{ to vertex } j - 1 \text{ in one local search step is at least } \frac{1}{3}. \tag{5.8}
\]

This can be shown as follows. Let \(\alpha\) be the current assignment from Class \((j)\) and let \(C\) be a clause that is not satisfied by \(\alpha\). The clause \(C\) contains at most 3 literals and so \(C\) depends on at most 3 variables. The assignment \(\alpha\) does not satisfy any of these literals. Since \(\alpha^*\) satisfies \(F\), and so \(C\), too, \(\alpha^*\) must differ from \(\alpha\) in at least one of these variables of \(C\). The algorithm SCHÖNING chooses one of these variables for flipping at random, and so one has probability of at least \(1/3\) to approach \(\alpha^*\) (i.e., to decrease the distance to \(\alpha^*\) by 1). Hence, the complementary probability\(^{20}\) of moving away from \(\alpha^*\) (of increasing the distance from \(\alpha^*\) by 1) is at most \(2/3\).

In what follows, we analyze for all \(i, j\), with \(i \leq j \leq n\), the probability \(q_{j,i}\) that the algorithm SCHÖNING starting in the class Class \((j)\) (in the vertex \(j\)) reaches \(\alpha^*\) (the vertex 0) in exactly \(j + 2 \cdot i\) local steps, or more precisely in \(j + i\) steps toward \(\alpha^*\) and \(i\) steps in the opposite direction (i.e., toward the vertex \(n\)). Since

\[
j + 2 \cdot i \leq 3 \cdot n,
\]

such a run of the algorithm is possible. To establish a lower bound on \(q_{j,i}\), we describe the movement of the algorithm during the \(j + 2 \cdot i\) steps on the graph in Figure 5.4 by a word (string) in \(\{+,-\}^{j+2 \cdot i}\). A move toward \(\alpha^*\) (from the left to the right in Figure 5.4) is represented by +, and a move in the opposite direction (from the right to the left in Figure 5.4) is represented by -. The number of words over \(\{+,-\}\) of length \(j + 2 \cdot i\) with exactly \(i - \) symbols is

\[
\binom{j + 2 \cdot i}{i}.
\]

But not every one of these strings corresponds to a possible computation from vertex \(j\) to vertex 0 in exactly \(j + 2 \cdot i\) steps. Only those words correspond to possible runs for which every suffix contains at least as many + symbols as

\(^{19}\)If the algorithm achieves vertex 0, then it halts.

\(^{20}\)As already observed, there is no possibility of remaining in the class Class \((j)\).
− symbols. The number of such words is at least one third\(^\text{21}\) of the number of strings from \(\{+,−\}\)^{\(j+2\cdot i\)} with exactly \(i+j\) symbols + and so there are at least
\[
\frac{1}{j+2i} \cdot \binom{j+2\cdot i}{i}
\]  
(5.9)
such words. Let \(w\) be an arbitrary word of this set. The word \(w\) can be viewed as the representation of the event \(\text{Event}(w)\) containing all computations\(^\text{22}\) whose movement in the graph is described by \(w\). Since the symbol + occurs with a probability of at least \(1/3\) and the − symbol occurs with a probability of at most \(2/3\),
\[
\text{Prob}(\text{Event}(w)) \geq \left(\frac{1}{3}\right)^{j+i} \cdot \left(\frac{2}{3}\right)^i
\]  
(5.10)
for each \(w\) of the string set considered. Combining (5.9) and (5.10), we obtain
\[
q_{j,i} \geq \frac{1}{j+2i} \cdot \binom{j+2\cdot i}{i} \cdot \left(\frac{1}{3}\right)^{j+i} \cdot \left(\frac{2}{3}\right)^i.
\]  
(5.11)

For \(j = 0,1,2,\ldots,n\), let \(q_j\) be the probability of reaching \(\alpha^*\) from an \(\alpha \in \text{Class}(j)\) in at most \(3 \cdot n\) local steps. Since \(j + 2 \cdot i \leq 3 \cdot n\) for all \(0 \leq i \leq j \leq n\), one obtains
\[
q_j \geq \sum_{i=0}^{j} q_{j,i}
\]  
(5.12)
for each \(j \in \{1,2,\ldots,n\}\). Note that \(q_0 = 1\). Inserting (5.11) in (5.12), we obtain
\[
q_j \geq \sum_{i=0}^{j} \left[ \frac{1}{j+2i} \cdot \binom{j+2\cdot i}{i} \cdot \left(\frac{1}{3}\right)^{j+i} \cdot \left(\frac{2}{3}\right)^i \right]
\]
\[
> \frac{1}{3} \cdot \binom{3 \cdot j}{j} \cdot \left(\frac{1}{3}\right)^{2 \cdot j} \cdot \left(\frac{2}{3}\right)^{j}
\]
\{\text{We took the element with } i = j \text{ of the sum as a lower bound for the whole sum.}\}

Inserting the Stirling formula (Section A.3)
\[
r! = \sqrt{2\pi r} \left(\frac{r}{e}\right)^r \cdot \left(1 + \frac{1}{12r} + O\left(\frac{1}{r^2}\right)\right) \sim \sqrt{2\pi r} \cdot \left(\frac{r}{e}\right)^r,
\]
\(^\text{21}\) We show in Lemma A.3.69 that the number of such words is exactly
\[
\binom{j+2i-1}{i} - \binom{j+2i-1}{i-1} = \binom{j+2i}{i} \cdot \frac{1}{j+2i}.
\]
\(^\text{22}\) as elementary events
we obtain

\[ q_j \geq \frac{1}{3} \cdot \frac{(3 \cdot j)!}{(2 \cdot j)! \cdot j!} \cdot \left( \frac{1}{3} \right)^{2j} \cdot \left( \frac{2}{3} \right)^j \]

\[ \geq \frac{1}{3} \cdot \frac{\sqrt{2\pi \cdot 3^j \cdot \left( \frac{3j}{e} \right)^{3j}}}{\sqrt{2\pi \cdot 2^j} \cdot \left( \frac{2j}{e} \right)^{2j} \cdot \sqrt{2\pi} \cdot \left( \frac{4}{e} \right)^j} \cdot \left( \frac{1}{3} \right)^{2j} \cdot \left( \frac{2}{3} \right)^j \]

\[ = \frac{1}{3} \cdot \frac{\sqrt{3}}{2 \cdot \sqrt{\pi} j} \cdot \left( \frac{1}{3} \right) \cdot \left( \frac{2}{3} \right)^j \]

\[ = \frac{1}{2} \cdot \sqrt{3\pi} j \cdot \left( \frac{1}{2} \right)^j. \quad (5.13) \]

Since the random samples (initial assignments) are chosen uniformly, the cardinalities of the classes Class \((j)\) determine the probability \(p_j\), that a random sample is in Class \((j)\), i.e.,

\[ p_j = \binom{n}{j} / 2^n. \quad (5.14) \]

Thus, we obtain

\[ p = \text{Prob} \left( \text{SCHÖNING finds } \alpha^* \text{ by a random sample followed by a local search of at most } 3n \text{ steps} \right) \]

\[ \geq \sum_{j=0}^{n} p_j \cdot q_j. \quad (5.15) \]

Inserting (5.13) and (5.14) into (5.15), we get

\[ p > \sum_{j=0}^{n} \left[ \binom{1}{2} \cdot \binom{n}{j} \cdot \left( \frac{1}{2} \cdot \sqrt{\frac{3\pi}{j}} \cdot \left( \frac{1}{2} \right)^j \right) \right] \]

\[ \geq \frac{1}{2 \cdot \sqrt{3\pi} n} \cdot \left( \frac{1}{2} \right)^n \cdot \sum_{j=0}^{n} \left[ \binom{n}{j} \cdot \left( \frac{1}{2} \right)^j \right] \]

\[ = \frac{1}{2 \cdot \sqrt{3\pi} n} \cdot \left( \frac{1}{2} \right)^n \cdot \left( 1 + \frac{1}{2} \right)^n \]

\[ \{ \text{One can write } \binom{n}{j} \cdot \left( \frac{1}{2} \right)^j \text{ as } \binom{n}{j} \cdot \left( \frac{1}{2} \right)^j \cdot 1^{n-j} \text{ and then apply the binomial formula.} \} \]

\[ = \frac{1}{2 \cdot \sqrt{3\pi} n} \cdot \left( \frac{3}{4} \right)^n = \tilde{p}. \quad (5.16) \]

Thus, the probability that the algorithm does not find any assignment satisfying \(F\) by one random sample followed by the considered local search is at most

\[ 1 - \tilde{p}. \]
Therefore, the error probability after $t$ independent attempts of the algorithm is at most\footnote{Lemma A.3.59}
\[
(1 - \tilde{p})^t \leq e^{-\tilde{p} t}.
\]
(5.17)

Taking $t = \text{ATMOST} = 20 \cdot \sqrt{3\pi n} \cdot \left(\frac{4}{3}\right)^n$ and together with (5.16) inserting it into (5.17), one obtains
\[
\text{Error}_{\text{SCHÖNING}}(F) \leq (1 - \tilde{p})^t \leq e^{-10} < 5 \cdot 10^{-5}.
\]

Thus, we have proved that the algorithm SCHÖNING is a one-sided-error Monte Carlo algorithm for 3SAT.

We have seen that a clever execution of random sampling can essentially help reduce the number of random samples. The idea presented here can also be extended for the $k$SAT problem for an arbitrary, fixed positive integer $k$.

**Exercise 5.3.11.** Extend the algorithm SCHÖNING for 4SAT. Observe that the lower bound on the probability of moving toward $\alpha^*$ in a local step decreases to $1/4$ in this case. How many repetitions of random sampling followed by a local search are necessary to get a constant success probability?

**Exercise 5.3.12** Prove, for every positive integer $k \geq 5$, that the concept of Schöning provides a 1MC-algorithm for $k$SAT that works in time
\[
O\left(|F| \cdot P(n) \cdot \left(2 - \frac{2}{k}\right)^n\right)
\]
for a polynomial $P$.

### 5.4 Random Sampling and Generating Quadratic Nonresidues

In Section 5.3 we indicated that it is not realistic to try to design bounded-error randomized polynomial-time algorithms for NP-hard problems. But this is far from the claim that there do not exist any randomized polynomial-time algorithms for problems that cannot be solved in deterministic polynomial time. The world between the class P of decision problems solvable in deterministic polynomial time and the class of NP-hard problems can be very rich of problems of distinct difficulty. There are several problems for which

(i) one does not know any deterministic polynomial-time algorithm, and

(ii) there are no proofs presenting their NP-hardness.
A problem of this kind and of enormous theoretical as well as practical importance is the factorization of natural numbers. Though the best known deterministic algorithms for the factorization run in exponential time, most researchers believe that factorization is not NP-hard. Since we consider the class of problems solvable by bounded-error randomized polynomial-time algorithms instead of the class P as the class of practically solvable problems, the research focuses on the design of efficient randomized algorithms for problems outside of P, but not NP-hard. One does not know any randomized polynomial-time algorithm for factorization, but there are other problems with properties (i) and (ii) for which efficient randomized algorithms were discovered. An example already presented is the 1MC algorithm for the comparison of two polynomials that is based mainly on the fingerprinting method. Here, we go even further and design an efficient Las Vegas algorithm for a problem with properties (i) and (ii).

In what follows, we consider the problem of generating a quadratic nonresidue modulo \( p \) for a given prime \( p \). A quadratic residue in the field \( \mathbb{Z}_p \) is any element \( a \in \mathbb{Z}_p \) such that
\[
a = x^2 \mod p
\]
for an \( x \in \mathbb{Z}_p \).

A quadratic nonresidue is any number \( b \in \mathbb{Z}_p \) such that
\[
d^2 \not\equiv b \pmod{p}
\]
for all \( d \in \mathbb{Z}_p \), i.e., every element of \( \mathbb{Z}_p \) that is not a quadratic residue. The problem of our current interest is the following one:

**Generation of a quadratic nonresidue**

*Input:* A prime \( p > 2 \).

*Output:* A quadratic nonresidue modulo \( p \).

Usually one considers this problem for primes that are several hundreds of digits long. Hence, it is not possible to check all\(^{26}\) elements of \( \mathbb{Z}_p \), whether they are quadratic residues or not. Note that the size of the input \( p \) is the length \( \lceil \log_2(p+1) \rceil \) of its binary representation and so \( |\mathbb{Z}_p| = p \) is exponential in input size.

---

\(^{24}\)Note that we do know of a polynomial-time quantum algorithm for factorization. Currently we are not able to build a quantum computer working with more than a few bits, but quantum computing can be viewed as a generalization of randomized computations, and this viewpoint opens a lot of interesting questions about the limits of physically based computers.

\(^{25}\)Note that \( \mathbb{Z}_p \) is a field iff \( p \) is a prime (a direct consequence of Theorem A.2.27).

\(^{26}\)their number is larger than the number of protons in the known universe.
It is also remarkable that the “dual” problem of generating a quadratic residue is trivial. It takes an arbitrary element $x$ of $\mathbb{Z}_p$ and computes the quadratic residue $x^2 \mod p$. Moreover, 1 is a quadratic residue for every prime $p$.

**Exercise 5.4.13.** Determine all quadratic nonresidues for

(i) $p = 5$,
(ii) $p = 11$,
(iii) $p = 17$.

We aim to show that by applying random sampling one can solve this problem efficiently. One takes a few samples from $\{1, 2, \ldots, p-1\}$ at random, and there is a quadratic nonresidue among these samples with a reasonably high probability. To convince the reader that this very simple idea really works, we have to prove the following two facts:

(A) For every prime $p$ and every $a \in \mathbb{Z}_p$, one can efficiently decide (in a deterministic way) whether $a$ is a quadratic residue or a quadratic nonresidue modulo $p$.

(B) For every prime $p$ exactly half of the elements of $\mathbb{Z}_p - \{0\}$ are quadratic nonresidues, i.e., a random sample from $\{1, 2, \ldots, p-1\}$ provides a quadratic nonresidue with probability $1/2$.

First, we present Euler’s criterion in order to prove (A). In what follows we also use the symbol $-1$ to denote $p - 1$ as the inverse element to 1 with respect to $\oplus \mod p$ in $\mathbb{Z}_p$. All notions and results of the number theory used in what follows are presented in detail in Section A.2.

**Theorem 5.4.14. Euler’s Criterion**

Let $p$, with $p > 2$, be a prime. For every $a \in \{1, 2, \ldots, p-1\}$,

(i) if $a$ is a quadratic residue modulo $p$, then

$$a^{(p-1)/2} \equiv 1 \pmod{p},$$

and

(ii) if $a$ is a quadratic nonresidue modulo $p$, then

$$a^{(p-1)/2} \equiv p - 1 \pmod{p}.$$ 

**Proof.** Following Fermat’s Little Theorem (Theorem A.2.28)

$$a^{p-1} \equiv 1 \pmod{p},$$

i.e.,

$$a^{p-1} - 1 \equiv 0 \pmod{p} \tag{5.18}$$

with respect to the uniform distribution.
5.4 Random Sampling and Generating Quadratic Nonresidues

for all \( a \in \{1, 2, \ldots, p - 1\} \).

Since \( p > 2 \) and \( p \) is odd, there is \( a^{28} p' \geq 1 \) such that
\[
p = 2 \cdot p' + 1.
\] (5.19)

Inserting (5.19) into (5.18), one obtains
\[
a^{p-1} - 1 = a^{2p'} - 1 = (a^{p'} - 1) \cdot (a^{p'} + 1) \equiv 0 \pmod{p}.
\] (5.20)

If a product of two integers is divisible by a prime, then one of the factors must be divisible\(^{29}\) by \( p \). Therefore (5.20) implies
\[
a^{(p-1)/2} - 1 \equiv 0 \pmod{p} \text{ or } a^{(p-1)/2} + 1 \equiv 0 \pmod{p},
\]
and so
\[
a^{(p-1)/2} \equiv 1 \pmod{p} \text{ or } a^{(p-1)/2} \equiv -1 \pmod{p}.
\]

In this way we have established that
\[
a^{(p-1)/2} \pmod{p} \in \{1, p - 1\}
\] (5.21)
for every \( a \in \{1, 2, \ldots, p - 1\} \). Now, we are ready to prove (i) and (ii).

(i) Let \( a \) be a quadratic residue modulo \( p \).

Then there exists an \( x \in \mathbb{Z}_p^* \) such that
\[
a \equiv x^2 \pmod{p}.
\]

Since Fermat’s Little Theorem implies\(^{30}\) \( x^{p-1} \equiv 1 \pmod{p} \), we obtain
\[
a^{(p-1)/2} \equiv (x^2)^{(p-1)/2} \equiv x^{p-1} \equiv 1 \pmod{p}.
\]

(ii) Let \( a \) be a quadratic nonresidue modulo \( p \).

Following (5.21) it is sufficient to show that
\[
a^{(p-1)/2} \pmod{p} \neq 1.
\]

Since \( (\mathbb{Z}_p^*, \cdot \pmod{p}) \) is a cyclic group,\(^{31}\) there exists a generator \( g \) of \( \mathbb{Z}_p^* \).

Since \( a \) is a quadratic nonresidue, \( a \) must be an even power of \( g \), i.e.,
\[
a = g^{2l+1} \pmod{p}
\]
for an integer \( l \geq 0 \). Hence,

\(^{28}\) \( p' = (p - 1)/2 \)

\(^{29}\) This is a direct consequence of the Fundamental Theorem of Arithmetics about the unambiguosity of factorization (Theorem A.2.3).

\(^{30}\) Theorem A.2.28

\(^{31}\) Recall that \( \mathbb{Z}_p - \{0\} = \mathbb{Z}_p^* \) for every prime \( p \).
Success Amplification and Random Sampling

\[ a^{(p-1)/2} \equiv (g^{2^l+1})^{(p-1)/2} \equiv g^{l \cdot (p-1)} \cdot g^{(p-1)/2} \pmod{p}. \] (5.22)

The Fermat’s Little Theorem implies \( g^{p-1} \pmod{p} = 1 \), and so

\[ g^{l \cdot (p-1)} \equiv (g^{p-1})^l \equiv 1^l \equiv 1 \pmod{p}. \] (5.23)

Inserting (5.23) into (5.22), we obtain

\[ a^{(p-1)/2} \equiv g^{(p-1)/2} \pmod{p}. \]

Since \( g \) is a generator of \((\mathbb{Z}_p^*, \odot \pmod{p})\), the order of \( g \) is \( p \), and so

\[ g^{(p-1)/2} \pmod{p} \neq 1. \]

Thus, \( a^{(p-1)/2} \pmod{p} \neq 1 \), too and, following (5.21), we obtain

\[ a^{(p-1)/2} \pmod{p} = -1 = p - 1. \]

Euler’s Criterion provides a simple way for testing whether an element \( a \in \{1, 2, \ldots, p-1\} \) is a quadratic residue or not. It is sufficient to compute the number

\[ a^{(p-1)/2}. \]

We know that this number can be computed in \( O(\log_2 p) \) operations\(^{32}\) over \( \mathbb{Z}_p \) by the method of repeated squaring (Section A.2). In this way, the proof of (A) is completed.

The following theorem proves assertion (B).

**Theorem 5.4.15.** For every odd prime\(^{33}\) \( p \), exactly half of the nonzero elements of \( \mathbb{Z}_p^* \) are quadratic residues modulo \( p \).

**Proof.** Let

\[ \text{Quad}(p) = \{1^2 \pmod{p}, 2^2 \pmod{p}, \ldots, (p-1)^2 \pmod{p}\} \]

be the set of all quadratic nonresidues in \( \mathbb{Z}_p^* \). Since every element from \( \mathbb{Z}_p^* = \{1, 2, \ldots, p-1\} \) is either a quadratic residue or a quadratic nonresidue, it is sufficient to show that

\[ |\text{Quad}(p)| = \frac{(p-1)}{2}. \] (5.24)

We prove the equality (5.24) by separately proving the inequalities\(^{34}\)

\[ |\text{Quad}(p)| \leq \frac{(p-1)}{2} \quad \text{and} \quad |\text{Quad}(p)| \geq \frac{(p-1)}{2}. \]

\(^{32}\)in \( O((\log_2 p)^3) \) binary operations

\(^{33}\)i.e., for every prime \( o \geq 3 \)

\(^{34}\)Note that for solving our problem by random sampling, it is sufficient to prove the inequality \( |\text{Quad}(p)| \leq \frac{(p-1)}{2} \).
5.4 Random Sampling and Generating Quadratic Nonresidues

(i) For every \( x \in \{1, 2, \ldots, p - 1\} \),
\[
(p - x)^2 = p^2 - 2 \cdot p \cdot x + x^2 = p \cdot (p - 2 \cdot x) + x^2 \equiv x^2 \pmod{p}.
\]
Therefore,
\[
|\text{Quad}(p)| \leq \frac{(p - 1)}{2}.
\]

(ii) For the opposite inequality, it is sufficient to show that the congruence
\[
x^2 \equiv y^2 \pmod{p} \tag{5.25}
\]
has at most one solution \( y \in \{1, 2, \ldots, p - 1\} \) different from \( x \). Without loss of generality, we assume \( y > x \), i.e., \( y = x + i \) for an \( i \in \{1, 2, \ldots, p - 2\} \).
Since
\[
y^2 = (x + i)^2 = x^2 + 2 \cdot i \cdot x + i^2,
\]
the congruence (5.25) implies
\[
2 \cdot i \cdot x + i^2 = i \cdot (2 \cdot x + i) \equiv 0 \pmod{p}.
\]
Since \( \mathbb{Z}_p \) is a field, and \( i \not\equiv 0 \pmod{p} \),
\[
2 \cdot x + i \equiv 0 \pmod{p}. \tag{5.26}
\]
Since \( i = -(2x) \) is the only solution of the congruence (5.26),
\[
|\text{Quad}(p)| \geq \frac{(p - 1)}{2}.
\]

Now, we are ready to present our algorithm for generating a quadratic nonresidue in \( \mathbb{Z}_p \) for any given prime \( p \).

**Algorithm NQUAD**

*Input:* A prime \( p \).

*Step 1:* Choose uniformly an \( a \in \{1, 2, \ldots, p - 1\} \) at random.

*Step 2:* Compute
\[
A := a^{(p-1)/2} \pmod{p}
\]
by the method of repeated squaring.

---

35 More precisely, if the product of two elements in a field is 0, then one of the elements must be 0, too. But here one can argue in a simpler way without using algebra. If a product of two integers is divisible by a prime \( p \), then one of the factors must be divisible by \( p \) (Corollary A.2.4).

36 Every element \( i \) of the group \( (\mathbb{Z}_p, + \pmod{p}) \) has exactly one inverse element with respect to \( + \pmod{p} \). If \( 2x < p \), then \( i = p - (2 \cdot x) \). If \( 2 \cdot xp \), then \( i = -(2 \cdot x - p) \).
Step 3:

if $A = p - 1$ then
output “a”
else
output “?”

The above proved claims (A) and (B) imply that

(i) NQUAD does not make any error$^{37}$, and
(ii) NQUAD finds$^{38}$ a quadratic nonresidue with the probability $1/2$.

Hence, NQUAD is a Las Vegas algorithm for generating a quadratic nonresidue. Its running time is in $O(\log_2 p)$ when taking one time unit for executing an operation over $\mathbb{Z}_p$ and in $O((\log_2 p)^3)$ when counting the number of binary operations.

Exercise 5.4.16. Modify the algorithm NQUAD in such a way that it always halts$^{39}$ with a quadratic nonresidue (i.e., the answer “?” never appears). Analyze the expected running time of your modified NQUAD and prove that the probability of executing an infinite computation is 0.

Exercise 5.4.17. Let $p$ be a prime and let $k \leq \log_2 p$ be a positive integer. Design an efficient Las Vegas algorithm that computes $k$ pairwise distinct quadratic nonresidues modulo $p$.

We have shown above that the problem of generating quadratic nonresidues is not hard because it can be solved efficiently by a randomized algorithm. The main reason for this success is that half the elements of $\{1, 2, \ldots, p - 1\}$ are quadratic nonresidues. Hence, one could think that it is possible to find a quadratic nonresidue in $\mathbb{Z}_p - \{0\}$ efficiently by a suitable deterministic search in $\{1, 2, \ldots, p - 1\}$. Unfortunately, until now no one has been able to discover a deterministic polynomial-time strategy for finding a quadratic nonresidue modulo $p$ in $\{1, 2, \ldots, p - 1\}$. If there does not exist any efficient deterministic algorithm for this task, one could say that the distribution of quadratic nonresidues in $\{1, 2, \ldots, p - 1\}$ can be considered random$^{40}$ (chaotic). On the other hand, we have to mention that there is an indication for the existence of some structure (order) in $\mathbb{Z}_p$. It is well known that if the famous extended Riemann’s Hypothesis holds, then $\mathbb{Z}_p$ must contain a quadratic nonresidue among its $O((\log_2 p)^2)$ smallest elements, i.e., a quadratic nonresidue can be found deterministically by simply checking all these smallest elements of $\mathbb{Z}_p$.

---

$^{37}$This is a direct consequence of the Euler's criterion (Theorem 5.4.14).
$^{38}$This is a consequence of claim (B) (Theorem 5.4.15).
$^{39}$if it halts at all
$^{40}$Recall the discussion about the random distribution of witnesses in the set of witness candidates.
5.5 Summary

In this chapter we have presented amplification not only as a method for reducing the error probability of already designed randomized algorithms, but also as a method that can be the core of algorithm design. The design of randomized algorithms for the MIN-CUT problem exemplifies this point of view on amplification. We have seen that it can be promising to exchange some parts of randomized computations having high error probability with deterministic ones. Repeating different parts of a randomized computation differently many times with respect to their success probability may also save a lot of computer work, and so be very helpful. Both these approaches lead to the design of randomized algorithms for the MIN-CUT problem that are more efficient than any naive application of amplification based on repeating the entire computations on a given input.

Some exponential algorithms are not only successfully applicable for realistic input sizes, but, for some applications, even more efficient than polynomial-time algorithms whose running time is bounded by a polynomial of a higher degree. Therefore it is meaningful to try to design algorithms with running time in \( O(p(n) \cdot c^n) \) for a polynomial \( p \) and a constant \( c < 2 \). Combining random sampling with local search we have designed an efficient randomized algorithm for 3SAT that, for any satisfiable formula, finds a satisfying assignment with probability at least \( \Omega \left( \frac{1}{2} \cdot \left( \frac{4}{3} \right)^n \right) \). Executing \( O(n \cdot \left( \frac{4}{3} \right)^n) \) independent runs of this algorithm, one gets a Monte Carlo algorithm for 3SAT. The essential point of this algorithm design is that pure random sampling cannot assure success probability greater than \( 2^{-n} \). Extending random sampling by a suitable local search increases the success probability to \( n^{-1/2} \cdot (3/4)^{-n} \).

There are computing tasks for which on the one hand no deterministic polynomial-time algorithms are known, and, on the other hand, no one is able to prove that these problems are NP-hard. This world between P and NP-hardness is the central area for applying randomization. Famous examples of problems in this world are factorization of integers, equivalence of two polynomials, and the generation of quadratic nonresidues. An efficient one-sided-error Monte Carlo algorithm for the equivalence of two polynomials has been presented in Chapter 4. Here, applying random sampling, we have even designed an efficient Las Vegas algorithm for generating a quadratic nonresidue. The method of random sampling is especially successful in situations where one searches for an object with some special properties in a set of objects, in which the objects of interest are randomly distributed and there is an abundance of them. It may also happen that the objects of our interest are not randomly distributed in the set of all objects (i.e., the set has some structure), but we are not able to recognize the order of their distribution. Also, in this case of our poor knowledge, applying random sampling is the best we can do.

Further successful applications of the methods of random sampling and amplification are presented in the comprehensive textbook of Motwani and
Raghavan [MR95]. Karger [Kar93] designed the algorithm CONTRACTION for MIN-CUT, and the most efficient version was developed by Karger and Stein [KS93]. The exponential randomized algorithm for 3SAT is due to Schöning [Sch99]. The Las Vegas algorithm for generating quadratic non-residues was used by Adleman, Manders, and Miller [AMM77] as a subroutine for computing roots of quadratic residues.