When events are dependent, then knowledge about the occurrence changes the probability of the other. This is made clear by the definition of *conditional probabilities.*

Definition 1.9. *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space and let $A, B \in 2^{\Omega}$ be events with $\mathbf{Pr}(B) > 0$. The* conditional probability of A given B *is defined as*

$$\mathbf{Pr}(A \mid B) = \frac{\mathbf{Pr}(A \cap B)}{\mathbf{Pr}(B)}.$$

The definition is consistent with our notion of independent events: If $A, B \in 2^{\Omega}$ with $\mathbf{Pr}(B) > 0$ are independent, then

$$\mathbf{Pr}(A \mid B) = \frac{\mathbf{Pr}(A \cap B)}{\mathbf{Pr}(B)} = \frac{\mathbf{Pr}(A) \cdot \mathbf{Pr}(B)}{\mathbf{Pr}(B)} = \mathbf{Pr}(A).$$

The occurrence of B does not influence the probability of A, thus the conditional probability $\mathbf{Pr}(A \mid B)$ is just $\mathbf{Pr}(A)$.

Example 1.10. *We continue Example 1.8. Events C and E are not independent. We argued that E influences C because its occurrence ensures that we did not roll a 1. Indeed, the conditional probability of C given E is greater than $\mathbf{Pr}(C)$:*

$$\mathbf{Pr}(C \mid E) = \frac{\mathbf{Pr}(C \cap E)}{\mathbf{Pr}(E)} = (1/36)/(1/6) = \frac{1}{6} > \frac{5}{36} = \mathbf{Pr}(C).$$

In Example 1.10, we computed the conditional probability from the data that we are given by the modeling of our random experiment. However, if the conditional probability of an event is *part of the modeling*, then we can use Definition 1.9 reversely to compute $\mathbf{Pr}(A \cap B)$ or $\mathbf{Pr}(B)$. We see an example for that as well.

Example 1.11. *Assume that we know the following facts about a group of people.*

- *In this group of people, 30% play the piano.*
- *Among the piano players, 90% like the composer Bach.*
- *Among the people that like Bach, 65% play the piano.*

Question: How many people in the whole group like Bach?

We define the events B for 'a person likes Bach' and P for 'a person plays the piano'. From the data given to us, we know that $\mathbf{Pr}(P) = 0.3$, $\mathbf{Pr}(B \mid P) = 0.9$ and $\mathbf{Pr}(P \mid B) = 0.65$ (if we choose a person uniformly at random from the group). By the definition of conditional probability,

$$\mathbf{Pr}(B \cap P) = \mathbf{Pr}(P) \cdot \mathbf{Pr}(B \mid P) = 0.27 \textit{ and}$$
$$\mathbf{Pr}(B) = \frac{\mathbf{Pr}(B \cap P)}{\mathbf{Pr}(P \mid B)} = \frac{0.27}{0.65} \approx 0.42.$$

Thus, 42% of the people in this group like the composer Bach.

Now we want to reduce the error probability of our polynomial tester by repetitions. Independent repetitions can be modeled by product spaces. Recall the definition of product spaces from 1.6. A product space of two probability spaces can be used to model two independent runs of an algorithm. We can verify that events that correspond to different components of this product space are always independent.

Fact 1.12. *Let $(\Omega_1, \mathbf{Pr}_1)$ and $(\Omega_2, \mathbf{Pr}_2)$ be discrete probability spaces and let $(\Omega, \mathbf{Pr})$ be the product space. Then the events $A \times \Omega_2$ and $\Omega_1 \times B$ are independent for all choices of $A \in \Omega_1$ and $B \in \Omega_2$.*

Notice that the definition of product spaces can be easily extended to products of t copies. If $(\Omega_1, \mathbf{Pr}_1), \ldots (\Omega_t, \mathbf{Pr}_t)$ are discrete probability spaces, then the product space is defined by $\Omega = \Omega_1 \times \ldots \times \Omega_t$ and $\mathbf{Pr}(x_1, \ldots, x_t) = \mathbf{Pr}_1(x_1) \cdot \ldots \cdot \mathbf{Pr}_t(x_t)$. Fact 1.6 and Fact 1.12 extend to this slightly more general case.

We will also see an example for an algorithm that is modeled with dependent runs. Then we will use a conditional modeling in the design of the algorithm, so that we can analyze it with conditional probabilities.

Application: Polynomial Tester (Part III: Multiple runs reduce error) We return to our randomized algorithm with one-sided error for the scenario where a black box answers whether $f(x)$ and $g(x)$ are equal for a given value $x \in \mathbb{R}$, and where the black box has no error. We observed that the error probability of the algorithm which chooses one of t values for x uniformly at random is at most d/t. We extend this algorithm. It now chooses n values $x_1, \ldots, x_n$ from $\{1, \ldots, t\}$ uniformly at random. It is possible that the algorithm chooses the same value multiple times. If $f(x_i) \neq g(x_i)$ for at least one $i \in \{1, \ldots, n\}$, then we output no, otherwise, we output yes. Let A_i be the event that x_i satisfies $f(x_i) = g(x_i)$. We know that $\mathbf{Pr}(A_i) \leq d/t$, or, for $t = 100d$, $\mathbf{Pr}(A_i) \leq 1/100$. The event that the algorithm fails is $A_1 \cap \ldots \cap A_n$. The events A_i are independent because they correspond to different components of the product space that models our independent runs. Thus, the failure probability is

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) = \mathbf{Pr}(A_1) \cdot \ldots \cdot \mathbf{Pr}(A_n) \leq \left(\frac{1}{100}\right)^n.$$

This probability decreases exponentially in the number of runs, demonstrating the power of independent runs for reducing the failure probability. Assume that we want to decrease the failure probability below δ, i.e. we want $(\frac{1}{100})^n \leq \delta$. Solving this inequality tells us that we need $(\log \frac{1}{\delta})/(\log 100)$ independent runs to achieve this. For example, reducing the failure probability to 1/100000000, we need 4 runs.

To make the events A_i independent, we had to allow that the randomized algorithm chooses the same value for x more than once. We can decrease the failure probability further by dropping the independence of different runs, at the cost of a more complex analysis. Assume that the algorithm picks x_i uniformly at random from $\{1, \ldots, t\} \backslash \{x_1, \ldots, x_{i-1}\}$ where we assume that $n \leq t$. Define A_i as before, but notice that these events are no longer independent. We rewrite the failure probability by using Definition 1.9:

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) = \mathbf{Pr}(A_n \mid A_1 \cap \ldots \cap A_{n-1}) \cdot \mathbf{Pr}(A_1 \cap \ldots \cap A_{n-1}).$$

Applying the definition of conditional probability iteratively, we get that

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) = \mathbf{Pr}(A_1) \cdot \mathbf{Pr}(A_2 \mid A_1) \cdot \mathbf{Pr}(A_3 \mid A_1 \cap A_2) \cdot \ldots \cdot \mathbf{Pr}(A_n \mid A_1 \cap \ldots \cap A_{n-1}).$$

Let $\mathbf{Pr}(A_i \mid A_1 \cap \ldots \cap A_{i-1})$ be one of the n factors. The condition $A_1 \cap \ldots \cap A_{i-1}$ means that we already chose $i-1$ values, and all of them satisfied $f(x) = g(x)$. Then there are now $i-1$ fewer values with this property available, and the probability that x_i is chosen as one of them is at most $(d-(i-1))/(t-(i-1))$. For $t = 100d$, we get

$$\mathbf{Pr}(A_1 \cap \ldots \cap A_n) \leq \prod_{i=1}^{n} \frac{d-(i-1)}{100d-(i-1)}.$$

This term is always bounded above by $(1/100)^n$. For $n \geq 2$ it is strictly smaller, so the error probability does indeed decrease when we avoid to choose the same values more than once.

1.3 More Applications

1.3.1 The Minimum Cut Problem

This section is solely devoted to one application, *the (global) minimum cut problem.* For this problem, we want to cut a graph into (at least) two pieces (connected components) by deleting as few edges as possible. Let $G = (V, E)$ be an undirected graph. We assume that $n = |V| \geq 2$, $m = |E|$ and that G is connected (otherwise, it already has two connected components).

It is convenient to define a cut based on nodes instead of edges. So let $S \subset V$ with $S \neq \emptyset$ and $S \neq V$ be a true subset of the nodes. To separate S from $V \backslash S$, we need to delete all edges that have one endpoint in S and one endpoint in $V \backslash S$. That are exactly the edges in the set $\delta(S) \subseteq E$ that is defined as follows:

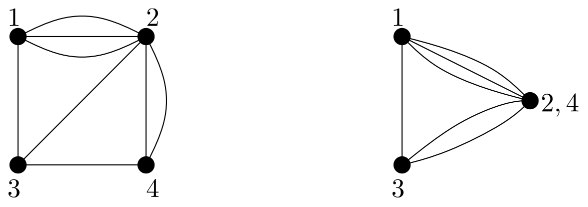
$$\delta(S) = \{\{u, v\} \in E \mid u \in S, v \notin S\}.$$

We want to minimize the number of edges in $\delta(S)$.

Problem 1.13 (MinCut). *Given a graph $G = (V, E)$,* the (global) minimum cut (MinCut) problem *is to compute a set $S \subset V$ with $S \neq \emptyset$ and $S \neq V$ that minimizes $|\delta(S)|$.*

We discuss two randomized algorithms for this problem. Our motivation is two-fold: Both algorithms are conceptually simple with short pseudo code descriptions. Additionally, the second algorithm is very efficient. Of course, deterministic algorithms for the MinCut problem are also known, the best one achieving a running time of $\mathcal{O}(nm + n^2 \log n)$, see [4] and [6]. For dense graphs (where $m \in \Theta(n^2)$), this is $\mathcal{O}(n^3)$. The second randomized algorithm that we learn about achieves a running time of $\mathcal{O}(n^2(\log n)^{\mathcal{O}(1/\delta)})$, where δ is the desired error probability. This is asymptotically faster.

Contracting edges Both algorithms are based on a contraction operation that we name `Contract-Edge(`G, e`)`. For an edge $e = \{u, v\}$ this operation contracts e. It replaces u and v by one node uv. Edges that had either u or v as an endpoint are rerouted to uv, edges between u and v are deleted. The operation can create parallel edges. Formally, this means that the algorithm works with *multipgraphs* where parallel edges are allowed. In the following picture, one of the edges between 2 and 4 is contracted:



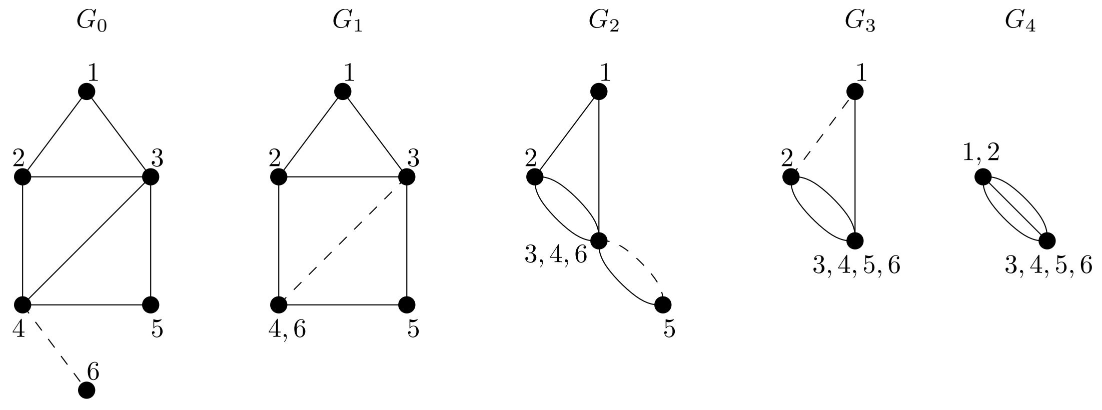
More precisely, the operation `Contract-Edge(`G, e`)` replaces a multigraph G by a multigraph $G' = (V', E')$ with $V' = V\backslash\{u, v\} \cup \{uv\}$ and $E' := \{\{w, uv\} \mid w \notin \{u, v\}, \{w, v\} \in E \vee \{w, u\} \in E\} \cup \{\{w, x\} \mid \{w, x\} \in E, w, x \notin \{u, w\}\}$.

Karger's Contract Algorithm The first randomized algorithm is due to Karger [1]. It does $n - 2$ iterations. In every iteration, it contracts a random edge. The edge is chosen uniformly at random from the edges that are still present. Thus, parallel edges are important: If three parallel edges connect u and v, then u and v are three times as likely to be merged than if only one edge connects them.

To describe the algorithm, we number the nodes arbitrarily, i.e. we assume that $V = \{1, \ldots, n\}$. Then we assign the label $\ell(V) = \{v\}$ to node v for all $v \in V$. The resulting graph with labels is the graph $G_0 = (V_0, E_0)$. Iteration $i \in \{1, \ldots, n - 2\}$ now creates the multigraph $G_i = (V_i, E_i)$. It chooses an edge $e = \{u, v\}$ uniformly at random from the edges in G_{i-1} and calls `Contract-Edge(`G_{i-1}, e`)`. It also sets the label of the new node uv to $\ell(u) \cup \ell(v)$. We get the following pseudocode:

```
Contract($\mathbf{G} = (\mathbf{V}, \mathbf{E})$)
  1. For all $v \in V$: label $v$ with $\ell(v) = \{v\}$
  2. While $|V| > 2$:
  3.     choose $e \in E$ uniformly at random, $e = \{u, v\}$
  4.     Contract-Edge($G, e$)        % creates node $uv$
  5.     Set  $\ell(uv) = \ell(u) \cup \ell(v)$
  6. Let  $G = (\{x, y\}, E')$ be the current graph
  7. Return $\ell(x)$
```

Every iteration reduces the number of nodes by one. The resulting graph G_{n-2} has two nodes, and the labels of the two nodes correspond to two disjoint subsets $S_1, S_2 \subset V$ with $S_1 \cup S_2 = V$. The algorithm outputs one of them (observe that $\delta(S_1) = \delta(S_2)$, so S_1 or S_2 are solutions with the same quality). The following picture shows an example run of the algorithm. The dashed line is the edge that is contracted in the next step.



In this example, the algorithm outputs $\{1, 2\}$ or $\{3, 4, 5, 6\}$, and the value of this solution is three because there are three edges between $\{1, 2\}$ and $\{3, 4, 5, 6\}$ (observe that these three edges exist in G_4, but also in G_3, G_2, G_1 and the original graph G_0). Of course, this example only shows one possible outcome of the algorithm. Since the edges are chosen uniformly at random, every cut is a possible output. However, not all solutions are equally likely. Recall that a pair u, v is more likely to be contracted if a lot of edges connect u and v. This increases the probability that we end up with a good solution.