Beyond the Worst-Case Analysis of Algorithms

Edited by
Tim Roughgarden

Contents

1
Perturbation Resilience

Konstantin Makarychev and Yury Makarychev

Abstract

This chapter introduces *perturbation resilience* (also known as Bilu-Linial stability). Loosely speaking, an instance is perturbation-resilient if the optimal solution remains the same when we perturb the instance. We present algorithmic and hardness results for perturbation-resilient instances. In particular, we describe *certified algorithms* that attempt to bridge the gap between the worst-case and structured instances: on one hand, they *always* find an approximate solution; on the other hand, they exactly solve perturbation-resilient instances.

1.1 Introduction

In this chapter, we discuss the notion of *perturbation resilience* (also known as stability), which was introduced by Bilu and Linial (2010). The notion of perturbation resilience aims to capture real-life instances of combinatorial optimization and clustering problems. Informally, an instance of a combinatorial optimization or clustering problem is perturbation-resilient if the optimal solution remains the same when we perturb the instance.

Definition 1.1 Consider a combinatorial optimization or clustering problem. Suppose that every instance has a number of parameters; for example, if the problem is a graph partitioning problem, the parameters are edge weights; if it is a constraint satisfaction problem, they are constraint weights; if it is a clustering problem, they are distances between points. A γ*-perturbation* of an instance $\mathcal{I}$ is an instance $\mathcal{I}'$ produced by multiplying each parameter in $\mathcal{I}$ by a number between 1 and γ (the number may be different for each parameter).[1]

Definition 1.2 An instance $\mathcal{I}$ is γ-perturbation resilient if every γ-perturbation

[1] All problems we consider are scale invariant, so equivalently we can divide the parameters by a number between 1 and γ; we will use this convention when we talk about clustering problems.

of $\mathcal{I}$ has the same optimal solution as $\mathcal{I}$ (we require that $\mathcal{I}$ have a unique optimal solution).

While the solution should not change, the *value* or *cost* of the solution may and, generally speaking, will change when we perturb the instance. The larger γ is, the more restrictive the γ-perturbation resilience condition becomes. In particular, the instance is 1-stable if and only if it has a unique solution.

In this chapter, we also describe *certified algorithms* (see Definition 1.4 below), which attempt to bridge the gap between the worst-case and structured instances: on one hand, they *always* find an approximate solution; on the other, they exactly solve perturbation-resilient instances.

Motivation. The definition of perturbation resilience is particularly applicable to machine learning problems, where we are interested in finding the *true* solution/clustering/partitioning rather than in optimizing the objective function per se. Indeed, when we frame a real-life problem as a machine learning problem, we make a number of somewhat arbitrary modelling decisions (for example, when we solve a clustering problem, we choose one similarity function among a number of reasonable choices). If the optimal solution is very sensitive to these modelling choices, then, by solving the problem exactly, we will likely not find the true solution. This suggests that there is no point in solving non-perturbation-resilient instances of many machine learning problems in the first place. Additionally, empirical evidence shows that in many real-life instances, the optimal solution stands out among all feasible solutions and is thus not sensitive to small perturbations of the parameters.

Weak perturbation resilience. The definition of perturbation resilience is somewhat strict. Perhaps, it is more natural to require that the optimal solution to a perturbed instance be *close* but not necessarily equal to the optimal solution for the original instance. This notion is captured in the definitions of (γ, N)-weak perturbation resilience.[2]

Definition 1.3 (Makarychev et al. (2014)) Consider an instance $\mathcal{I}$ of a combinatorial optimization problem. Let s^* be an optimal solution and N be a set of solutions, which includes all optimal solutions. Then s^* is better than any solution s outside of N. Assume further that for every γ-perturbation $\mathcal{I}'$ of $\mathcal{I}$, s^* is a better solution for $\mathcal{I}'$ than s is. Then we say that $\mathcal{I}$ is (γ, N)-weakly perturbation-resilient or simply γ-weakly perturbation-resilient. We say that an algorithm solves a weakly perturbation-resilient instance $\mathcal{I}$, if given a (γ, N)-weakly perturbation-resilient instance, it finds a solution $s \in N$ (crucially, the algorithm does not know N).

One should think of set N in Definition 1.3 as the set of solutions that are close to s^* in some sense. Let us say we solve Max Cut. Then, N may be the set of

[2] We note that a related notion of weak perturbation-resilience, called (γ, ε)-perturbation resilience, was introduced by (Balcan and Liang, 2016).

cuts (S', T') that can be obtained from the optimal cut (S^*, T^*) by moving at most an ε fraction of the vertices from one side of the cut to the other. Or, N may be the set of cuts that partition some subset of the vertices V_0 (informally, the "core of the graph" or the subset of "important vertices") in the same way as (S^*, T^*). Or, it may be a set of cuts that satisfy some other structural property. Note that an instance is $(\gamma, \{s^*\})$-weakly perturbation-resilient if and only if it is γ-perturbation-resilient.

It would be interesting to further relax the definition of perturbation resilience. In particular, it would be more natural to require only that the optimal solution not change if we *randomly* perturb the input. Unfortunately, we do not have any results for this weaker definition of perturbation resilience.

Certified Algorithms. Let us now define the notion of a certified approximation algorithm (Makarychev and Makarychev, 2019). The definition is inspired by the definitions of perturbation resilience and smoothed analysis (Spielman and Teng, 2004) (see also Chapters 13–15 of this book). Recall that in the smoothed analysis framework, we analyze the performance of an algorithm on a small random perturbation of the input instance. That is, we show that, after we randomly perturb the input, the algorithm can solve it with the desired accuracy in the desired time. A certified algorithm perturbs the input instance *on its own* and then solves the obtained instance exactly. Importantly, the perturbation does not have to be random or small (in fact, we will later see that for many problems the perturbation must be considerable).

Definition 1.4 A γ-certified algorithm is an algorithm that, given an instance $\mathcal{I}$ of the problem, returns a γ-perturbation $\mathcal{I}'$ of $\mathcal{I}$ and an optimal solution s^* for $\mathcal{I}'$. We will say that $\mathcal{I}'$ certifies s^*.

As we will see in Section 1.2, certified algorithms have a number of desirable properties. A γ-certified algorithm always gives a γ-approximation for the problem and its "complement", exactly solves γ-perturbation-resilient instances, and solves weakly perturbation-resilient instances. Also, one may run a certified algorithm, get a perturbed instance $\mathcal{I}'$ and an optimal solution s^* for it, then, taking into account problem-specific considerations, decide for oneself whether $\mathcal{I}'$ is similar enough to $\mathcal{I}$ and, consequently, whether s^* is a reasonably good solution for $\mathcal{I}$.

Robust Algorithms. Most algorithms for perturbation-resilient instances of combinatorial optimization problems (but not clustering problems) that we discuss in this chapter are *robust* – they never output an incorrect answer, even if the input is not γ-perturbation-resilient.

Definition 1.5 An algorithm for γ-perturbation-resilient instances is robust if the following holds: if the input instance is γ-perturbation-resilient, the algorithm finds the optimal solution; if the instance is not γ-perturbation-resilient, the algorithm either finds an optimal solution or reports that the instance is not γ-perturbation-resilient.

This property is very desirable, as in real life we can only assume that input instances are perturbation-resilient but we cannot be completely certain that they indeed are.

Running time. The running time of most certified algorithm we consider in this chapter will be polynomial in the size of the input and the magnitude of the parameters. Thus, we will refer to these algorithms as pseudo-polynomial-time algorithms. Specifically, the running time will be polynomial in the size of *the input* and *the ratio between the largest and the smallest parameters.* To simplify the exposition, we will additionally assume that the parameters are integers between 1 and W. However, this assumption is not crucial (see (Makarychev and Makarychev, 2019)). In this chapter, we will also talk about other ("non-certified") algorithms for perturbation and weakly-perturbation resilient instances – these algorithms will be *true* polynomial-time algorithms, whose running time is polynomial in the input size.

Organization. We discuss results for combinatorial optimization problems in Sections 1.2–1.3 and results for clustering problems in Sections 1.5-1.7.

1.2 Combinatorial Optimization Problems

In this section, we describe properties of certified algorithm for combinatorial optimization problems.

Preliminaries. We will formally define what a combinatorial optimization problem is. Our definition will capture various constraint satisfaction, graph partitioning, and covering problems. It will be instructive for us to keep in mind two examples of such problems, Max Cut and Min Uncut.

Definition 1.6 In *Max Cut*, given a graph $G = (V, E, w_e)$, the goal is to find a cut $(S, \bar{S})$ in G that maximizes the total weight of the cut edges. In *Min UnCut*, given a graph $G = (V, E, w_e)$, the goal is to find a cut $(S, \bar{S})$ in G that minimizes the total weight of the edges not cut by $(S, \bar{S})$.

For a given graph G, the value of a cut $(S, \bar{S})$ w.r.t. the Max Cut objective plus the cost of $(S, \bar{S})$ w.r.t. the Min Uncut objective equals the total weight of all the edges and does not depend on the specific cut $(S, \bar{S})$. In particular, the optimal solution for Max Cut is also an optimal solution for Min Uncut and vice versa. However, as we will discuss later a good *approximate solution* for one of the problems is not necessarily a good solution for the other. We say that Max Cut and Min Uncut are *complementary* problems. Now we give a general definition of a combinatorial optimization problem.

Definition 1.7 An instance of *a combinatorial optimization problem* is specified by a set of feasible solutions $\mathcal{S}$ (the solution space), a set of constraints $\mathcal{C}$, and constraint weights $w_c > 0$ for $c \in \mathcal{C}$. Typically, the solution space $\mathcal{S}$ is of exponential size and is not provided explicitly. Each constraint is a map from $\mathcal{S}$ to $\{0, 1\}$. We say that a feasible solution $s \in \mathcal{S}$ satisfies a constraint c in $\mathcal{C}$ if $c(s) = 1$.

We consider maximization and minimization objectives.

- The *maximization objective* is to maximize the total weight of the satisfied constraints: find $s \in \mathcal{S}$ that maximizes $\mathrm{val}_{\mathcal{I}}(s) = \sum_{c\in\mathcal{C}} w_c c(s)$.
- The *minimization objective* is to minimize the total weight of the unsatisfied constraints: find $s \in \mathcal{S}$ that minimizes $\sum_{c\in\mathcal{C}} w_c(1-c(s)) = w(\mathcal{C}) - \mathrm{val}_{\mathcal{I}}(s)$ (where $w(\mathcal{C}) = \sum_{c\in\mathcal{C}} w(c)$ is the total weight of all the constraints).

We say that maximization and minimization are *complementary* objectives; likewise, we call two instances that only differ in the objective *complementary* instances. Note that complementary instances have the same optimal solutions.

Weights $\{w_c\}_{c\in\mathcal{C}}$ are the parameters of the instance in the sense of Definition 1.1.

As is standard for maximization and minimization constraint satisfaction problems, we do not require that a feasible solution $s \in \mathcal{S}$ satisfy all of the constraints. In other words, we assume that the constraints are "soft"; later we will consider problems with "hard" and "soft" constraints (see Theorem 1.12).

Definition 1.8 An optimization problem is a family $\mathcal{F}$ of instances. We require that all instances in $\mathcal{F}$ have the same type of the objective (either all of them have a maximization or all have a minimization objective). We assume that if an instance $(\mathcal{S}, \mathcal{C}, w)$ is in $\mathcal{F}$, then so is $(\mathcal{S}, \mathcal{C}, w')$ for any choice of positive weights w.

Let us see why this definition captures Max Cut and Min Uncut. For a given instance $G = (V, E, w)$ of Max Cut or Min Uncut, $\mathcal{S}$ is the set of all the cuts in G. For every edge $e \in E$, there is a constraint c_e; $c_e((S, \bar{S})) = 1$ if e is cut by $(S, \bar{S})$. The objective for Max Cut is to maximize $\sum_{c\in\mathcal{C}} w_c c(S, \bar{S})$. The objective for Min Uncut is to minimize $\sum_{c\in\mathcal{C}} w_c(1 - c(S, \bar{S}))$.

Consider two other examples.

Example 1.9 In *Minimum Multiway Cut*, we are given a graph $G = (V, E, w_e)$ and a set of terminals $t_1, \dots, t_k$. The goal is to partition G into k clusters $P_1, \dots, P_k$ such that $t_i \in P_i$ for $i \in \{1, \dots, k\}$ so as to minimize the total weight of the cut edges. For this problem, $\mathcal{S}$ is the set of all partitions $P_1, \dots, P_k$ such that $t_i \in P_i$ for every i. For every edge $e \in E$, there is a constraint c_e; $c_e((P_1, \dots, P_k)) = 1$ if e is *not* cut by $(P_1, \dots, P_k)$. The objective is to minimize $\sum_{c\in\mathcal{C}} w_c(1 - c(P_1, \dots, P_k))$.

Example 1.10 In *Maximum Independent Set*, we are given a graph $G = (V, E, w_v)$, where w_v are positive vertex weights. The goal is to find an independent set[3] I that

[3] Recall that a set $I \subset V$ is an *independent set* if no edge in $e \in E$ has both its endpoints in I. A set $C \subset V$ is a *vertex cover* if every edge $e \in E$ has at least one endpoint in C. Note that I is an independent set if and only if $V \setminus I$ is a set cover.

maximizes $w(I)$. For this problem, $\mathcal{S}$ is the set of all independents sets I in G. For every vertex $v \in V$, there is a constraint c_v; $c_v(I) = 1$ if $v \in I$. The objective is to maximize $\sum_{c\in\mathcal{C}} w_c c(I)$. The problem complementary to Maximum Independent Set is *Minimum Vertex Cover.* In Minimum Vertex Cover, the objective is to find a vertex cover $C \subset V$ that minimizes $w(C)$; equivalently, the objective is to find an independent set I that minimizes $\sum_{c\in\mathcal{C}} w_c(1 - c(I))$.

Basic Properties of Certified Algorithms. Now, we discuss basic properties of certified algorithms. First, we show that certified algorithms provide an approximate solution for worst case instances and solve perturbation-resilient and weakly perturbation-resilient instances.

Theorem 1.11 *Consider a γ-certified algorithm $\mathcal{A}$.*

- *$\mathcal{A}$ finds a γ-approximate solution regardless of what the input instance is. Further, it finds a γ-approximation for both the maximization and minimization objectives.*
- *If the instance is γ-perturbation-resilient, $\mathcal{A}$ finds the optimal solution. If it is (γ, N)-weakly stable, $\mathcal{A}$ finds a solution in N.*

Proof Consider an instance $\mathcal{I}$. Denote its optimal solution by s^*. Denote the instance and solution found by $\mathcal{A}$ by $\mathcal{I}'$ and s'. For each constraint $c \in \mathcal{C}$, let w_c and w'_c be its weights in $\mathcal{I}$ and $\mathcal{I}'$, respectively.
I. First, we prove that the algorithm always gives a γ-approximation for both objectives. Consider the maximization objective. The value of s' (w.r.t. weights w_c) equals

$$\sum_{c\in\mathcal{C}} w_c c(s') \geq \sum_{c\in\mathcal{C}} \frac{w'_c}{\gamma} c(s') = \frac{1}{\gamma}\sum_{c\in\mathcal{C}} w'_c c(s') \overset{(\star)}{\geq} \frac{1}{\gamma}\sum_{c\in\mathcal{C}} w'_c c(s^*) \geq \frac{1}{\gamma}\sum_{c\in\mathcal{C}} w_c c(s^*),$$

where $(\star)$ holds since s' is an optimal solution for $\mathcal{I}'$. We conclude that s' is a γ-approximate solution for the maximization objective. Similarly, we upper bound the approximation factor for the minimization objective.

$$\sum_{c\in\mathcal{C}} w_c(1 - c(s')) \leq \sum_{c\in\mathcal{C}} w'_c(1 - c(s')) \leq \sum_{c\in\mathcal{C}} w'_c(1 - c(s^*)) \leq \gamma \sum_{c} w_c(1 - c(s^*)).$$

II. Now, assume that $\mathcal{I}$ is γ-perturbation-resilient. By the definition of perturbation resilience, $\mathcal{I}$ and $\mathcal{I}'$ have the same optimal solution. Thus, s^* is an optimal solution not only for $\mathcal{I}'$ but also for $\mathcal{I}$. Finally, assume that $\mathcal{I}$ is (γ, N)-weakly perturbation-resilient. Since $\mathcal{I}$ is (γ, N) weakly perturbation-resilient and $\mathcal{I}'$ is a γ-perturbation of $\mathcal{I}$, the optimal solution s' for $\mathcal{I}'$ must lie in N. □

We note that traditional approximation results for maximization and minimization objectives are often very different. For example, the algorithm for Max Cut by Goemans and Williamson (1995) gives an $\alpha_{GW} \approx 0.878$ approximation, while the best known approximation algorithm for Min Uncut gives only an $O(\sqrt{\log n})$

approximation (Agarwal et al., 2005). Similarly, Minimum Vertex Cover admits a 2-approximation algorithm, while its complement, Maximum Independent Set, does not even have an $n^{1-\delta}$ approximation if $\mathbf{P} \neq \mathbf{NP}$(for every $\delta > 0$).

Consider an instance of an optimization problem. We may choose a subset of constraints $H \subset \mathcal{C}$ and require that all of them be satisfied. We call them *hard constraints* and the obtained instance an instance with hard constraints. Formally, given an instance $(\mathcal{S}, \mathcal{C}, w)$ and a subset of constraints H, we define the correspondent instance $(\mathcal{S}', \mathcal{C}', w)$ with hard constraints as follows: $\mathcal{S}' = \{a \in \mathcal{S} : c(s) = 1$ for every $c \in H\}$; $\mathcal{C}' = \mathcal{C} \setminus H$; $w'(c) = w(c)$ for $c \in \mathcal{C}'$.

Theorem 1.12 *(Makarychev and Makarychev, 2019) Assume that there is a pseudo-polynomial-time γ-certified algorithm for a problem P, where $\gamma = \gamma_n$ is at most polynomial in n. Then there is also a pseudo-polynomial-time γ-certified algorithm for a variant P' of P with hard constraints. Accordingly, the maximization and minimization variants of P' admit γ-approximation algorithms.*

We leave the proof as an exercise (see Exercise 1.3).

Remark Constraint satisfaction problems (CSP) with hard constraints are often much harder for approximation algorithms than those without hard constraints. More precisely, algorithms for *minimization* CSPs without hard constraints typically can also solve instances with hard constraints. However, algorithms for *maximization* CSPs often cannot solve instances with hard constraints. For example, the algorithm for Max 2-SAT by Lewin et al. (2002) gives a 0.9401 approximation. However, there is no even an $n^{1-\delta}$ approximation algorithm for Max 2-SAT with hard constraints. The latter is also true for Max 2-Horn SAT (which is a variant of Max 2-SAT in which all the constraints are Horn clauses).

1.3 Designing Certified Algorithms

In this section, we will describe a general framework for designing certified algorithms, robust algorithms for perturbation-resilient instances, and algorithms for weakly perturbation-resilient instances, as well as proving that LP or SDP relaxations for perturbation-resilient instances are integral (Makarychev et al., 2014; Makarychev and Makarychev, 2019). To use this framework, one needs to either develop a procedure for solving a certain combinatorial task (see Task 1.13 and Lemma 1.14 below) or design a rounding scheme (procedure) that satisfies so-called approximation and co-approximation properties (see Theorems 1.17 and 1.19 below).

General Framework. Consider an optimization problem. We design a certified algorithm that (1) starts with an arbitrary solution and (2) then iteratively improves it. This approach is somewhat similar to local search, except that the improvements are not necessarily local. We show that it suffices to have a procedure for the following task.

Task 1.13 Assume that we are given an instance $\mathcal{I}(\mathcal{S}, \mathcal{C}, w)$, a partition of its constraints $\mathcal{C} = C_1 \cup C_2$, and a parameter $\gamma \geq 1$. The task is either

- *Option 1:* to find $s \in \mathcal{S}$ such that $\gamma \sum_{c \in C_1} w_c c(s) > \sum_{c \in C_2} w_c(1 - c(s))$, or
- *Option 2:* to report that for every $s \in \mathcal{S}$: $\sum_{c \in C_1} w_c c(s) \leq \sum_{c \in C_2} w_c(1 - c(s))$.

(Note that the above options are not mutually exclusive.)

When we use this procedure, C_1 and C_2 will be the sets of the constraints that are currently unsatisfied and satisfied, respectively. To give some intuition what Options 1 and 2 say, imagine that $\gamma = 1$. Then Option 1 is to find a solution s such that the weight of the currently *unsatisfied* constraints *satisfied* by s is greater than the weight of the currently *satisfied* constraints *unsatisfied* by s. In other words, Option 1 is to find a solution s better than the current solution. Option 2 is to report that there is no solution better than s.

Lemma 1.14 *Assume that (1) there is a polynomial-time algorithm for Task 1.13 above and (2) there is a polynomial-time algorithm that finds some solution $s \in \mathcal{S}$. Then there exists a pseudo-polynomial-time certified algorithm for the problem.*

Before we prove Lemma 1.14, we show how to get a certified algorithm for Max Cut and Min Uncut.

Theorem 1.15 *There exists a pseudo-polynomial-time γ-certified algorithm for Max Cut and Min Uncut, where $\gamma = O(\sqrt{\log n} \log\log n)$ is the approximation factor of the algorithm for Sparsest Cut with non-uniform demands by Arora et al. (2008).*

Proof To prove the theorem, we show how to solve Task 1.13 in polynomial time. Recall that in our formulation of Max Cut, $c_e(S, \bar{S}) = 1$ if edge e is cut. Let $E_1 = \{e \in E : c_e \in C_1\}$ and $E_2 = \{e \in E : c_e \in C_2\}$; denote the total weight of the edges in E_i cut by $(S, \bar{S})$ by $w(E_i(S, \bar{S}))$. Let $\phi(S) = \frac{w(E_2(S,\bar{S}))}{w(E_1(S,\bar{S}))}$. Then our goal is to either find a cut $(S, \bar{S})$ such that $\phi(S) < \gamma$ or report that $\phi(S) \geq 1$ for every $(S, \bar{S})$. Now the problem of minimizing $\phi(S)$ over all cuts $(S, \bar{S})$ is the same as finding the sparsest cut with non-uniform demands in graph (V, E_2) with edge capacities w, demand pairs E_1, and demand weights w. We run the approximation algorithm for Sparsest Cut and get a cut $(S, \bar{S})$ that approximately – within a factor of γ – minimizes $\phi(S)$. If $\phi(S) < \gamma$, we report cut $(S, \bar{S})$; otherwise, we report that $\phi(S') \geq 1$ for every cut $(S', \bar{S}')$. □

Our certified algorithm gives $\gamma_n = O(\sqrt{\log n} \log\log n)$ approximation for Max Cut and Min Uncut. Let us compare this result with known *approximation* results for Max Cut and Min Uncut. For Max Cut, we can obtain a much better approximation factor of $\alpha_{GW} \approx 0.878$ (Goemans and Williamson, 1995). However, for Min

Uncut, the best known approximation factor is $O(\sqrt{\log n})$ (Agarwal et al., 2005), which is comparable to γ_n. Note that there is also a γ_n-certified algorithm with an approximation factor of α_{GW}. The algorithm first finds an α_{GW} approximation for Max Cut, and then iteratively improves it as described in Theorem 1.15. Can the bound on γ_n be improved? It turns out that the optimal value of γ_n is essentially equal to the best approximation factor α_n for Sparsest Cut with non-uniform demands (see Makarychev et al. (2014) for details).

Proof of Lemma 1.14 As stated in the introduction, we assume that all weights w_c are integers between 1 and W. We first find a feasible solution s and then iteratively improve it.

Improvement Procedure. At each iteration, we let $C_1 = \{c \in \mathcal{C} : c(s) = 0\}$ and $C_2 = \{c \in \mathcal{C} : c(s) = 1\}$ be the sets of unsatisfied and satisfied constraints, respectively. Define weights w' as follows: $w'_c = w_c$ if $c \in C_1$ and $w'_c = \gamma w_c$ if $c \in C_2$. We run the procedure for Task 1.13 on instance $\mathcal{I}' = (\mathcal{S}, \mathcal{C}, w')$. Consider two cases. Assume first that the procedure returns a solution s' such that $\gamma \sum_{c\in C_1} w'_c c(s') > \sum_{c\in C_2} w'_c(1 - c(s'))$ (Option 1). We get that $\sum_{c\in C_1} w_c c(s') > \sum_{c\in C_2} w_c(1 - c(s'))$ and thus $\mathrm{val}_{\mathcal{I}}(s') = \sum_{c\in C_1\cup C_2} w_c c(s') > \sum_{c\in C_2} w_c = \mathrm{val}_{\mathcal{I}}(s)$. Therefore, solution s' improves s. We use this s' in the next iteration of the algorithm.

Assume now that the procedure reports that for every solution s': $\sum_{c\in C_1} w'_c c(s') \leq \sum_{c\in C_2} w'_c(1 - c(s'))$ (Option 2) or, equivalently, $\mathrm{val}_{\mathcal{I}'}(s') = \sum_{c\in C_1\cup C_2} w'_c c(s') \leq \sum_{c\in C_2} w'_c = \mathrm{val}_{\mathcal{I}'}(s)$ We return instance $\mathcal{I}'$ and solution s.

When the algorithm terminates, it outputs an instance $\mathcal{I}'$, which is a γ-perturbation of $\mathcal{I}$, and an optimal solution s for it. Thus, the algorithm is indeed a γ-certified algorithm. It remains to bound its running time. At each iteration, the value of the solution increases by at least 1 (recall that we have assumed that all weights are integers). Therefore, the algorithm terminates in at most $\sum_{c\in\mathcal{C}} w_c \leq |\mathcal{C}|W$ iterations. Since each iteration requires polynomial time, the running time is polynomial in n and W. □

Using Convex Relaxations. We now describe how to design certified algorithms using linear or semidefinite programming relaxations (or, in fact, any polynomially-tractable convex relaxations).

While our ultimate goal is to design a *certified* approximation algorithm, imagine for a moment that we simply want to design a regular approximation algorithm. One standard approach is to write a relaxation for the problem and design a rounding scheme for it. For example, to solve Maximum Independent Set (see Example 1.10), we can use the following linear programming (LP) relaxation:

$$\text{maximize} \sum_{u\in V} w_u x_u \tag{1.1}$$

$$\text{subject to: } x_u + x_v \leq 1 \text{ for } (u,v) \in E \text{ and } 0 \leq x_u \leq 1 \text{ for } u \in V$$

Our discussion below applies to any combinatorial optimization problem; but it might be instructive to keep relaxation (1.1) in mind. We refer to problem solutions $s \in \mathcal{S}$ as *combinatorial* solutions and relaxation solutions x as *fractional* solutions; we say that x is *integral* if it corresponds to a combinatorial solution $s \in \mathcal{S}$. We assume that in the relaxation we have a variable x_c for each constraint c so that $x_c = c(s)$ for every integral solution x and corresponding combinatorial solution s.[4]

Suppose first that we design an approximation algorithm for a maximization problem. Then the relaxation is to maximize $\mathrm{fval}(x) = \sum_{c \in \mathcal{C}} w_c x_c$ subject to certain problem specific constraints. We solve the relaxation, find a fractional solution x, and round it to a combinatorial solution $\mathcal{R}(x)$ using a randomized rounding scheme $\mathcal{R}$. Assume that the rounding scheme satisfies the following approximation condition for some $\alpha \geq 1$:

- **Approximation Condition.** The probability[5] that each constraint $c \in \mathcal{C}$ is satisfied by $\mathcal{R}(x)$ is at least x_c/α.

Then the expected weight of the constraints satisfied by $\mathcal{R}(x)$ is at least $\frac{\mathrm{fval}}{\alpha}$. Thus, we get a randomized α-approximation algorithm.

Suppose now that we design an algorithm for a minimization problem. Then the relaxation objective is to minimize $\sum_{c \in \mathcal{C}} w_c(1-x_c)$. Now, we use a rounding scheme that satisfies the following co-approximation condition for some $\beta \geq 1$:

- **Co-approximation Condition.** The probability that each constraint $c \in \mathcal{C}$ is unsatisfied by $\mathcal{R}(x)$ is at most $\beta(1 - x_c)$.

The expected weight of the unsatisfied constraints is at most $\beta \sum_{c \in \mathcal{C}} w_c(1-x_c)$. We get a β-approximation algorithm. We see that approximation and co-approximation conditions play a central role in the design of traditional approximation algorithms. It turns out that they can also be used to design *certified* algorithms.

Definition 1.16 We say that a rounding scheme $\mathcal{R}$ is an (α, β)-rounding if it simultaneously satisfies the approximation and co-approximation conditions with parameters α and β.

Theorem 1.17 *Assume that there exists an (α, β)-rounding scheme $\mathcal{R}$.*
I. Assume that $\mathcal{R}$ is computable in randomized polynomial time. Let $W = \frac{\max_{c \in \mathcal{C}} w_c}{\min_{c \in \mathcal{C}} w_c}$ be the ratio between the maximum and the minimum weights. Then there exists a randomized[6] *certified γ-approximation algorithm for the problem where $\gamma = \alpha\beta + \varepsilon$; its running time is polynomial in the instance size, W, and $1/\varepsilon$.*
II. Now we make a stronger assumption. Assume that the support of $\mathcal{R}$ is of polynomial size and can be found in polynomial time and that all the weights are integers

[4] Note that if we do not have variables x_c in the relaxation, we can usually add them, since expressions for them appear in the relaxation objective anyway.
[5] The probability is over the random choices made by $\mathcal{R}$.
[6] More precisely, the algorithm is a Las Vegas algorithm and as such, always outputs a correct solution.

between 1 *and* W. *Then there exists a certified* γ*-approximation algorithm where* $\gamma = \alpha\beta$*; its running time is polynomial in the instance size and* W.

In both cases, the solution s^* *returned by the algorithm is an optimal solution for the convex relaxation for* $\mathcal{I}'$.

Proof To simplify the exposition, we will only prove part II. The proof of part I is very similar but more technical. We refer the reader to Makarychev and Makarychev (2019) for details. Note that the condition that the support of $\mathcal{R}$ can be found in polynomial time is not very restrictive; most rounding schemes satisfy it.

We use Lemma 1.14 to design the algorithm. Namely, we show how to solve Task 1.13 in polynomial time. First, we solve the convex relaxation for the problem and obtain a fractional solution x. If $\sum_{c\in C_1} w_c x_c \le \sum_{c\in C_2} w_c(1-x_c)$, then for every s

$$\sum_{c\in C_1} w_c c(s) + \sum_{c\in C_2} w_c c(s) \le \sum_{c\in C_1} w_c x_c + \sum_{c\in C_2} w_c x_c \le \sum_{c\in C_2} w_c. \tag{1.2}$$

So we report that $\sum_{c\in C_1} w_c c(s) \le \sum_{c\in C_2} w_c(1-c(s))$ for every s (Option 2). In this case, the certified algorithm from Lemma 1.14 returns a solution s^* of value $w(C_2) = \sum_{c\in C_2} w_c$. Equation (1.2) shows that the value of every fractional solution (let alone integral) is at most $\mathrm{val}_{\mathcal{I}}(s^*) = w(C_2)$.

Assume now that $\sum_{c\in C_1} w_c x_c > \sum_{c\in C_2} w_c(1-x_c)$. We apply rounding scheme $\mathcal{R}$ and obtain a solution $\mathcal{R}(x)$. From the approximation and co-approximation conditions, we get

$$\begin{aligned}\mathbf{E}\left[\gamma \sum_{c\in C_1} w_c c(\mathcal{R}(x)) - \sum_{c\in C_2} w_c(1-c(\mathcal{R}(x)))\right] &\ge \frac{\gamma}{\alpha}\sum_{c\in C_1} w_c x_c - \beta \sum_{c\in C_2} w_c(1-x_c) \\ &\overset{\text{since } \gamma=\alpha\beta}{=} \beta\Big(\sum_{c\in C_1} w_c x_c - \sum_{c\in C_2} w_c(1-x_c)\Big) > 0.\end{aligned}$$

Thus for some solution s in the support of $\mathcal{R}(x)$, we have $\gamma\sum_{c\in C_1} w_c c(s) > \sum_{c\in C_2} w_c(1-c(s))$. We find and return such a solution s. □

As an immediate corollary, we get an algorithm for solving γ-perturbation-resilient and γ-weakly perturbation-resilient instances of combinatorial optimization problems. As we will describe below (see Theorem 1.19), it is actually sufficient to make slightly weaker assumptions to get algorithms for perturbation-resilient and weakly perturbation-resilient instances. Before we state Theorem 1.19, let us discuss how we can relax the conditions in Theorem 1.17. First, it is sufficient to design a rounding scheme that only rounds an optimal fractional solution. For some problems, this may be an easier task as the optimal fractional solution may satisfy certain additional properties (e.g., be half-integral). Also, it is sufficient to design a rounding scheme that only rounds fractional solutions that are close to integral solutions.

Definition 1.18 Let us say that a fractional solution x is δ-close to integral if $x = (1-\delta)x^{int} + \delta x^{frac}$ for some integer solution x^{int} and fractional solution x^{frac} (for LP relaxations this condition implies that each LP variable x_c is in $[0,\delta]\cup[1-\delta,1]$). Rounding scheme $\mathcal{R}$ is a δ-local (α,β)-rounding if it is defined and satisfies the approximation and co-approximation conditions for fractional solutions x that are δ-close to an integral solution; the rounding scheme may but does not have to be defined or satisfy the approximation and co-approximation conditions for fractional solutions that are not δ-close to integral solutions.

Remark It is sufficient to have a δ-local rounding scheme (with $\delta \geq 1/\operatorname{poly}(n)$) in Theorem 1.17. Designing such a scheme may be a considerably easer task than designing a rounding scheme for arbitrary solutions. If we have such a scheme, we proceed as follows. Denote the fractional solution corresponding to the combinatorial solution s by $x^{(s)}$. We find an optimal fractional solution x^* and then let $x = (1-\delta)x^{(s)} + \delta x^*$. Note that x is δ-close to integral. It is easy to see that if x^* is better than $x^{(s)}$, then so is x. Then we use x in the proof of Theorem 1.17 (see Makarychev and Makarychev (2019) for details).

Now, we describe a condition under which polynomial-time algorithms for solving γ-perturbation-resilient and $(\gamma+\varepsilon, N)$-weakly stable instances of combinatorial optimization problems are guaranteed to exist. Note that we do not make any assumptions about the weights w_c.

Theorem 1.19 *Makarychev and Makarychev (2016); Angelidakis et al. (2017) Assume that there is an (α,β)-rounding scheme or a δ-local (α,β)-rounding scheme $\mathcal{R}$. Let $\gamma = \alpha\beta$. Then we have:*

- *The convex relaxation is integral for γ-perturbation-resilient instances. $\mathcal{R}$ does not have to be computable in polynomial-time.*
- *There exists a robust polynomial-time algorithm for solving γ-perturbation-resilient instances. The running time depends only on the size of the input. Again, $\mathcal{R}$ does not have to be computable in polynomial-time (we use $\mathcal{R}$ only in the analysis of the algorithm).*
- *Assume that the support of $\mathcal{R}(x)$ is of polynomial size and can be found in polynomial time and that $\varepsilon, \delta \geq 1/\operatorname{poly}(n)$. Then there exists a polynomial-time algorithm for solving $(\gamma+\varepsilon)$-weakly perturbation-resilient instances.*

1.4 Examples of Certified Algorithms

Maximum Independent Set. We now prove that there exist a $(k-1)$-certified algorithm for Maximum Independent Set (MIS) in k-colorable graphs and a robust algorithm for $(k-1)$-perturbation-resilient instances of the problem (see Example 1.10 for the definition of MIS). To get the algorithms, we follow the approach we discussed in the previous section and design an (α, β)-rounding scheme with $\alpha\beta = k - 1$.

Consider a k-colorable graph $G = (V, E, w)$. Solve relaxation (1.1) for MIS. Let x be an optimal *vertex* solution. It is known that x is half-integral (Nemhauser and Trotter, 1975). Define $V_t = \{u \in V : x_u = t\}$ for $t \in \{0, 1/2, 1\}$. Consider the following rounding scheme due to Hochbaum (1983) (the rounding scheme needs to know a proper k-coloring $(C_1, \dots, C_k)$ of V).

Rounding scheme $\mathcal{R}$

```
choose $i \in \{1, \dots, k\}$ uniformly at random
return $S = V_1 \cup (V_{1/2} \cap C_i)$.
```

Theorem 1.20 *(Angelidakis et al. (2019)) $\mathcal{R}$ is an (α, β)-rounding for MIS with $\alpha = k/2$ and $\beta = 2(k-1)/k$. Given the coloring, the rounding algorithm outputs a distribution of independent sets in polynomial time; the distribution support is of polynomial size.*

Proof It is easy to see that the rounding scheme always outputs an independent set S. If $u \in V_1$, then $u \in S$ (always); if $u \in V_0$, then $u \notin S$ (always) – in these two cases there is no randomness involved, and the approximation and co-approximation conditions trivially hold. Now if $u \in V_{1/2}$, then $u \in S$ with probability $1/k$ (this happens when i is the color of u). The approximation condition holds since $\Pr(u \in S) = 1/k = x_u/\alpha$; the co-approximation condition holds since $\Pr(u \notin S) = \frac{k-1}{k} = \beta(1 - x_u)$. □

We conclude that there exists a polynomial-time $(k-1)$-certified algorithm for MIS and a robust polynomial-time algorithm for $(k-1)$-perturbation-resilient instances. Note that the certified algorithm needs to know the k-coloring of the graph, but the robust algorithm does not; the algorithm simply solves the LP relaxation and outputs the solution, which is guaranteed to be integral (see Theorem 1.19).

Minimum Multiway Cut. Now, we design certified and robust algorithms for Minimum Multiway Cut (see Example 1.9 for the definition of the problem). To simplify the exposition, we focus on the case $k = 3$. Consider the LP relaxation by Călinescu et al. (1998) for the problem. For every vertex u, there is a vector $\bar{u} = (u_1, u_2, u_3)$ in the LP. In the integral solution corresponding to a partition (P_1, P_2, P_3): $u_i = 1$, if $u \in P_i$; and $u_i = 0$, otherwise. That is, $\bar{u} = e_i$ (the i-th standard basis vector) if $u \in P_i$. The objective is to minimize $\frac{1}{2}\sum_{e=(u,v)\in E} w(e)\|\bar{u} - \bar{v}\|_1$ subject to (i) $\bar{t}_j = e_j$ for all $j \in \{1, 2, 3\}$, (ii) $u_1 + u_2 + u_3 = 1$ for all $u \in V$, and (iii) $u_j \geq 0$ for all $u \in V$, $j \in \{1, 2, 3\}$. It is easy to see that by adding auxiliary variables, we can write the objective as a linear function. Let $d(\bar{u}, \bar{v}) = \frac{1}{2}\|\bar{u} - \bar{v}\|_1$. The relaxation requires that each vector $\bar{u}$ lie in the triangle $\Delta = \mathrm{conv}(e_1, e_2, e_3)$ with vertices e_1, e_2, e_3. Our goal is to design a δ-local (α, β)-rounding for Minimum Multiway Cut with $\alpha\beta = 4/3$ and $\delta = 1/30$. As is standard for approximation

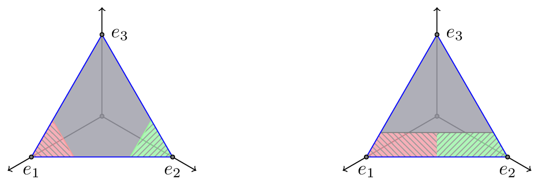

Figure 1.1 Each vector $\bar{u}$ lies in the triangle with vertices e_1, e_2, e_3. The left figure shows the 2-vertex cut of radius 3/10 with pivot $i = 3$; the right figure shows the ball cut of radius 4/5 with pivot $i = 3$.

algorithms for Minimum Multiway Cut, we will consider a rounding scheme that (randomly) cuts triangle Δ into 3 pieces $\hat{P}_1, \hat{P}_2, \hat{P}_3$ so that $e_i \in \hat{P}_i$ and then lets $P_i = \{u : \bar{u} \in \hat{P}_i\}$ for each i. It is immediate that the rounding gives a feasible solution, since $\bar{t}_i = e_i \in \hat{P}_i$. We will describe how to cut triangle Δ so that the obtained rounding scheme is a δ-local (α, β)-rounding.

We are going to define two families of cuts, *two-vertex cuts* and *ball cuts* (introduced by Karger et al. (2004)). Given a vertex e_i and radius $r \in (0, 1)$, let $B_r(e_i) = \{\bar{x} : d(\bar{x}, e_i) \leq r\}$ be the ball of radius r around e_i w.r.t. distance d. Geometrically, $B_r(e_i)$ is the triangle with vertices e_i, $(1-r)e_i + re_{j_1}$, and $(1-r)e_i + re_{j_2}$, where e_{j_1}, e_{j_2} are the basis vectors other than e_i. The two-vertex cut of radius $r \in (0, 1)$ with pivot $i \in \{1, 2, 3\}$, shown on Figure 1.1, is defined by: $\hat{P}_j = B_r(e_j)$ for $j \in \{j_1, j_2\}$ and $\hat{P}_i = \Delta \setminus (\hat{P}_{j_1} \cup \hat{P}_{j_2})$. The ball cut of radius $r \in (0, 1)$ with pivot $i \in \{1, 2, 3\}$, shown on Figure 1.1, is defined by: $\hat{P}_i = B_r(e_i)$, every point $\bar{x} \notin \hat{P}_i$ belongs to either $\hat{P}_{j_1}$ or $\hat{P}_{j_2}$ depending on whether it lies closer to e_{j_1} or e_{j_2} w.r.t. distance d. Now we are ready to present the rounding scheme.

Rounding scheme $\mathcal{R}$

```
  choose $r \in (0, 2/5)$ uniformly at random
  choose pivot $i \in \{1, 2, 3\}$ uniformly at random
  with probability 1/3, let $(\hat{P}_1, \hat{P}_2, \hat{P}_3)$ be the two-corner cut of radius $r$ with pivot $i$
  otherwise, let $(\hat{P}_1, \hat{P}_2, \hat{P}_3)$ be the ball cut of radius $1 - r$ with pivot $i$
  let $P_j = \{u \in V : \bar{u} \in \hat{P}_j\}$ for $j \in \{1, 2, 3\}$
  return partition $P = (P_1, P_2, P_3)$
```

Theorem 1.21 *(Angelidakis et al. (2017)) $\mathcal{R}$ is a δ-local (α, β)-rounding for Minimum Multiway Cut with $\alpha = 10/9$ and $\beta = 6/5$.*

We leave the proof as an exercise (see Exercise 1.4). We conclude that there exists a polynomial-time 4/3-certified algorithm for Multiway Cut with 3 terminals and a robust polynomial-time algorithm for 4/3-perturbation-resilient instances. These results generalize to $(2 - 2/k)$-perturbation-resilient instances of Multiway Cut with k terminals; see Angelidakis et al. (2017).

1.5 Perturbation-resilient Clustering Problems

In this section, we consider k-clustering problems with the ℓ_p objective. This is a broad class of problems which includes k-means, k-medians, and k-center.

Definition 1.22 (k-clustering with the ℓ_p objective) An instance of k-clustering with the ℓ_p objective ($p \geq 1$) consists of a metric space (X, d) and a natural number k. The goal is to partition X into k disjoint clusters $C_1, \dots, C_k$ and assign a center c_i to each cluster C_i so as to minimize the following objective function:

$$\sum_{i=1}^{k} \sum_{u \in C_i} d^p(u, c_i).$$

For $p = \infty$, the objective function is $\max_{\substack{i \in \{1,\dots,k\}\\ u \in C_i}} |d(u, c_i)|$.

Note that k-medians is the k-clustering problem with the ℓ_1 objective; k-means is k-clustering with the ℓ_2 objective; and k-center is k-clustering with the ℓ_∞ objective.

Consider an instance (X, d) of k-clustering with the ℓ_p objective. In an optimal solution to this problem, each point is assigned to the closest center $c_1, \dots, c_k$. That is, if $u \in C_i$, then $d(u, c_i) \leq d(u, c_j)$ for all $j \neq i$. This is an important property common to all so-called *clustering problems with a center based objective.* Note that the optimal clustering $C_1, \dots, C_k$ is determined by the centers $c_1, \dots, c_k$. Specifically, $\{C_1, \dots, C_k\}$ is the Voronoi partition for $c_1, \dots, c_k$; that is, C_i is the set of points in X that are closer to c_i than to any other c_j.

Now let us assume that the distance from every point in X to its own center is less than the distances to other centers by a certain margin. Specifically, suppose that there exists an optimal clustering $C_1, \dots, C_k$ with centers $c_1, \dots, c_k$ that satisfies the following condition called λ-center proximity: for every $u \in C_i$, not only $d(u, c_i) \leq d(u, c_j)$ but also $\lambda d(u, c_i) < d(u, c_j)$.

Definition 1.23 (λ-center proximity) Let (X, d) be an instance of the k-clustering problem with the ℓ_p objective. Consider an optimal solution $C_1, \dots, C_k$ with centers $c_1, \dots, c_k$. We say that $c_1, \dots, c_k$ satisfies the λ-center proximity condition (where $\lambda \geq 1$) if for every $u \in C_i$ and $j \neq i$, we have $\lambda d(u, c_i) < d(u, c_j)$.

We say that (X, d) has an optimal solution satisfying the λ-center proximity condition if there exists an optimal solution $C_1, \dots, C_k$ with centers $c_1, \dots, c_k$ satisfying the λ-center proximity condition.

The optimal set of centers is not necessarily unique for a given clustering $C_1, \dots, C_k$. Some optimal sets of centers for $C_1, \dots, C_k$ may satisfy the λ-center proximity condition, while others do not.

In Section 1.6, we show that there exists an algorithm – a variant of the classic single-linkage clustering – that finds the optimal clustering if this clustering satisfies the 2-center proximity condition for some set of optimal centers. We note that it is

NP-hard to find the optimal clustering in instances satisfying λ-center proximity condition with $\lambda < 2$ (Ben-David and Reyzin, 2014). Now we restate Definitions 1.1 and 1.2 taking into account specifics of clustering problems.

Definition 1.24 (Perturbations and Metric Perturbations) Consider a metric space (X, d). We say that a symmetric function $d' : X \times X \to \mathbf{R}^+$ is a γ-perturbation of d if for all $u, v \in X$ we have $\frac{1}{\gamma} d(u,v) \leq d'(u,v) \leq d(u,v)$. We say that d' is a *metric* γ-perturbation of d if d' is a γ-perturbation of d and a metric itself i.e., d' satisfies the triangle inequality.

Note that a (non-metric) γ-perturbation d' may violate the triangle inequality and thus is not necessarily a metric.

Definition 1.25 (Perturbation Resilience) Consider an instance (X, d) of the k-clustering problem with the ℓ_p objective. Let $C_1, \dots, C_k$ be the optimal clustering. Then, (X, d) is *γ-perturbation-resilient* if for every γ-perturbation of d, the unique optimal clustering of (X, d') is $C_1, \dots, C_k$. Similarly, (X, d) is *metric γ-perturbation-resilient* if for every metric γ-perturbation of d, the unique optimal clustering of (X, d') is $C_1, \dots, C_k$.

The definition of metric γ-perturbation resilience is less restrictive than that of γ-perturbation resilience: if an instance is γ-perturbation resilient, it is also *metric* γ-perturbation-resilient but not the other way around. In particular, every algorithm that solves metric γ-perturbation-resilient instances also solves γ-perturbation resilient instances.

Note that in the definition of γ-perturbation resilience we do not require that the optimal centers of the clusters $C_1, \dots, C_k$ are the same for distance functions d and d'. If we added this requirement we would get a much stronger definition of γ-perturbation resilience or metric γ-perturbation resilience (see Exercise 1.7).

Perturbation resilience is a stronger notion than center proximity: Every γ-perturbation-resilient instance satisfies the γ-center proximity condition. We prove this implication in Theorem 1.27. The converse statement does not hold and, thus, these notions are not equivalent (see Exercise 1.10).

In this chapter, we present two results on γ-perturbation resilience. First, we give a dynamic programming algorithm that finds an exact solution to any 2-center proximity instance of the k-clustering problem (see Theorem 1.29). Since every metric 2-perturbation-resilient instance satisfies the 2-center proximity condition (see Theorem 1.27), our algorithm also works for metric 2-perturbation-resilient instances. Then, we discuss a connection between perturbation resilience and local search and show that the standard local search algorithm for k-medians is a $(3 + \varepsilon)$-certified algorithm. Thus, this algorithm returns the optimal clustering for γ-perturbation-resilient instances and gives a $(3 + \varepsilon)$-approximation for arbitrary instances.

Open Question 1.26 Suppose we replace the requirement that d' be a γ-perturbation of d with the requirement that d' be a *metric* γ-perturbation of d in the definition of a γ-certified algorithm (see Definition 1.4). Can we design a $(3+\varepsilon)$-certified algorithm according to the new definition?

1.5.1 Metric Perturbation Resilience Implies Center Proximity

We show that metric perturbation resilience implies center proximity.

Theorem 1.27 (Metric Perturbation Resilience Implies Center Proximity, Awasthi et al. (2012) and Angelidakis et al. (2017)) *Let (X, d) be a metric γ-perturbation-resilient instance of the k-clustering problem with the ℓ_p objective ($p \geq 1$). Consider the unique optimal solution $\mathcal{C} = (C_1, \dots, C_k)$ and an optimal set of centers $\{c_1, \dots, c_k\}$ (which is not necessarily unique). Then, centers $c_1, \dots, c_k$ satisfy the γ-center proximity property.*

Proof Consider an arbitrary point p in X. Let c_i be the closest center to p in $\{c_1, \dots, c_k\}$ and c_j be another center. We need to show that $d(p, c_j) > \gamma d(p, c_i)$. Suppose that $d(p, c_j) \leq \gamma d(p, c_i)$. Let $r^* = d(p, c_i)$. Define a new metric d'. Consider the complete graph $G = (X, E)$ on the metric space X. Let the length $\mathrm{len}(u, v)$ of every edge (u, v) be $d(u, v)$. Then, $d(u, v)$ is the shortest path metric on G. We now shorten edge (p, c_j) while preserving the lengths of all other edges. Specifically, we let $\mathrm{len}'(p, c_j) = r^*$ and $\mathrm{len}'(u, v) = d(u, v)$ for $(u, v) \neq (p, c_j)$. Let d' be the shortest path metric on graph G with edge lengths $\mathrm{len}'(u, v)$. Observe that $d'(u, v) = d(u, v)$ unless there is a shortcut that goes along the edge (p, c_j). That is, the distance $d'(u, v)$ between any two points u and v equals the length of the shortest of the following three paths: (1) $u \to v$, (2) $u \to p \to c_j \to v$, and (3) $u \to c_j \to p \to v$. Thus,

$$d'(u, v) = \min\big(d(u, v), d(u, p) + r^* + d(c_j, v), d(u, c_j) + r^* + d(p, v)\big).$$

Note that $\mathrm{len}(u, v)/\gamma \leq \mathrm{len}'(u, v) \leq \mathrm{len}(u, v)$. Hence, $d(u, v)/\gamma \leq d'(u, v) \leq d(u, v)$ and thus $d'(u, v)$ is a γ-perturbation. Thus, the optimal clustering of X for d' is the same as for d. Namely, it is $C_1, \dots, C_k$. However, generally speaking, the optimal centers for clusters $C_1, \dots, C_k$ may differ for metrics d and d' (for some γ-perturbations they do!). Nevertheless, we claim that c_i and c_j are optimal centers for clusters C_i and C_j with respect to metric d'. This leads to a contradiction with our assumption that $d(p, c_j) \leq \gamma d(p, c_i)$, because p must be closer to its own center c_i than to c_j and, consequently, we must have $d(p, c_i) = d'(p, c_i) < d'(p, c_j) = d(p, c_i)$.

Therefore, to finish the proof we need to show that c_i and c_j are optimal centers for clusters C_i and C_j with respect to the metric d'. To this end, we prove that the metric d' equals d within clusters C_i and C_j and, hence any optimal center for C_i w.r.t. d is also an optimal center w.r.t d' and vice versa. □

Lemma 1.28 *For all $u, v \in C_i$, we have $d(u,v) = d'(u,v)$. Also, for all $u, v \in C_j$, we have $d(u,v) = d'(u,v)$.*

Proof To prove that $d'(u,v) = d(u,v)$, we need to show that $d(u,v) < \min(d(u,p) + r^* + d(c_j,v), d(u,c_j) + r^* + d(p,v))$. Assume without loss of generality that $d(u,p) + r^* + d(c_j,v) \le d(u,c_j) + r^* + d(p,v)$. Then,

$$d(u,p) + r^* + d(c_j,v) = \big(\underbrace{d(u,p) + d(p,c_i)}_{\ge d(u,c_i)}\big) + d(c_j,v) \ge d(u,c_i) + d(c_j,v).$$

1. If $u, v \in C_i$, then the closest center to v is c_i and, particularly, $d(c_j,v) > d(c_i,v)$. Thus, $d(u,p) + r^* + d(c_j,v) > d(u,c_i) + d(c_i,v) \ge d(u,v)$.
2. If $u, v \in C_j$, then the closest center to u is c_j and, particularly, $d(u,c_i) > d(u,c_j)$. Thus, $d(u,p) + r^* + d(c_j,v) > d(u,c_j) + d(c_j,v) \ge d(u,v)$. □

1.6 Algorithm for 2-Perturbation-resilient Instances

In this section, we prove that a variant of the single-linkage clustering algorithm finds the exact optimal solution for instances of clustering problems with the ℓ_p objective satisfying the 2-center proximity condition (or, more formally: instances that have an optimal solution $C_1, \dots, C_k$ with centers $c_1, \dots, c_k$ that satisfy the 2-center proximity condition).

Single-linkage clustering is a classic algorithm that works as follows. Given a metric space (X,d) on n points, it creates n clusters each containing a single point from X. Then, at every step it picks the two closest clusters and merges them. The distance between clusters is usually defined as the distance between the two closest points in the clusters i.e., $d(C', C'') = \min_{\substack{u \in C' \\ v \in C''}} d(u,v)$. Thus, at every step the number of clusters decreases by 1. The algorithm stops when only k clusters remain.

Single-linkage clustering is a fairly simple and relatively fast algorithm. However, it fails to find an optimal clustering when the clusters are not isolated from each other. It is also very fragile to noise because adding just a few extra points to the data set X can drastically alter the output. We cannot use single-linkage clustering as is for perturbation-resilient instances, since this algorithm may output a very bad clustering even if the instance is γ-perturbation-resilient with arbitrarily large γ (see Exercise 1.11) and for this reason will use a dynamic programming-based postprocessing step.

Theorem 1.29 (Angelidakis et al. (2017)) *There exists a polynomial-time algorithm that given an instance (X,d) of k-clustering with the ℓ_p objective outputs an optimal solution if (X,d) has an optimal solution satisfying the 2-center proximity condition.*

Algorithm. Consider the complete graph G on X, in which every edge (u, v) has length $d(u, v)$. Our algorithm first constructs the minimum spanning tree (MST) T in G and then clusters T using dynamic programming. To construct the MST, we can use one of many known algorithms, particularly Kruskal's algorithm which is essentially a variant of single-linkage clustering. We describe the dynamic program later in this section. Now we show that if an instance has an optimal solution satisfying the 2-center proximity condition then all clusters in that solution must form connected components in the minimum spanning tree.

Theorem 1.30 *Consider an instance (X, d) of k-clustering with the ℓ_p objective. Let $C_1, \dots, C_k$ be an optimal clustering with centers $c_1, \dots, c_k$ satisfying the 2-center proximity condition; and let $T = (X, E)$ be the minimum spanning tree (MST) in the complete graph on X with edge lengths $d(u, v)$. Then, each cluster C_i is a subtree of T (i.e., for every two vertices $u, v \in C_i$, the unique shortest path from u to v in T completely lies within C_i).*

We will need the following lemma.

Lemma 1.31 *Consider an instance (X, d) of the k-clustering problem with the ℓ_p objective. Suppose that $C_1, \dots, C_k$ is an optimal clustering for (X, d) and $c_1, \dots, c_k$ is an optimal set of centers. If $c_1, \dots, c_k$ satisfy the 2-center proximity property, then for every two distinct clusters C_i and C_j and all points $u \in C_i$ and $v \in C_j$, we have $d(u, c_i) < d(u, v)$.*

Proof Since $c_1, \dots, c_k$ satisfy the 2-center proximity property, we have $2d(u, c_i) < d(u, c_j)$ and $2d(v, c_j) < d(v, c_i)$. Thus, by the triangle inequality, $2d(u, c_i) < d(u, v) + d(v, c_j)$ and $2d(v, c_j) < d(u, v) + d(u, c_i)$. We sum these inequalities with coefficients 2/3 and 1/3 and obtain the desired bound: $d(u, c_i) < d(u, v)$. □

Proof of Theorem 1.30 Since T is a tree, it suffices to show that for every $u \in C_i$, all points on the unique path in T from u to c_i lie in C_i. Consider an arbitrary point $u \in C_i$ and denote the path from u to c_i by $u_1, \dots, u_M$, where $u_1 = u$ and $u_M = c_i$. We prove by induction on m that all u_m $(m = 1, \dots, M)$ are in C_i. The point $u_1 = u$ is in C_i. Also, $u_M \in C_i$ because $u_M = c_i$ is the center of C_i. Suppose that $u_m \in C_i$ and $m < M - 1$, we show that $u_{m+1} \in C_i$. By the MST cycle property, (u_M, u_m) is the longest edge in the cycle $u_m \to u_{m+1} \to \cdots \to u_M \to u_m$ (since all edges in the cycle but edge (u_M, u_m) belong to the MST). Particularly, $d(u_m, c_i) \equiv d(u_m, u_M) \geq d(u_m, u_{m+1})$. By the induction hypothesis $u_m \in C_i$. Therefore, by Lemma 1.31, u_{m+1} also belongs to C_i (because if u_{m+1} was not in C_i we would have $d(u_m, c_i) < d(u_m, u_{m+1})$). □

Dynamic Program. We now describe a dynamic program for finding the optimal clustering in the MST. Let us choose an arbitrary vertex r in X as a root for the MST T. Denote by T_u the subtree rooted at vertex u. We define two types of subproblems OPT and OPT_{AC}:

1. Let $\mathrm{OPT}(u, m)$ be the optimal cost of partitioning subtree T_u into m clusters that are subtrees of T.
2. Let $\mathrm{OPT}_{AC}(u, m, c)$ be the optimal cost of partitioning subtree T_u into m clusters subject to the following constraint: vertex u and all points in its cluster must be assigned to the center c.

The cost of k-clustering X equals $OPT(r, k)$. For simplicity, we assume that the MST is a binary tree (the general case can be handled by transforming any tree to a binary tree by adding "dummy" vertices). Let $\mathrm{left}(u)$ be the left child of u and $\mathrm{right}(u)$ be the right child of u.

We write recurrence relations on OPT and OPT_{AC}. To compute $\mathrm{OPT}(u, m)$ we need to find the optimal center for u and return $\mathrm{OPT}(u, m, c)$. Thus,

$$\mathrm{OPT}(u, m) = \min_{c \in X} \mathrm{OPT}_{AC}(u, m, c).$$

To find $\mathrm{OPT}_{AC}(u, m, c)$, we find the optimal solutions for the left and right subtrees and combine them. To this end, we need to guess the number of clusters m_L and m_R in the left and right subtrees. We present formulas for $\mathrm{OPT}_{AC}(u, m, c)$ in the four possible cases.

1. If both children $\mathrm{left}(u)$ and $\mathrm{right}(u)$ are in the same cluster as u, then
$$\min_{\substack{m_L, m_R \in \mathbf{Z}^+ \\ m_L + m_R = m+1}} d(c, u) + \mathrm{OPT}_{AC}(\mathrm{left}(u), c, m_L) + \mathrm{OPT}_{AC}(\mathrm{right}(u), c, m_R).$$
2. If $\mathrm{left}(u)$ is in the same cluster as u, but $\mathrm{right}(u)$ is in a different cluster, then
$$\min_{\substack{m_L, m_R \in \mathbf{Z}^+ \\ m_L + m_R = m}} d(c, u) + \mathrm{OPT}_{AC}(\mathrm{left}(u), c, m_L) + \mathrm{OPT}(\mathrm{right}(u), m_R).$$
3. If $\mathrm{right}(u)$ is in the same cluster as u, but $\mathrm{left}(u)$ is in a different cluster, then
$$\min_{\substack{m_L, m_R \in \mathbf{Z}^+ \\ m_L + m_R = m}} d(c, u) + \mathrm{OPT}(\mathrm{left}(u), m_L) + \mathrm{OPT}_{AC}(\mathrm{right}(u), c, m_R).$$
4. If u, $\mathrm{left}(u)$, and $\mathrm{right}(u)$ are in different clusters, then
$$\min_{\substack{m_L, m_R \in \mathbf{Z}^+ \\ m_L + m_R = m-1}} d(c, u) + \mathrm{OPT}(\mathrm{left}(u), m_L) + \mathrm{OPT}(\mathrm{right}(u), m_R).$$

We compute the values of $\mathrm{OPT}_{AC}(u, m, c)$ in the four cases above and choose the minimum among them.

The sizes of the DP tables for OPT and OPT_{AC} are $O(n \times k)$ and $n \times k \times n) = O(n^2 k)$, respectively. It takes $O(n)$ and $O(k)$ time to compute each entry in the tables OPT and OPT_{AC}, respectively. Thus, the total running time of the DP algorithm is $O(n^2 k^2)$. The running time of Prim's MST algorithm is $O(n^2)$.

1.7 $(3+\varepsilon)$-Certified Local Search Algorithm for k-medians

A common heuristic for clustering as well as for related problems such as Facility Location is local search (see also Chapter 13). Sometimes, local search algorithms are used on their own and sometimes to process the output of other algorithms. It is known that local search gives a $(3+\varepsilon)$-approximation for k-medians and a $(9+\varepsilon)$-approximation for k-means (Arya et al. (2004); Kanungo et al. (2004)), where $\varepsilon > 0$ is arbitrary. The running time is exponential in $1/\varepsilon$. We will see that local search is a $(3+\varepsilon)$-certified algorithm k-medians.

Below, we will focus on the k-medians problem though similar results hold for any clustering problems with the ℓ_p objective. Consider an arbitrary set of centers $c_1, \dots, c_k$. The optimal clustering $C_1, \dots, C_k$ for this set of centers is defined by the Voronoi partition i.e., $u \in C_i$ if and only if c_i is the closest center to u (ties between centers are broken arbitrarily). Denote by $\text{cost}(c_1, \dots, c_k)$ its cost.

We now describe a 1-local search algorithm. The algorithm maintains a set of k centers $c_1, \dots, c_k$. It starts with an arbitrary set of centers $c_1, \dots, c_k$. Then, at every step, it considers all possible swaps $c_i \to u$, where c_i is a center in the current set of centers, and u is a point outside of this set. If we can improve the solution by swapping c_i with u, we perform this swap. In other words, if for some pair (c_i, u), we have $\text{cost}(c_1, \dots, c_{i-1}, u, c_{i+1}, \dots, c_k) < \text{cost}(c_1, \dots, c_k)$, then we replace the center c_i with u. The algorithm terminates when no swap $c_i \to u$ can improve the solution. We call the obtained set of centers *1-locally optimal* and denote it by L. A more powerful (alas less practical) version of the local search algorithm considers swaps of size up to ρ instead of 1. We call this algorithm the ρ-local search algorithm. Its running time is exponential in ρ.

Theorem 1.32 (Cohen-Addad and Schwiegelshohn (2017) and Balcan and White (2017)) *The ρ-local search algorithm for k-medians outputs the optimal solution on $(3 + O(1/\rho))$-perturbation-resilient instances*[7].

This result follows from the following theorem.

Theorem 1.33 *The ρ-local search algorithm for k-medians is $(3 + O(1/\rho))$-certified.*

Proof Consider an arbitrary metric space (X, d). Suppose that the local search algorithm outputs a clustering with centers $L = \{l_1, \dots, l_k\}$. We show that there exists a γ-perturbation of d – a distance function $d' : X \times X \to \mathbf{R}^+$ for which L is the optimal solution (where $\gamma = 3 + O(1/\rho)$). We note that, generally speaking, d' does not have to satisfy the triangle inequality and thus (X, d') is not necessarily a metric space.

We define d' as follows: If l_i is the closest center to u in L, then $d'(u, l_i) =$

[7] In contrast to Theorem 1.29, Theorem 1.32 requires that the instance be perturbation resilient, not only *metric* perturbation resilient.

$d(u,l_i)/\gamma$; otherwise, $d'(u,l_i) = d(u,l_i)$. Consider an arbitrary set of centers $S = \{s_1,\dots,s_k\}$. We need to show that the cost of the k-median clustering with centers in L is at most the cost of the k-median clustering with centers in S with respect to the new distance function d' and thus L is an optimal solution for d'. Let $l(u)$ and $s(u)$ be the closest centers to point u in L and S respectively with respect to d; and let $l'(u)$ and $s'(u)$ be the closest centers to point u in L and S respectively with respect to d'. Our goal is to prove that

$$\sum_{u\in X} d'(u,l'(u)) \le \sum_{u\in X} d'(u,s'(u)). \tag{1.3}$$

Observe that for every point $u \in X$, we have $d(u,v) = d'(u,v)$ for all v but $v = l(u)$. Thus, $l'(u) = l(u)$ and $d'(u,l'(u)) = d(u,l(u))/\gamma$. Consequently, the left hand side of (1.3) equals $\sum_{u\in X} d(u,l(u))/\gamma$. Similarly, $s'(u) = s(u)$ and $d'(u,s'(u)) = d(u,s(u))$ if $l(u) \notin S$. However, if $l(u) \in S$, then $d'(u,s'(u)) = \min\big(d(u,s(u)), d(u,l(u))/\gamma\big)$ as, in this case, the optimal center for u in S w.r.t. d' can be $l(u)$.

Let us split all vertices in X into two groups $A = \{u : l(u) \in S\}$ and $B = \{u : l(u) \notin S\}$. Then, for $u \in A$, we have $d'(u,s'(u)) = \min\big(d(u,s(u)), d(u,l(u))/\gamma\big)$; and for $u \in B$, we have $d'(u,s'(u)) = d(u,s(u))$. Thus, inequality (1.3) is equivalent to

$$\sum_{u\in X} \frac{d(u,l(u))}{\gamma} \le \sum_{u\in A} \min\Big(d(u,s(u)), \frac{d(u,l(u))}{\gamma}\Big) + \sum_{u\in B} d(u,s(u)),$$

which after multiplying both parts by γ can be written as

$$\sum_{u\in X} d(u,l(u)) \le \sum_{u\in A} \min\Big(\gamma d(u,s(u)), d(u,l(u))\Big) + \sum_{u\in B} \gamma d(u,s(u)). \tag{1.4}$$

For $u \in A$, we have $d(u,s(u)) \le d(u,l(u))$ since both $s(u)$ and $l(u)$ are in S and $s(u) = \arg\min_{v\in S} d(u,v)$. Thus, $\min\big(\gamma d(u,s(u)), d(u,l(u))\big) \ge d(u,s(u))$. Consequently, inequality (1.4) follows from the following theorem. $\square$

Theorem 1.34 (Local Approximation; Cohen-Addad and Schwiegelshohn (2017)) *Let L be a ρ-locally optimal set of centers with respect to a metric d and S be an arbitrary set of k centers. Define sets A and B as above. Then, $\sum_{u\in X} d(u,l(u)) \le \sum_{u\in A} d(u,s(u)) + \gamma \sum_{u\in B} d(u,s(u))$, for some $\gamma = 3 + O(1/\rho)$.*

We refer to Cohen-Addad and Schwiegelshohn (2017) for the proof.

1.8 Bibliographical Notes

In the first paper on the subject, Bilu and Linial (2010) defined perturbation resilience, explained its importance, and gave an algorithm for $O(n)$-perturbation-resilient instances of Max Cut.[8] Bilu et al. (2013) gave an algorithm for $O(\sqrt{n})$-perturbation-resilient instances of Max Cut. Mihalák et al. (2011) designed a greedy algorithm for 1.8-perturbation-resilient instances of TSP. Then, Makarychev et al. (2014) designed a general framework for solving perturbation-resilient instances of combinatorial optimization problems (which we described in this chapter). The framework provides a general recipe for designing algorithms for perturbation-resilient and weakly perturbation-resilient instances, as well as proving that LP and SDP relaxations for perturbation-resilient instances are integral. This framework was used to design algorithms for several optimization problems, including algorithms for $O(\sqrt{\log n}\log\log n)$-perturbation-resilient instances of Max Cut (Makarychev et al., 2014), $(2-2/k)$-perturbation-resilient instances of Minimum Multiway Cut (Angelidakis et al., 2017), $(1+\varepsilon)$-perturbation-resilient instances of *planar* Maximum Independent Set, and $(k-1)$-perturbation-resilient instances of Maximum Independent Set in k-colorable graphs (Angelidakis et al., 2019). Makarychev and Makarychev (2019) introduced certified algorithms and showed how to design them using the framework we discussed above.

There are a number of negative results for perturbation-resilient instances. Most of the negative results show that there are no robust algorithms for γ-perturbation-resilient instances and no polynomial-time γ-certified algorithms. The assumption that the algorithms are certified or robust is crucial. In fact, proving that there is no polynomial-time algorithm, robust or otherwise, for perturbation-resilient instances is very challenging; to do so, one needs to get a reduction that maps known "hard instances" of some problem to perturbation-resilient instances of the problem at hand. Nevertheless, we know that there is no polynomial-time algorithm for ∞-perturbation-resilient instances of Max k-Cut (for every $k \geq 3$) if $\mathbf{RP} \neq \mathbf{NP}$ (Makarychev et al., 2014), and there is no polynomial-time algorithm for $o(\sqrt{n})$-perturbation-resilient instances of Maximum Independent Set if finding a planted clique in a random graph is hard (Angelidakis et al., 2019). Note that there are strong negative results for some very basic problems: there are no polynomial-time $n^{1-\delta}$-certified algorithms for Set Cover, Min Vertex Cover/Maximum Independent Set, Max 2-Horn SAT; also, there are no polynomial-time *robust* algorithms for $n^{1-\delta}$-perturbation-resilient instances of these problems. The result for Max 2-Horn SAT is particularly striking, since maximization and minimization variants of the problem admit a constant-factor approximation. The negative results suggest that one should study algorithms for special families of perturbation-resilient instances. This was done by Angelidakis et al. (2019), who gave an algorithm for *planar* $(1+\varepsilon)$-perturbation-resilient instances of Maximum Independent Set. This result is particularly interesting as it holds when the perturbation resilience parameter $\gamma = 1+\varepsilon$ is arbitrarily close to 1.

The study of perturbation-resilient instances of clustering problems was initiated

[8] Note that Bilu and Linial, as well as many other authors, refer to "perturbation resilience" as "stability".

by Awasthi et al. (2012) who gave an algorithm for finding the optimal clustering for 3-perturbation-resilient instances. Similarly to the algorithm we discussed in this chapter, their algorithm first runs single-linkage clustering and then recovers the optimal solution using dynamic programming. However, the dynamic program used in their algorithm is quite different from the one we presented here – it is simpler and faster but requires that the input be more perturbation-resilient. Balcan and Liang (2016) designed an algorithm for $(1+\sqrt{2})$-perturbation-resilient instances $(1+\sqrt{2}\approx 2.414)$. Balcan et al. (2015) gave algorithms for 2-perturbation-resilient instances of symmetric and asymmetric k-center and obtained matching hardness results. Angelidakis et al. (2017) offered the definition of *metric* γ-perturbation resilience and presented an algorithm for metric 2-perturbation-resilient instances of k-medians and k-means, which we discussed in this chapter (see Theorem 1.29). Ben-David and Reyzin (2014) showed that it is **NP**-hard to find the optimal clustering for instances of k-medians satisfying the $(2-\varepsilon)$-center proximity condition.

Cohen-Addad and Schwiegelshohn (2017) observed that the local search algorithm finds the optimal clustering for $(3+\varepsilon)$-perturbation-resilient instances. In their paper, they used a slightly different model from the one we discussed in this chapter. Theorem 1.32 is due to Balcan and White (2017). Very recently, Friggstad et al. (2019) designed an algorithm for solving $(1+\varepsilon)$-perturbation-resilient instances of Euclidean k-means and k-medians (where the points lie in a *fixed* dimensional Euclidean space). As was noted in (Makarychev and Makarychev, 2019), their algorithm is also $(1+\varepsilon)$-certified.

We also refer the reader to the survey by Makarychev and Makarychev (2016) for a more detailed though somewhat outdated overview of known results for perturbation-resilient instances. Finally, we note that this chapter is based in part on (Makarychev and Makarychev, 2019).

References

Agarwal, Amit, Charikar, Moses, Makarychev, Konstantin, and Makarychev, Yury. 2005. $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF Deletion, and directed cut problems. In: *Proceedings of the Symposium on Theory of Computing.*

Angelidakis, Haris, Makarychev, Konstantin, and Makarychev, Yury. 2017. Algorithms for stable and perturbation-resilient problems. In: *Proceedings of the Symposium on Theory of Computing.*

Angelidakis, Haris, Awasthi, Pranjal, Blum, Avrim, Chatziafratis, Vaggos, and Dan, Chen. 2019. Bilu-Linial stability, certified algorithms and the Independent Set problem. In: *Proceedings of the European Symposium on Algorithms.*

Arora, Sanjeev, Lee, James, and Naor, Assaf. 2008. Euclidean distortion and the sparsest cut. *Journal of the American Mathematical Society*, **21**(1), 1–21.

Arya, Vijay, Garg, Naveen, Khandekar, Rohit, Meyerson, Adam, Munagala, Kamesh, and Pandit, Vinayaka. 2004. Local search heuristics for k-median and facility location problems. *SIAM Journal on computing*, **33**(3), 544–562.

Awasthi, Pranjal, Blum, Avrim, and Sheffet, Or. 2012. Center-based clustering under perturbation stability. *Information Processing Letters*, **112**(1-2), 49–54.

Balcan, Maria Florina, and Liang, Yingyu. 2016. Clustering under perturbation resilience. *SIAM Journal on Computing*, **45**(1), 102–155.

Balcan, Maria-Florina, and White, Colin. 2017. Clustering under local stability: Bridging the gap between worst-case and beyond worst-case analysis. *arXiv preprint arXiv:1705.07157*.

Balcan, Maria-Florina, Haghtalab, Nika, and White, Colin. 2015. k-center Clustering under Perturbation Resilience. *arXiv preprint arXiv:1505.03924*.

Ben-David, Shalev, and Reyzin, Lev. 2014. Data stability in clustering: A closer look. *Theoretical Computer Science*, **558**, 51–61.

Bilu, Yonatan, and Linial, Nathan. 2010. Are Stable Instances Easy? In: *Innovations in Computer Science*.

Bilu, Yonatan, Daniely, Amit, Linial, Nati, and Saks, Michael. 2013. On the practically interesting instances of MAXCUT. In: *International Symposium on Theoretical Aspects of Computer Science*.

Călinescu, Gruia, Karloff, Howard, and Rabani, Yuval. 1998. An improved approximation algorithm for multiway cut. In: *Proceedings of the Symposium on Theory of Computing*.

Cohen-Addad, Vincent, and Schwiegelshohn, Chris. 2017. On the local structure of stable clustering instances. In: *Proceedings of the Symposium on Foundations of Computer Science*.

Friggstad, Zachary, Khodamoradi, Kamyar, and Salavatipour, Mohammad R. 2019. Exact Algorithms and Lower Bounds for Stable Instances of Euclidean K-means. In: *Proceedings of the Symposium on Discrete Algorithms*.

Goemans, Michel X, and Williamson, David P. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, **42**(6), 1115–1145.

Hochbaum, Dorit S. 1983. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, **6**(3), 243–254.

Kanungo, Tapas, Mount, David M, Netanyahu, Nathan S, Piatko, Christine D, Silverman, Ruth, and Wu, Angela Y. 2004. A local search approximation algorithm for k-means clustering. *Computational Geometry*, **28**(2-3), 89–112.

Karger, David R, Klein, Philip, Stein, Cliff, Thorup, Mikkel, and Young, Neal E. 2004. Rounding algorithms for a geometric embedding of minimum multiway cut. *Mathematics of Operations Research*, **29**(3), 436–461.

Lewin, Michael, Livnat, Dror, and Zwick, Uri. 2002. Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems. In: *In Proceedings of the Conference on Integer Programming and Combinatorial Optimization*.

Makarychev, Konstantin, and Makarychev, Yury. 2016. Bilu-Linial Stability. Chap. 13 of: Hazan, T., Papandreou, G., and Tarlow, D. (eds), *Perturbations, Optimization, and Statistics*. MIT Press.

Makarychev, Konstantin, and Makarychev, Yury. 2019. *Certified algorithms: Worst Case Analysis and Beyond*. manuscript.

Makarychev, Konstantin, Makarychev, Yury, and Vijayaraghavan, Aravindan. 2014. Bilu-Linial stable instances of Max Cut and Minimum Multiway Cut. In: *Proceedings of the Symposium on Discrete Algorithms*.

Mihalák, Matúš, Schöngens, Marcel, Šrámek, Rastislav, and Widmayer, Peter. 2011. On the complexity of the metric tsp under stability considerations. In: *SOFSEM 2011: Theory and Practice of Computer Science.*

Nemhauser, George L, and Trotter, Leslie Earl. 1975. Vertex packings: structural properties and algorithms. *Mathematical Programming*, **8**(1), 232–248.

Spielman, Daniel A, and Teng, Shang-Hua. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *JACM*, **51**(3).

Exercises

1.1 Consider an instance $\mathcal{I}$ of a maximization constraint satisfaction problem (such as Max 3SAT or Max 2CSP), in which the constraints are individually satisfiable. Assume that $\mathcal{I}$ has a unique optimal solution. Show that $\mathcal{I}$ is ∞-perturbation-resilient iff there is a solution that satisfies all the constraints.

1.2 Give examples of α-approximation algorithms for combinatorial optimization and clustering problems that are not α-certified.

1.3 Prove Theorem 1.12. (Hint: assign very large weights to $c \in H$.)

1.4 Prove Theorem 1.21.

1.5 Consider a maximization optimization problem P. Assume that every instance $\mathcal{I} = (\mathcal{S}, \mathcal{C}, w)$ has value at least $\alpha \cdot w(\mathcal{C})$ for some $\alpha \leq 1$ (for example, $\alpha = 1/2$ for Max Cut, $\alpha = 1/2^k$ for Boolean k-CSP). Prove that every γ-perturbation-resilient instance $\mathcal{I} = (\mathcal{S}, \mathcal{C}, w)$ has a solution of value at least $\frac{\gamma}{\gamma+1/\alpha} w(\mathcal{C})$.

1.6 Consider a maximization problem P. Assume that it does not admit an α-approximation ($\alpha > 1$); more precisely, there is a Karp-reduction from 3-SAT to P that maps a yes-instance to an instance whose optimal solution has value at least $c \cdot w(\mathcal{C})$, and a no-instance to an instance whose optimal solution has value less than $\frac{c \cdot w(\mathcal{C})}{\alpha}$. Prove that then there is no polynomial-time algorithm for deciding whether an instance $\mathcal{I}$ of P is α-perturbation-resilient (if $\mathbf{NP} \neq \mathbf{coNP}$).

1.7 Show that there are no instances (X, d) of k-clustering problems with $|X| > k$ and $\gamma \geq 2$ satisfying the following strong version of the γ-perturbation resilience condition: For every metric γ-perturbation d' of d there is only one set of optimal centers, and this set of centers is the same as for metric (X, d).

1.8 Consider a γ-perturbation-resilient instance (X, d) of the k-clustering problem with the ℓ_p objective. Show that a γ-certified solution to (X, d) is an optimal solution for this problem.

1.9 Show that a γ-certified solution to an arbitrary instance of the k-clustering problem with the ℓ_p objective is a γ^p approximation to the optimal solution.

1.10 Give an example of an instance (X, d) of k-medians which is not γ-perturbation-resilient, but whose unique optimal solution satisfies the γ-center proximity property.

1.11 Give an example of a 100-perturbation-resilient instance for which the single-linkage clustering is suboptimal.