1 Randomized Approximation Algorithms

Randomized techniques give rise to some of the simplest and most elegant approximation algorithms. This section gives several examples.

1.1 A Randomized 2-Approximation for Max-Cut

In the max-cut problem, one is given an undirected graph $G = (V, E)$ and a positive weight w_e for each edge, and one must output a partition of V into two subsets A, B so as to maximize the combined weight of the edges having one endpoint in A and the other in B.

We will analyze the following extremely simple randomized algorithm: assign each vertex at random to A to B with equal probability, such that the random decisions for the different vertices are mutually independent. Let $E(A, B)$ denote the (random) set of edges with one endpoint in A and the other endpoint in B. The expected weight of our cut is

$$\mathbb{E}\left(\sum_{e \in E(A,B)} w_e\right) = \sum_{e \in E} w_e \cdot \Pr(e \in E(A, B)) = \frac{1}{2} \sum_{e \in E} w_e.$$

Since the combined weight of all edges in the graph is an obvious upper bound on the weight of any cut, this shows that the expected weight of the cut produced by our algorithm is at least half the weight of the maximum cut.

1.1.1 Derandomization using pairwise independent hashing

In analyzing the expected weight of the cut defined by our randomized algorithm, we never really used the full power of our assumption that the random decisions for the different vertices are mutually independent. The only property we needed was that for each pair of vertices u, v, the probability that u and v make different decisions is exactly $\frac{1}{2}$. It turns out that one can achieve this property using only $k = \lceil \log_2(n) \rceil$ independent random coin tosses, rather than n independent random coin tosses.

Let $\mathbb{F}_2$ denote the field $\{0, 1\}$ under the operations of addition and multiplication modulo 2. Assign to each vertex v a distinct vector $\mathbf{x}(v)$ in the vector space $\mathbb{F}_2^k$; our choice of $k = \lceil \log_2(n) \rceil$ ensures that the vector space contains enough elements to assign a distinct one to each vertex. Now let r be a uniformly random vector in $\mathbb{F}_2^k$, and partition the vertex set V into the subsets

$$A_{\mathbf{r}} = \{v \mid \mathbf{r} \cdot \mathbf{x}(v) = 0\}$$
$$B_{\mathbf{r}} = \{v \mid \mathbf{r} \cdot \mathbf{x}(v) = 1\}.$$

For any edge $e = (u, v)$, the probability that $e \in E(A_{\mathbf{r}}, B_{\mathbf{r}})$ is equal to the probability that $\mathbf{r} \cdot (\mathbf{x}(v) - \mathbf{x}(u))$ is nonzero. For any fixed nonzero vector $\mathbf{w} \in \mathbb{F}_2^k$, we have $\Pr(\mathbf{r} \cdot \mathbf{w} \neq 0) = \frac{1}{2}$ because the set of $\mathbf{r}$ satisfying $\mathbf{r} \cdot \mathbf{w} = 0$ is a linear subspace of $\mathbb{F}_2^k$ of dimension $k - 1$, hence exactly 2^{k-1} of the 2^k possible vectors r have zero dot product with $\mathbf{w}$ and the other 2^{k-1} of them have nonzero dot product with $\mathbf{w}$. Thus, if we sample $\mathbf{r} \in \mathbb{F}_2^k$ uniformly at random, the expected weight of the cut defined by $(A_\mathbf{r}, B_\mathbf{r})$ is at least half the weight of the maximum cut.

The vector space $\mathbb{F}_2^k$ has only $2^k = O(n)$ vectors in it, which suggests a deterministic alternative to our randomized algorithm. Instead of choosing $\mathbf{r}$ at random, we compute the weight of the cut $(A_\mathbf{r}, B_\mathbf{r})$ for every $r \in \mathbb{F}_2^k$ and take the one with maximum weight. This is at least as good as choosing r at random, so we get a deterministic 2-approximation algorithm at the cost of increasing the running time by a factor of $O(n)$.

1.1.2 Derandomization using conditional expectations

A different approach for converting randomization approximation algorithms into deterministic ones is the *method of conditional expectations*. In this technique, rather than making all of our random decisions simultaneously, we make them sequentially. Then, instead of making the decisions by choosing randomly between two alternatives, we evaluate both alternatives according to the conditional expectation of the objective function if we fix the decision (and all preceding ones) but make the remaining ones at random. Then we choose the alternative that optimizes this conditional expectation.

To apply this technique to the randomized max-cut algorithm, we imagine maintaining a partition of the vertex set into three sets A, B, C while the algorithm is running. Sets A, B are the two pieces of the partition we are constructing. Set C contains all the vertices that have not yet been assigned. Initially $C = V$ and $A = B = \emptyset$. When the algorithm terminates C will be empty. At an intermediate stage when we have constructed a partial partition (A, B) but C contains some unassigned vertices, we can imagine assigning each element of C randomly to A or B with equal probability, independently of the other elements of C. If we were to do this, the expected weight of the random cut produced by this procedure would be

$$w(A, B, C) = \sum_{e \in E(A,B)} w_e + \frac{1}{2} \sum_{e \in E(A,C)} w_e + \frac{1}{2} \sum_{e \in E(B,C)} w_e + \frac{1}{2} \sum_{e \in E(C,C)} w_e.$$

This suggests the following deterministic algorithm that considers vertices one by one, assigning them to either A or B using the function $w(A, B, C)$ to guide its decisions.

Algorithm 1 Derandomized max-cut algorithm using method of conditional expectations

Initialize $A = B = \emptyset$, $C = V$.
for all $v \in V$ **do**
    Compute $w(A + v, B, C - v)$ and $w(A, B + v, C - v)$.
    if $w(A + v, B, C - v) > w(A, B + v, C - v)$ **then**
        $A = A + v$
    else
        $B = B + v$
    end if
    $C = C - v$
end for
return A, B

The analysis of the algorithm is based on the simple observation that for every partition of V into three sets A, B, C and every $v \in C$, we have

$$\frac{1}{2} w(A + v, B, C - v) + \frac{1}{2} w(A, B + v, C - v) = w(A, B, C).$$

Consequently

$$\max\{w(A+v,B,C-v),\, w(A,B+v,C-v)\} \geq w(A,B,C)$$

so the value of $w(A,B,C)$ never decreases during the execution of the algorithm. Initially the value of $w(A,B,C)$ is equal to $\frac{1}{2}\sum_{e\in E} w_e$, whereas when the algorithm terminates the value of $w(A,B,C)$ is equal to $\sum_{e\in E(A,B)} w_e$. We have thus proven that the algorithm computes a partition (A,B) such that the weight of the cut is at least half the combined weight of all edges in the graph.

Before concluding our discussion of this algorithm, it's worth noting that the algorithm can be simplified by observing that

$$w(A+v,B,C-v) - w(A,B+v,C-v) = \frac{1}{2}\sum_{e\in E(B,v)} w_e - \frac{1}{2}\sum_{e\in E(A,v)} w_e.$$

The algorithm runs faster if we skip the step of actually computing $w(A+v,B,C-v)$ and jump straight to computing their difference. This also means that there's no need to explicitly keep track of the vertex set C.

Algorithm 2 Derandomized max-cut algorithm using method of conditional expectations

Initialize $A = B = \emptyset$.
for all $v \in V$ **do**
    if $\sum_{e\in E(B,v)} w_e - \sum_{e\in E(A,v)} w_e > 0$ **then**
        $A = A + v$
    else
        $B = B + v$
    end if
end for
return A, B

This version of the algorithm runs in linear time: the amount of time spent on the loop iteration that processes vertex v is proportional to the length of the adjacency list of that vertex. It's also easy to prove that the algorithm has approximation factor 2 without resorting to any discussion of random variables and their conditional expectations. One simply observes that the property

$$\sum_{e\in E(A,B)} w_e \geq \sum_{e\in E(A,A)} w_e + \sum_{e\in E(B,B)} w_e$$

is a loop invariant of the algorithm. The fact that this property holds at termination implies that $\sum_{e\in E(A,B)} w_e \geq \frac{1}{2}\sum_{e\in E} w_e$ and hence the algorithm's approximation factor is 2.

1.1.3 Epilogue: Semidefinite programming

For many years, it was not known whether any polynomial-time approximation algorithm for max-cut could achieve an approximation factor better than 2. Then in 1994, Michel Goemans and David Williamson discovered an algorithm with approximation factor roughly 1.14, based on a technique called *semidefinite programming* that is a generalization of linear program. Semidefinite programming is beyond the scope of these notes, but it has become one of the most powerful and versatile techniques in the modern theory of approximation algorithm design.

1.2 A Randomized 2-Approximation for Vertex Cover

For the unweighted vertex cover problem (the special case of weighted vertex cover in which $w_v = 1$ for all v) the following incredibly simple algorithm is a randomized 2-approximation.

Algorithm 3 Randomized approximation algorithm for unweighted vertex cover

Initialize $S = \emptyset$.
for all $e = (u, v) \in E$ **do**
  if neither u nor v belongs to S **then**
    Randomly choose u or v with equal probability.
    Add the chosen vertex into S.
  end if
end for
return S

Clearly, the algorithm runs in linear time and always outputs a vertex cover. To analyze its approximation ratio, as usual, we define an appropriate loop invariant. Let OPT denote any vertex cover of minimum cardinality. Let S_i denote the contents of the set S after completing the i^{th} iteration of the loop. We claim that for all i,

$$\mathbb{E}[|S_i \cap \text{OPT}|] \geq \mathbb{E}[|S_i \setminus \text{OPT}|]. \tag{1}$$

The proof is by induction on i. In a loop iteration in which $e = (u, v)$ is already covered by S_{i-1}, we have $S_i = S_{i-1}$ so (1) clearly holds. In a loop iteration in which $e = (u, v)$ is not yet covered, we know that at least one of u, v belongs to OPT. Thus, the left side of (1) has probability at least 1/2 of increasing by 1, while the right side of (1) has probability at most 1/2 of increasing by 1. This completes the proof of the induction step.

Consequently, letting S denote the random vertex cover generated by the algorithm, we have $\mathbb{E}[|S \cap \text{OPT}|] \geq \mathbb{E}[|S \setminus \text{OPT}|]$ from which it easily follows that $\mathbb{E}[|S|] \leq 2 \cdot |\text{OPT}|$.

The same algorithm design and analysis technique can be applied to weighted vertex cover. In that case, we choose a random endpoint of an uncovered edge (u, v) with probability inversely proportional to the weight of that endpoint.

Algorithm 4 Randomized approximation algorithm for weighted vertex cover

Initialize $S = \emptyset$.
for all $e = (u, v) \in E$ **do**
  if neither u nor v belongs to S **then**
    Randomly choose u with probability $\frac{w_v}{w_u + w_v}$ and v with probability $\frac{w_u}{w_u + w_v}$.
    Add the chosen vertex into S.
  end if
end for
return S

The loop invariant is

$$\mathbb{E}\left[\sum_{v \in S_i \cap \text{OPT}} w_v\right] \geq \mathbb{E}\left[\sum_{v \in S_i \setminus \text{OPT}} w_v\right].$$

In a loop iteration when (u, v) is uncovered, the expected increase in the left side is at least $\frac{w_u w_v}{w_u + w_v}$ whereas the expected increase in the right side is at most $\frac{w_u w_v}{w_u + w_v}$.

2 Linear Programming with Randomized Rounding

Linear programming and randomization turn out to be a very powerful when used in combination. We will illustrate this by presenting an algorithm of Raghavan and Thompson for a problem of routing paths in a network to minimize congestion. The analysis of the algorithm depends on the *Chernoff bound*, a fact from probability theory that is one of the most useful tools for analyzing randomized algorithms.

2.1 The Chernoff bound

The Chernoff bound is a very useful theorem concerning the sum of a large number of independent random variables. Roughly speaking, it asserts that for any fixed $\beta > 1$, the probability of the sum exceeding its expected value by a factor greater than β tends to zero exponentially fast as the expected sum tends to infinity.

Theorem 1. *Let $X_1, \ldots, X_n$ be independent random variables taking values in $[0,1]$, let X denote their sum, and let $\mu = \mathbb{E}[X]$. For every $\beta > 1$,*

$$\Pr(X \geq \beta\mu) < e^{-\mu[\beta \ln(\beta) - (\beta - 1)]}. \tag{2}$$

Proof. The key idea in the proof is to make use of the moment-generating function of X, defined to be the following function of a real-valued parameter t:

$$M_X(t) = \mathbb{E}\left[e^{tX}\right].$$

From the independence of $X_1, \ldots, X_n$, we derive:

$$M_X(t) = \mathbb{E}\left[e^{tX_1} e^{tX_2} \cdots e^{tX_n}\right] = \prod_{i=1}^{n} \mathbb{E}\left[e^{tX_i}\right]. \tag{3}$$

To bound each term of the product, we reason as follows. Let Y_i be a $\{0,1\}$-valued random variable whose distribution, conditional on the value of X_i, satisfies $\Pr(Y_i = 1 \mid X_i) = X_i$. Then for each $x \in [0,1]$ we have

$$\mathbb{E}\left[e^{tY_i} \middle| X_i = x\right] = xe^t + (1-x)e^0 \geq e^{tx} = \mathbb{E}\left[e^{tX_i} \middle| X_i = x\right],$$

where the inequality in the middle of the line uses the fact that e^{tx} is a convex function. Since this inequality holds for every value of x, we can integrate over x to remove the conditioning, obtaining

$$\mathbb{E}\left[e^{tY_i}\right] \geq \mathbb{E}\left[e^{tX_i}\right].$$

Letting μ_i denote $\mathbb{E}[X_i] = \Pr(Y_i = 1)$ we find that

$$\left[e^{tX_i}\right] \leq \left[e^{tY_i}\right] = \mu_i e^t + (1 - \mu_i) = 1 + \mu_i(e^t - 1) \leq \exp\left(\mu_i(e^t - 1)\right),$$

where $\exp(x)$ denotes e^x, and the last inequality holds because $1 + x \leq \exp(x)$ for all x. Now substituting this upper bound back into (3) we find that

$$\mathbb{E}\left[e^{tX}\right] \leq \prod_{i=1}^{n} \exp\left(\mu_i(e^t - 1)\right) = \exp\left(\mu(e^t - 1)\right).$$

On the other hand, since e^{tX} is positive for all t, X, we have $\mathbb{E}\left[e^{tX}\right] \geq e^{t\beta\mu}\Pr(X \geq \beta\mu)$, hence

$$\Pr(X \geq \beta\mu) \leq \exp\left(\mu(e^t - 1 - \beta t)\right).$$

We are free to choose $t > 0$ so as to minimize the right side of this inequality. The minimum is attained when $t = \ln\beta$, which yields the inequality specified in the statement of the theorem. □

Corollary 2. *Suppose $X_1, \ldots, X_k$ are independent random variables taking values in $[0, 1]$, such that $\mathbb{E}[X_1 + \cdots + X_k] \leq 1$. Then for any $N > 2$ and any $b \geq \frac{3\log N}{\log\log N}$, where* log *denotes the base-2 logarithm, we have*

$$\Pr(X_1 + \cdots + X_k \geq b) < \frac{1}{N}. \tag{4}$$

Proof. Let $\mu = \mathbb{E}[X_1 + \cdots + X_k]$ and $\beta = b/\mu$. Applying Theorem 1 we find that

$$\begin{aligned}\Pr(X_1 + \cdots + X_k \geq b) &\leq \exp\left(-\mu\beta\ln(\beta) + \mu\beta - \mu\right)\\ &= \exp\left(-b(\ln(\beta) - 1) - \mu\right) \leq e^{-b(\ln(\beta/e))}.\end{aligned} \tag{5}$$

Now, $\beta = b/\mu \geq b$, so

$$\frac{\beta}{e} \geq \frac{b}{e} \geq \frac{3\log N}{e\log\log N}$$

and

$$\begin{aligned}b\ln\left(\frac{\beta}{e}\right) &\geq \left(\frac{3\ln N}{\ln(\log N)}\right)\cdot\ln\left(\frac{3\log N}{e\log\log N}\right)\\ &= 3\ln(N)\cdot\left(1 - \frac{\ln(\log\log N) - \ln(3) + 1}{\ln(\log N)}\right) > \ln(N),\end{aligned} \tag{6}$$

where the last inequality holds because one can verify that $\ln(\log x) - \ln(3) + 1 < \frac{2}{3}\ln(x)$ for all $x > 1$ using basic calculus. Now, exponentiating both sides of (6) and combining with (5) we obtain the bound $\Pr(X_1 + \cdots + X_k \geq b) < 1/N$, as claimed. □

2.2 An approximation algorithm for congestion minimization

We will design an approximation algorithm for the following optimization problem. The input consists of a directed graph $G = (V, E)$ with positive integer edge capacities c_e, and a set of source-sink pairs (s_i, t_i), $i = 1, \ldots, k$, where each (s_i, t_i) is a pair of vertices such that G contains at least one path from s_i to t_i. The algorithm must output a list of paths $P_1, \ldots, P_k$ such that P_i is a path from s_i to t_i. The load on edge e, denoted by ℓ_e, is defined to be the number of paths P_i that traverse edge e. The congestion of edge e is the ratio ℓ_e/c_e, and the algorithm's objective is to minimize congestion, i.e. minimize the value of $\max_{e\in E}(\ell_e/c_e)$. This problem turns out to be NP-hard, although we will not prove that fact here.

The first step in designing our approximation algorithm is to come up with a linear programming relaxation. To do so, we define a decision variable $x_{i,e}$ for each $i = 1, \ldots, k$ and each $e \in E$, denoting whether or not e belongs to P_i, and we allow this variable to take fractional values. The resulting linear program can be written as follows, using $\delta^+(v)$ to denote the set of edges leaving v and $\delta^-(v)$ to denote the set of edges entering v.

$$
\begin{array}{lll}
\min & r & \\
\text{s.t.} & \sum_{e\in\delta^+(v)} x_{i,e} - \sum_{e\in\delta^-(v)} x_{i,e} = \begin{cases} 1 & \text{if } v = s_i \\ -1 & \text{if } v = t_i \\ 0 & \text{if } v \neq s_i, t_i \end{cases} & \forall i = 1,\ldots,k,\ v \in V \\
& \sum_{i=1}^{k} x_{i,e} \leq c_e \cdot r & \forall e \in E \\
& x_{i,e} \geq 0 & \forall i = 1,\ldots,k,\ e \in E
\end{array}
\tag{7}
$$

When $(x_{i,e})$ is a $\{0,1\}$-valued vector obtained from a collection of paths $P_1,\ldots,P_k$ by setting $x_{i,e} = 1$ for all $e \in P_i$, the first constraint ensures that P_i is a path from s_i to t_i while the second one ensures that the congestion of each edge is bounded above by r.

Our approximation algorithm solves the linear program (7), does some postprocessing of the solution to obtain a probability distribution over paths for each terminal pair (s_i, t_i), and then outputs an independent random sample from each of these distributions. To describe the postprocessing step, it helps to observe that the first LP constraint says that for every $i \in \{1,\ldots,k\}$, the values $x_{i,e}$ define a network flow of value 1 from s_i to t_i. Define a flow to be *acyclic* if there is no directed cycle C with a positive amount of flow on each edge of C. The first step of the postprocessing is to make the flow $(x_{i,e})$ acyclic, for each i. If there is an index $i \in \{1,\ldots,k\}$ and a directed cycle C such that $x_{i,e} > 0$ for every edge $e \in C$, then we can let $\delta = \min\{x_{i,e} \mid e \in C\}$ and we can modify $x_{i,e}$ to $x_{i,e} - \delta$ for every $e \in C$. This modified solution still satisfies all of the LP constraints, and has strictly fewer variables $x_{i,e}$ taking nonzero values. After finitely many such modifications, we must arrive at a solution in which each of the flow $(x_{i,e})$, $1 \leq i \leq k$ is acyclic. Since this modified solution is also an optimal solution of the linear program, we may assume without loss of generality that in our original solution $(x_{i,e})$ the flow was acyclic for each i.

Next, for each $i \in \{1,\ldots,k\}$ we take the acyclic flow $(x_{i,e})$ and represent it as a probability distribution over paths from s_i to t_i, i.e. a set of ordered pairs (P, π_P) such that P is a path from s_i to t_i, π_P is a positive number interpreted as the probability of sampling P, and the sum of the probabilities π_P over all paths P is equal to 1. The distribution can be constructed using the following algorithm.

Algorithm 5 Postprocessing algorithm to construct path distribution

Given: Source s_i, sink t_i, acyclic flow $x_{i,e}$ of value 1 from s_i to t_i.
Initialize $\mathcal{D}_i = \emptyset$.
while there is a path P from s_i to t_i such that $x_{i,e} > 0$ for all $e \in P$ **do**
    $\pi_P = \min\{x_{i,e} \mid e \in P\}$
    $\mathcal{D}_i = \mathcal{D}_i \cup \{(P, \pi_P)\}$.
    for all $e \in P$ **do**
        $x_{i,e} = x_{i,e} - \pi_P$
    end for
end while
return $\mathcal{D}_i$

Each iteration of the **while** loop strictly reduces the number of edges with $x_{i,e} > 0$, hence the algorithm must terminate after selecting at most m paths. When it terminates, the flow $(x_{i,e})$

has value zero (as otherwise there would be a path from s_i to t_i with positive flow on each edge) and it is acyclic because $(x_{i,e})$ was initially acyclic and we never put a nonzero amount of flow on an edge whose flow was initially zero. The only acyclic flow of value zero is the zero flow, so when the algorithm terminates we must have $x_{i,e} = 0$ for all e.

Each time we selected a path P, we decreased the value of the flow by exactly π_P. The value was initially 1 and finally 0, so the sum of π_P over all paths P is exactly 1 as required. For any given edge e, the value $x_{i,e}$ decreased by exactly π_P each time we selected a path P containing e, hence the combined probability of all paths containing e is exactly $x_{i,e}$.

Performing the postprocessing algorithm 5 for each i, we obtain probability distributions $\mathcal{D}_1, \dots, \mathcal{D}_k$ over paths from s_i to t_i, with the property that the probability of a random sample from $\mathcal{D}_i$ traversing edge e is equal to $x_{i,e}$. Now we draw one independent random sample from each of these k distributions and output the resulting k-tuple of paths, $P_1, \dots, P_k$. We claim that with probability at least 1/2, the parameter $\max_{e \in E} \{\ell_e / c_e\}$ is at most αr, where $\alpha = \frac{3\log(2m)}{\log\log(2m)}$. This follows by a direct application of Corollary 2 of the Chernoff bound. For any given edge e, we can define independent random variables $X_1, \dots, X_k$ by specifying that

$$X_i = \begin{cases} (c_e \cdot r)^{-1} & \text{if } e \in P_i \\ 0 & \text{otherwise.} \end{cases}$$

These are independent and the expectation of their sum is $\sum_{i=1}^{k} x_{i,e}/(c_e \cdot r)$, which is at most 1 because of the second LP constraint above. Applying Corollary 2 with $N = 2m$, we find that the probability of $X_1 + \cdots + X_k$ exceeding α is at most $1/(2m)$. Since $X_1 + \cdots X_k = \ell_e (c_e \cdot r)^{-1}$, this means that the probability of ℓ_e / c_e exceeding αr is at most $1/(2m)$. Summing the probabilities of these failure events for each of the m edges of the graph, we find that with probability at least 1/2, none of the failure events occur and $\max_{e \in E} \{\ell_e / c_e\}$ is bounded above by αr. Now, r is a lower bound on the parameter $\max_{e \in E} \{\ell_e / c_e\}$ for *any* k-tuple of paths with the specified source-sink pairs, since any such k-tuple defines a valid LP solution and r is the optimum value of the LP. Consequently, our randomized algorithm achieves approximation factor α with probability at least 1/2.