15 Approximation Algorithms

An optimization problem can be represented by a function $o : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$, where $o(x, y)$ is the objective value of output $y \in \{0,1\}^*$ for input $x \in \{0,1\}^*$. Restricting our attention to minimization problems, a function $f : \{0,1\}^* \to \{0,1\}^*$ provides an optimal solution to optimization problem o if for every input $x \in \{0,1\}^*$, $o(x, f(x)) = \min\{o(x, y) : y \in \{0,1\}^*\}$. When f is hard to compute, one might instead try to find a solution that is as good as possible. The most common notion of quality in this context is a worst-case multiplicative one: a function $g : \{0,1\}^* \to \{0,1\}^*$ will be called an α-approximation for o, for some $\alpha \geqslant 1$, if for all $x \in P$, $o(x, g(x)) \leqslant \alpha o(x, f(x))$. In what follows we will be interested in algorithms that compute function g in polynomial time, and will refer to such an algorithm as a (polynomial-time) α-approximation algorithm for optimization problem o.

While in principle α could depend on the size of the input, we will only consider problems in the complexity class APX, which have an α-approximation algorithm for some constant α. The class PTAS $\subseteq$ APX, for polynomial-time approximation scheme, contains problems that possess an $(1+\epsilon)$-approximation algorithm for any $\epsilon > 0$, where the (polynomial) running time can depend on ϵ in an arbitrary way. APX-hardness is established by a reduction corresponding to the smaller class PTAS, and implies that a problem can be approximated in polynomial time up to some, but not every, constant factor.

15.1 The Max-Cut Problem

Given an undirected graph $G = (V, E)$, the max-cut problem asks for a cut of G that maximizes the number of edges crossing from one side to the other, i.e., a set $S \subseteq V$ such that $|E \cap (S \times (V \setminus S))|$ is as large as possible. The max-cut problem is NP-complete and thus cannot be solved exactly in polynomial time unless $P = NP$.

On the other hand, a simple greedy algorithm provides a $1/2$-approximation. First of all, observe that every graph has a cut of size at least $|E|/2$. For this, consider a random cut $S \subseteq V$ such that for each $v \in V$, $v \in S$ independently with probability $1/2$. Then, the number of edges across the cut is a random variable $Q = \sum_{\{i,j\} \in E} \mathbb{I}[Q_i \neq Q_j]$, where $\mathbb{I}$ denotes the indicator function on binary events and for each $i \in V$, Q_i is a Bernoulli random variable with parameter $1/2$. Thus,

$$\mathbb{E}[Q] = \mathbb{E}\Big[\sum_{\{i,j\} \in E} \mathbb{I}[Q_i \neq Q_j]\Big] = \sum_{\{i,j\} \in E} \mathbb{E}\big[\mathbb{I}[Q_i \neq Q_j]\big] = \sum_{\{i,j\} \in E} \mathbb{P}[Q_i \neq Q_j]\big] = |E|/2,$$

where the second equality holds by linearity of expectation, so there must exist a cut of size at least $|E|/2$. This probabilistic argument can be de-randomized efficiently using the *method of conditional probabilities*. To this end, each vertex is considered in turn, replacing the random decision whether the vertex is included in S by a deterministic one. The goal is to ensure that the conditional probability of obtaining a cut of size at least $|E|/2$, assuming that the remaining decisions are taken randomly, remains positive. While the conditional probability itself may not be easy to determine, it suffices in each step to maximize the conditional expectation of the random variable Q, because the maximum is guaranteed to be at least $|E|/2$. Let $U \subseteq V$ be the set of vertices for which a decision has already been made, and let $S \subseteq U$ be the resulting cut. The conditional expectation of Q given this choice is

$$\Big|E \cap \big(S \times (U \setminus S)\big)\Big| + \frac{1}{2}\Big|E \cap \big(V \times (V \setminus U)\big)\Big|,$$

and it can be maximized by maximizing the first term. Since the size of a cut can be at most $|E|$, a greedy algorithm that considers each vertex in turn and adds it to either S or $U \setminus S$ in order to maximize the first term provides a $1/2$-approximation.

A better approximation can be obtained using semidefinite programming.

Theorem 15.1 (Goemans and Williamson, 1995). *There exists a 0.87856-approximation algorithm for the max-cut problem.*

Proof sketch. The max-cut problem can be written as the following integer quadratic program:

$$\begin{aligned} \text{maximize} \quad & \sum_{\{i,j\}\in E} \frac{1 - x_i x_j}{2} \\ \text{subject to} \quad & x_i \in \{-1, 1\} \quad \text{for all } i \in V. \end{aligned} \tag{15.1}$$

The intuition behind the objective is that $x_i = 1$ if $i \in S$, and edge $\{i, j\} \in E$ contributes 1 to the sum if and only if $|\{i, j\} \cap S| = 1$.

Since the max-cut problem is NP-complete, an optimal solution of (15.1) cannot be found in polynomial time unless P = NP. Note, however, that

$$\sum_{\{i,j\}\in E} \frac{1 - x_i x_j}{2} = \frac{|E|}{2} - \frac{1}{4} x^T C x = \frac{|E|}{2} - \frac{1}{4}\langle C, xx^T\rangle,$$

where $C \in \{0, 1\}^{|V|\times|V|}$ with $C_{ij} = 1$ if $\{i, j\} \in E$ and $C_{ij} = 0$ otherwise. Moreover, xx^T is a positive semidefinite matrix. We can thus relax the constraints, and obtain an upper bound on the optimal solution of (15.1), by replacing xx^T by a general positive semidefinite matrix X with $X_{ii} = 1$ for all $i \in V$. We arrive at the following optimization problem, which is an SDP:

$$\begin{aligned} \text{maximize} \quad & \frac{|E|}{2} - \frac{1}{4}\langle C, X\rangle \\ \text{subject to} \quad & X_{ii} = 1 \quad \text{for all } i \in V \\ & X \succeq 0. \end{aligned}$$

Since the constraints have been relaxed, an optimal solution of this SDP need not be feasible for (15.1). Intuitively, a feasible solution of (15.1) corresponds to a set of $|V|$

unit vectors in $\mathbb{R}^1$, while a feasible solution of the relaxed problem corresponds to a set of $|V|$ unit vectors in $\mathbb{R}^n$. The latter can be "rounded" to the former, however, by randomly picking a hyperplane in $\mathbb{R}^n$ that passes through the origin and mapping each unit vector in $\mathbb{R}^n$ to -1 or 1 depending on its relative position to this hyperplane. Surprisingly, this changes the objective value by a factor of at most 0.87856, and thus yields a feasible solution of (15.1) that is within the same factor of an optimal one. All the necessary steps can be carried out in polynomial time, and the method can also be de-randomized without affecting the approximation factor. □

15.2 Hardness of Approximation

An obvious question is whether the bound of Theorem 15.1 is optimal, or whether it can be improved further. NP-hardness of a problem establishes that it is hard to distinguish instances with a certain optimum, like the size of a maximum cut in the case of the max-cut problem, from instances whose optimum is smaller or larger. A possible approach for showing that a problem does not admit an α-approximation algorithm for some $\alpha < 1$ would be to create a gap between positive and negative instances, and show that it is hard to distinguish instances with a large optimum from instances with a small optimum. The problem with this approach is that our characterization of the class NP is very fragile. Cook's proof of Theorem 5.1 uses a class of Boolean formulae to encode the computations of a Turing machine, which in turn are very sensitive to small changes. And indeed, if one were to inspect the formulae more closely, one would see that it is very easy for each of them to find an assignment that satisfies every clause except one. What is needed to show hardness of approximation is a more robust model of NP. Such a model is provided by *probabilistically checkable proofs* (PCPs).

PCPs can be obtained using a probabilistic modification of the definition of NP. As before, we are given access to an input x and to a certificate y which acts as proof that x satisfies a certain property. Instead of a deterministic Turing machine as in the case of NP, however, we want to use a *probabilistic verifier* V to check the proof. The fact that V is probabilistic can be modeled by assuming that besides x and y it takes an additional input r, which is a string of random bits, and then performs a deterministic computation based on x, y, and r. For fixed x and y, we say that V accepts x and y with probability p if it accepts with this probability for a uniformly distributed random string r. Note that so far we have only made the verifier more powerful, by giving it access to a random string. To be able to say something interesting about its relationship to the class NP, we therefore have to restrict what it can do with its inputs. We call a verifier V $(r(n), q(n))$-restricted, for two functions $r : \mathbb{Z} \to \mathbb{Z}$ and $q : \mathbb{Z} \to \mathbb{Z}$, if for every input of length n and every certificate y, it queries at most $q(n)$ bits of y and uses at most $r(n)$ random bits. A problem L then is in the class $\text{PCP}[r(n), q(n)]$ if there exists an $(r(n), q(n))$-restricted verifier with the following properties: if $x \in L$, then there exists a certificate y such that V accepts x and y with probability 1; if $x \notin L$, then for every certificate y, V accepts x and y with probability at most $1/2$.

It is easy to see, for example, that $\mathrm{PCP}[O(\log n), O(\log n)] \subseteq \mathrm{NP}$: a verifier that uses a logarithmic number of random bits can easily be de-randomized by considering all possible random strings of logarithmic length. The PCP theorem states a surprising converse: every problem in NP has a probabilistic verifier that uses a logarithmic number of random bits and examines a *constant* number of bits of the certificate.

THEOREM 15.2 (Arora et al., 1998). $\mathrm{NP} = \mathrm{PCP}[O(\log n), O(1)]$

The PCP theorem can be used to show that max-3SAT, the problem of computing the maximum number of simultaneously satisfiable clauses of an instance of SAT with three literals per clause, is APX-hard.

THEOREM 15.3 (Arora et al., 1998). *There exists* $\epsilon > 0$ *such that there is no* $(1-\epsilon)$-*approximation algorithm for max-3SAT unless* $\mathrm{P} = \mathrm{NP}$.

Proof sketch. By the PCP Theorem, any NP-complete problem has a $(c \log n, q)$-restricted verifier V for some constants c and q. For a particular random string r, V chooses q positions $i_1^r, \ldots, i_q^r$ of the certificate y and a function $f_x^r : \{0,1\}^q \to \{0,1\}$, and accepts if and only if $f_x^r(y_{i_1^r}, \ldots, y_{i_q^r}) = 1$.

The proof works by constructing, for each $x \in L$, a Boolean formula ϕ_x with variables $\nu_1, \ldots, \nu_{|y|}$ and clauses representing the constraint $f_x^r(\nu_{i_1^r}, \ldots, \nu_{i_q^r}) = 1$ for every possible random string r. Formula ϕ_x has a number m of clauses that is polynomial in $|x|$, and the following can be shown to hold for $\epsilon = \frac{1}{2}\frac{1}{q2^q}$:

if $x \in L$, then ϕ_x is satisfiable;

if $x \notin L$, then no assignment satisfies more than $(1-\epsilon)m$ clauses of ϕ_x.

By distinguishing between the case where the number of satisfiable clauses is m and the case where the number of satisfiable clauses is $(1-\epsilon)m$, we can thus distinguish between the cases $x \in L$ and $x \notin L$, thereby solving an NP-complete problem. The existence of a polynomial-time algorithm for the former problem would thus imply that $\mathrm{P} = \mathrm{NP}$. □

Since there is a PTAS reduction from max-3SAT to max-cut, the latter is APX-hard as well. Improved PCP characterizations of NP have lead to better bounds for various problems, which in some cases are tight: max-3SAT, for example, is NP-hard to approximate to a factor of $7/8+\epsilon$ for any $\epsilon > 0$, and a factor of $7/8$ can be achieved easily by choosing a value for each variable uniformly at random. For max-cut the same techniques yield an upper bound of $16/17+\epsilon$, which does not match the lower bound of Theorem 15.1. The apparent difficulty in improving the upper bound seems to be related to the fact that constraints in the max-cut problem involve two variables, compared to three in the case of max-3SAT, and that the known PCP characterizations corresponding to the two-variable case are weaker. The *unique games conjecture* postulates the existence of a PCP construction for the two-variable case that would imply an upper bound for max-cut that matches the bound of Theorem 15.1.