We could use the same proof to show the following fundamentally different statement.

Fact 2.20. *Let* `SQuickSort` *be the version of* `QuickSort` *where the pivot element is always chosen as* x_1*. If the permutation of the elements is assumed to be chosen uniformly at random from the set of all possible permutations, then the expected number of comparisons that* `SQuickSort` *performs is* $2 \ln n + \Theta(n)$*.*

The randomness is shifted from the algorithm to the input: If we assume that the distribution with which different input permutations occur is uniform, then we can show the same guarantee for a deterministic `QuickSort` variant. This is the type of statement that we will see in the second part of the lecture. However, assuming that the input is a uniformly chosen permutation is a very strong assumption. It is preferable to randomize the algorithm to guarantee the expected performance. Probabilistic analysis is about showing running time bounds for algorithms under much weaker assumptions on the input data.

2.3.2 Randomized Approximation Algorithms

Recall that an α-approximation algorithm for a minimization problem is an algorithm that outputs solutions with a value that is at most α times the value of an optimal solution. Let $\mathfrak{I}$ be the set of all possible input instances to an optimization problem, let ALG be an algorithm that computes a feasible solution $S^{ALG}(I)$ for any $I \in \mathfrak{I}$. For any $I \in \mathfrak{I}$, let $S^*(I)$ be an optimal solution for I. For a minimization problem, suppose that $c(S)$ is the cost of a solution S. Then ALG is an α-approximation algorithm iff

$$\max_{I \in \mathfrak{I}} \frac{c(S^{ALG}(I))}{c(S^*(I))} = \alpha.$$

For a maximization problem, let $v(S)$ be the value of a solution S. In this case, ALG is an α-approximation algorithm iff

$$\max_{I \in \mathfrak{I}} \frac{v(S^*(I))}{v(S^{ALG}(I))} = \alpha.$$

In both cases, we know that $\alpha \geq 1$ since ALG cannot compute solutions that are better than an optimal solution. For randomized algorithms, $c(S^{ALG}(I))$ or $v(S^{ALG}(I))$, respectively, is a random variable. We might either want to achieve that it is close to the optimum value with high probability, or that its expected value is close to the optimum value. We choose the second alternative and will later discuss how to obtain statements of the first type. Thus, we say that a randomized algorithm ALG is an (expected) α-approximation algorithm for a minimization problem iff

$$\max_{I \in \mathfrak{I}} \frac{\mathbf{E}\left[c(S^{ALG}(G))\right]}{c(S^*(G))} = \alpha$$

and it is an (expected) α-approximation algorithm in the case of a maximization problem iff

$$\max_{I \in \mathfrak{I}} \frac{v(S^*(G))}{\mathbf{E}\left[v(S^{ALG}(G))\right]} = \alpha.$$

The MaxCut problem. We consider the *weighted maximum cut* problem, abbreviated as the *MaxCut problem.* We are given an undirected graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, $E \subseteq V \times V$, and a weight function $w : E \to \mathbb{R}^+$ that maps each edge to a positive weight. We are supposed to output a subset $S \subseteq V$ that *maximizes* the total weight of the edges that are cut. In other words, we are supposed to maximize

$$v(S) := \sum_{e \in \delta(S)} w(e).$$

For the min cut problem, we explicitly excluded the cases $S = V$ and $S = \emptyset$. These solutions both cut no edge, so the sum then is zero. Since we are now interested in maximization, we do not need to exclude these solutions which are the worst possible solutions anyway.

The MaxCut problem is NP-hard, it was among the 21 problems that Karp showed to be NP-hard in 1972 [Kar72]. We discuss a randomized approximation algorithm for the MaxCut problem with expected approximation guarantee 2. The algorithm can be derandomized, i.e. one can give a deterministic 2-approximation algorithm. The deterministic variant was published by Sahni and Gonzales [SG76] in 1976. In the subsequent years, approximation algorithms were found with a guarantee of $2 - o(1)$, but no algorithm that improved the guarantee to a *constant* below 2. Thus, for a long time, the approximation factor achieved by the simple randomized algorithm was essentially the best approximation guarantee.

The constant was finally improved in 1994, in a seminal paper by Goemans and Williamson [GW94, GW95], who give a randomized 1.38-approximation by using a technique called *semidefinite programming* which goes beyond the scope of this lecture.

The randomized algorithm we consider is very simple: Start with $S = \emptyset$. For each vertex v, add v to S with probability 1/2. The running time of this algorithm is $\Theta(|V|)$. For any edge $e = \{v_i, v_j\} \in E$, $i < j$, we define $X_e = X_{ij}$ to be the indicator random variable for the event that e is in $\delta(S)$. This happens if and only if either $v_i \in S, v_j \notin S$ or $v_i \notin S, v_j \in S$. The decisions for v_i and v_j are independent, so $\mathbf{Pr}(v_i \in S \cap v_j \notin S) = (1/2) \cdot (1/2) = 1/4$, and also $\mathbf{Pr}(v_i \in S \cap v_j \notin S) = 1/4$. Both events are disjoint, so $\mathbf{Pr}(e \in \delta(S)) = (1/4) + (1/4) = 1/2$. By linearity of expectation, we obtain

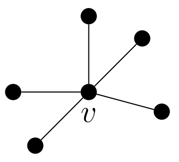
$$\mathbf{E}\left[\sum_{e \in \delta(S)} w_e\right] = \mathbf{E}\left[\sum_{e \in E} X_e w_e\right] = \sum_{e \in E} w_e \mathbf{E}[X_e] = \sum_{e \in E} w_e \cdot (1/2).$$

We can never cut more than *all* edges, so $\sum_{e \in E} w_e$ is an upper bound on the value of the best possible solution. Thus, the algorithm has an expected approximation guarantee of 2.

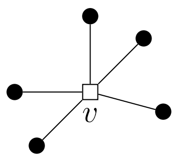
The VertexCover problem. Now we consider the *(unweighted) vertex cover problem (VertexCover).* We are given an undirected and unweighted graph $G = (V, E)$. We want to output a set $C \subseteq V$ such that every edge in the graph is *covered* by C: For every $e \in E$ with $e = \{u, v\}$, we require that $u \in C$ or $v \in C$ (or both). The goal is to minimize the the number of nodes $|C|$. The vertex cover problem is a minimization problem.

VertexCover is NP-hard, it also belonged to the problems proven to be NP-hard in [Kar72]. Different 2-approximation algorithms for the problem are known, including deterministic algorithms. It is unknown whether an α-approximation algorithm is possible for a constant $\alpha < 2$ (there are hints that this might be impossible).

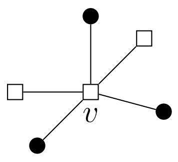
From our experience with the MaxCut problem, we might try the algorithm that adds every vertex to C with probability $1/2$. We observe two problems with this approach: First, the output might not actually be a vertex cover, some edges might not get covered. Second, the algorithm could pick a lot more vertices than an optimal solution. Consider a star graph S, i.e. a graph with vertex set $\{v\} \cup V'$ for some set V' and edge set $\{\{v, v'\} \mid v' \in V'\}$. If we decide for each vertex whether it is in C or not uniformly at random, then the expected number of vertices in C is $|V'|/2$. However, an optimal vertex cover for this graph is $C^* = \{v\}$ with $|C^*| = 1$. Thus, the expected approximation ratio of this algorithm (that also does not produce feasible solutions) would be $\Omega(|V|)$.



a star graph S



optimal solution C^*



a solution with $|C| = \frac{|V'|+1}{2}$

The following simple algorithm works. Start with $C = \emptyset$. For all *edges* $e \in E$, $e = \{u, v\}$, check if e is already covered by C. If not, add u to S with probability $1/2$, otherwise, add v. After processing all edges, it is ensured that all edges are covered. The running time of the algorithm is $\Theta(|V| + |E|)$. The algorithm is due to Pitt, who developed a version for the more general case of *weighted* vertex cover and proved that it is a 2-approximation [Pit85].

In the following, for any graph G, let $C^{ALG}(G)$ be the solution computed by the simple vertex cover approximation algorithm, and let $C^*(G)$ be an optimal vertex cover. We reconsider the above example, i.e. we let S be a star graph like defined above. We already observed that $C^*(S) = \{v\}$. We are interested in the random variable $Z := |C^{ALG}(S)|$. The algorithm starts with $C^{ALG}(S) = \emptyset$, so Z starts at zero. The edges of S are considered in an arbitrary order. For edge $e = \{v, v'\}$, two events can happen: With probability $1/2$, v' is added to $C^{ALG}(S)$, thus Z increases by one, and only the edge $\{v, v'\}$ is covered by the new vertex. With probability $1/2$, v is added. In this case, Z also increases by one, but all edges are now covered and Z freezes. We observe that Z is a geometrically distributed random variable with parameter $1/2$! Thus, we immediately conclude that $E[Z] = 2$. Since the optimum consists of one vertex, we have now proven that the expected approximation guarantee is 2 if the input is a star graph. The following theorem extends the analysis to general input graphs.

Theorem 2.21. *The randomized vertex cover algorithm is an (expected) 2-approximation algorithm.*

Proof. Let $G = (V, E)$ be an arbitrary input graph. We use the abbreviations $C^* = C^*(G)$ and $C^{ALG} = C^{ALG}(G)$. Furthermore, we set $k = C^*$ and name the vertices in the optimum vertex cover by assuming that $C^* = \{v_1^*, \ldots, v_k^*\}$.

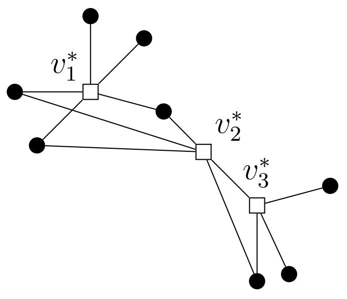
Step 1: In order to use our observations about star graphs, we want to partition G into stars. We observe the following: The set of edges covered by a vertex v_i^* is a star. These stars are not disjoint, so they do not form a true partitioning of G. However, they cover G completely. We formalize this idea. For any $v_i^* \in C^*$, let $G_i^* = (V_i^*, E_i^*)$ be defined as

$$V_i^* = \{v_i^*\} \cup \{u \in V \mid \{u, v_i^*\} \in E\}$$
$$E_i^* = \{\{u, v_i^*\} \mid \{u, v_i^*\} \in E\}$$

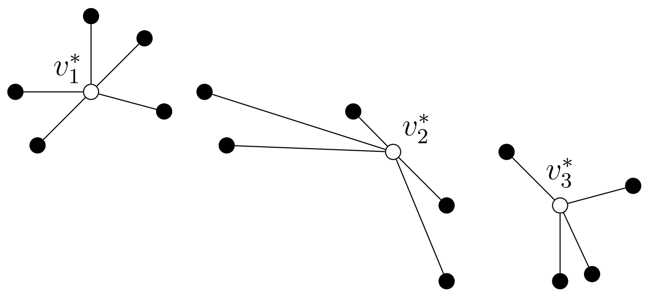
As already indicated, the intersection of V_i^* and V_j^* for $i \neq j$ is not necessarily empty, the same holds for E_i^* and E_j^* for $i \neq j$. However, we have that

$$V = \bigcup_{i=1}^{k} V_i^* \text{ and } E = \bigcup_{i=1}^{k} E_i^*$$

because C^* is a feasible vertex cover. The following is an example, observe that the subgraphs overlap.



G and an optimal C^*



subgraphs G_1^*, G_2^*, G_3^*

Step 2: Since the G_i^* cover G, we observe that

$$|C^{ALG}| \leq |C^{ALG} \cap V_1^*| + |C^{ALG} \cap V_2^*| + |C^{ALG} \cap V_3^*| \ldots + |C^{ALG} \cap V_k^*|.$$

We define the random variable $Z_i := |C^{ALG} \cap V_i^*|$ as the ith term in this sum. Observe that Z_i can only be increased in iterations where the algorithm considers edges from E_i^*. For every edge $e = \{u, v_i^*\}$ in E_i^*, the algorithm adds a vertex to C^{ALG} and thus increases Z_i by one if and only if v_i^* has not yet been added to C^{ALG}. If it adds a vertex, then this vertex is u with probability $1/2$, and it is v_i^* with probability $1/2$. We see adding v_i^* as a success and not adding it as a failure to observe that Z_i is a geometrically distributed random variable with parameter $1/2$. Thus, $E(Z_i) = 2$.

Step 3: By linearity of expectation, we obtain that

$$\mathbf{E}\left[|C^{ALG}|\right] \leq \mathbf{E}\left[\sum_{i=1}^{k} |C^{ALG} \cap V_i^*|\right] = \sum_{i=1}^{k} \mathbf{E}[Z_i] = 2k.$$

Since the optimal vertex cover has k nodes, we have now proven that the algorithm computes an expected 2-approximation. $\square$

Chapter 3

Concentration bounds

In this chapter, we see bounds that bound the probability that a random variable deviates from its expected value, depending on the extent of the deviation. Bounding this probability is not always possible, in fact, the expected value of a random variable might not even exist and then the question becomes irrelevant. However, if we know that the random variable has good properties, we can apply one of various concentration bounds, also called *tail bounds.* We see three types of tail bounds in this chapter. The first one only needs that the random variable is non-negative and that its expected value exists. These are rather weak preconditions, and the bound is also the weakest that we see in this chapter. Nevertheless, it is tremendously useful.

Theorem 3.1 (Markov's inequality)**.** *Let $(\Omega, \mathbf{Pr})$ be a discrete probability space, let $X : \Omega \to \mathbb{R}^{\geq 0}$ be a random variable that attains non-negative values. Assume that $\mathbf{E}[X]$ exists. Then*

$$\mathbf{Pr}(X \geq a) \leq \frac{\mathbf{E}[X]}{a}$$

holds for all $a \geq 0$. Thus, it also holds

$$\mathbf{Pr}(X \geq b \cdot \mathbf{E}[X]) \leq \frac{1}{b}$$

for all $b > 1$.

Proof. Let Y be the indicator random variable with

$$Y = \begin{cases} 1 & \text{if } X \geq a \\ 0 & \text{else.} \end{cases}$$

Then, $\mathbf{E}[Y] = \mathbf{Pr}(Y = 1) = \mathbf{Pr}(X \geq a)$ is the quantity that we want to bound. We observe that $Y \leq \frac{X}{a}$ is always true. We know that $X/a \geq 0$ since $X \geq 0$ and $a > 0$. Thus, if $X < a$, implying that $Y = 0$, then $X/A \geq 0 = Y$. Furthermore, if $X \geq a$ and thus $Y = 1$, then $X/a \geq 1 = Y$, so $X/a \geq Y$ holds as well. This extends to the expected value, i.e. $\mathbf{E}[Y] \leq \mathbf{E}[X/a]$ (recall Observation 2.6). Now we can conclude by linearity that

$$\mathbf{Pr}(X \geq a) = \mathbf{E}[Y] \leq \mathbf{E}\left[\frac{X}{a}\right] = \frac{\mathbf{E}[X]}{a}.$$

The second inequality follows by setting $a = b \cdot \mathbf{E}[X]$. Notice that we could do this for values $b \in (0, 1]$ as well, but then the bound is not meaningful, so we restrict b to values greater than one. $\square$