CS264: Beyond Worst-Case Analysis
Lecture #8: Exact Recovery in Stable Cut Instances[*]

Tim Roughgarden[†]

October 15, 2014

1 Exact Recovery

This lecture makes explicit a theme introduced surreptitiously last lecture: *exact recovery.* The genre of question in this sequence of lectures is: for an algorithm A of interest, for which inputs does A solve the problem exactly? For example, last lecture, we proved that the single-link++ algorithm recovers the optimal solution in every 3-stable k-median instance. (See Homework #4 for an improvement.)[1]

We generally think of the algorithm A above as either: (i) an already-implemented algorithm that is available to us, such as a linear programming solver; or (ii) a relatively simple algorithm that we have a good understanding of and could code up easily. The reason is that, in practice, designing a new algorithm from scratch is often a last resort. It is usually far preferable if something in the current algorithmic toolbox is good enough, at least for the problem instances relevant to the application. The exact recovery results in this sequence of lectures develop theoretical guarantees that correspond well to this style of thinking.[2]

This lecture covers another case study on exact recovery, for a graph cut problem. We'll retain from last lecture the definition of γ-stable instances and the flavor of the problems studied, as cut problems are a type of clustering problem. What will be different is the algorithms studied — linear programming instead of more elementary algorithms.

The broader theme, to be elaborated on in future lectures, is to explain when NP-hard problems can be solved exactly via linear programming. (Since linear programs can be solved in polynomial time, we do not expect this to be true for worst-case instances.) Thus, even

[†]Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: `tim@cs.stanford.edu`.

[1]*Approximate* notions of recovery are also interesting, such as the guarantees we covered in Lecture #6. For simplicity we'll stick with exact recovery for these lectures.

[2]This contrasts with a majority (but certainly not all) theoretical computer science research on algorithms, where the primary goal is usually to come up with new algorithms that are in some sense "better" than the state-of-the-art.

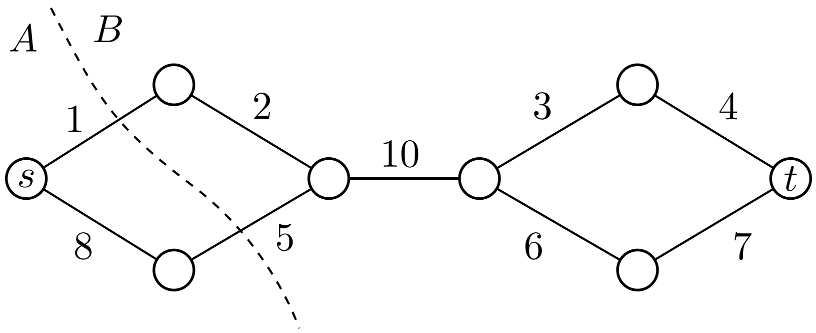

Figure 1: The minimum s-t cut of a graph.

more so than in the last lecture, we'll use the stability assumption only in the analysis of an algorithm (linear programming), and not as a guide for designing new algorithms. This lecture serves as a segue into other exact recovery results, like those in compressive sensing, where linear programming relaxations play a starring role.

2 The Minimum s-t Cut Problem

We begin with a warm-up, the minimum s-t cut problem. Everything in this section has been known for a half-century, but we'll phrase it in a way that generalizes surprisingly effortlessly to NP-hard generalizations of the problem (under stronger assumptions).

Recall the input to the problem is an undirected graph $G = (V, E)$ with a source vertex $s \in V$ and a sink vertex $t \in V$. Every edge $e \in E$ has a positive *cost* $c_e > 0$. The desired output is an s-t cut — a partition of the vertex set V into A and B with $s \in A$ and $t \in B$ — that minimizes the sum of the costs of the cut edges. (An edge is cut if it has one endpoint in each of A and B.) See Figure 1.

As you may know, the minimum s-t cut problem can be solved in polynomial time, for example using a maximum flow algorithm. The point of this section is to show that, alternatively, one can just solve solve a suitable linear program and the minimum s-t cut pops out as the optimal solution. For simplicity, we assume throughout this section that the optimal solution is unique. When we study harder problems later, we'll upgrade this uniqueness assumption to a suitable γ-stability assumption.

Recall that a linear program has a set of real-valued decision variables, and the goal is to optimize an objective function that is linear in the decision variables, subject to constraints that are also linear. Geometrically, the constraints are an intersection of halfspaces, the objective function gives a direction, and the goal is to find the feasible point that is furthest in the desired direction (Figure 2).

How do we encode the minimum s-t cut problem as a linear program? There are two sets of decision variables. First, a nonnegative variable $x_e \geq 0$ for each edge $e \in E$, with the semantics that $x_e = 1$ if e gets cut and $x_e = 0$ otherwise. Second, there is a nonnegative

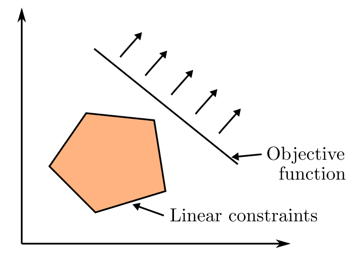

Figure 2: Cartoon representation of a linear program.

decision variable $d_v \geq 0$ for each vertex $v \in V$; the intention is that $d_v = 0$ if v is in the source-side A of the cut, and $d_v = 1$ if $v \in B$. Given these semantics, two obvious constraints are $d_s = 0$ and $d_t = 1$. The more interesting constraints are meant to enforce the property that, whenever the endpoints u, v of an edge e are on different sides of the s-t cut, edge e is cut. Precisely, these constraints are:

$$\begin{aligned} x_e &\geq d_u - d_v \\ x_e &\geq d_v - d_u \end{aligned}$$

for every edge $e = (u, v) \in E$. These constraints are equivalent to the single constraint $x_e \geq |d_u - d_v|$; we have written them so that it is obvious that they are linear constraints. Given the intended semantics for the x-variables, it is clear that the (linear) objective function is

$$\min \sum_{e \in E} c_e x_e.$$

We denote this linear program by (LP1).

It is clear that every s-t cut (A, B) induces a feasible 0-1 solution to (LP1): set $d_v = 0$ for $v \in A$, $d_v = 1$ for $v \in T$, and $x_e = 1$ for the cut edges and $x_e = 0$ for the rest. There are, however, feasible solutions to the linear program that do not correspond to s-t cuts. This is intuitively obvious from a glance at Figure 2 — the s-t cuts correspond to a discrete and finite set of points, whereas the feasible region of a linear program is a convex set. More concretely, in the graph of Figure 1, one fractional solution is to set $x_e = \frac{1}{5}$ for every edge, with d_v-values of vertices increasing by $\frac{1}{5}$ with each hop away from s and toward t.

An important fact, that you should definitely know, is that linear programs can be solved efficiently, both in theory and in practice.

Fact 2.1 *There is a polynomial-time algorithm for linear programming.*

Whether proving theorems or solving real-world problems, most users of linear programming algorithms treat them as "black boxes" and don't understand how or why they work. We will do the same in the course. If you want to see what's "under the hood," check out e.g. [5].

In this section, we prove the following.

Proposition 2.2 *For every minimum s-t cut problem with a unique optimal solution, the optimal solution to the linear program (LP1) is the (straightforward encoding of) the minimum s-t cut.*

The point of Proposition 2.2 is that, while the feasible region of the linear program includes plenty of points that don't correspond to s-t cuts, we don't need to worry about them — as long as we solve the linear program to optimality, we'll get back an integer solution that corresponds to the minimum cut.[3]

To prove the proposition, let C^* denote the minimum s-t cut, and let F^* denote the edges cut by it. For another s-t cut C, cutting the edges F, we let $\Delta(C)$ denote the extent to which the total cost of C exceeds that of C^*:

$$\Delta(C) = \sum_{e \in F \setminus F^*} c_e - \sum_{e \in F^* \setminus F} c_e.$$

Since C^* is the unique minimum s-t cut,

$$\Delta(C) \geq 0, \tag{1}$$

for every s-t cut C, with equality holding if and only if $C = C^*$.

Similarly, for the optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{d}})$ to (LP1), we denote by $\Delta(\hat{\mathbf{x}})$ the extent to which the objective function value of the integer solution corresponding to C^* exceeds that of $\hat{\mathbf{x}}$:

$$\Delta(\hat{\mathbf{x}}) = \sum_{e \notin F^*} c_e \hat{x}_e - \sum_{e \in F^*} c_e (1 - \hat{x}_e).$$

Since C^* corresponds to a feasible solution to (LP1), and $\hat{\mathbf{x}}$ corresponds to the optimal solution,

$$\Delta(\hat{\mathbf{x}}) \leq 0. \tag{2}$$

The key to proving Proposition 2.2 is the following lemma.

Lemma 2.3 *There is a randomized algorithm that outputs a (random) s-t cut C such that, for every edge $e \in E$,*

$$\mathbf{Pr}[e \text{ cut by } C] = \hat{x}_e \tag{3}$$

and hence

$$\mathbf{Pr}[e \text{ not cut by } C] = 1 - \hat{x}_e. \tag{4}$$

Before proving the lemma, we explain why it implies Proposition 2.2. The intuition is that since (i) the optimal fractional solution $\hat{\mathbf{x}}$ can only be better than the optimal integral solution C^*; and (ii) Lemma 2.3 gives a distribution over s-t cuts that is as good, on average, as C^*; the distribution must be a point mass on the cut C^*, which implies that $\hat{\mathbf{x}}$ is a 0-1 solution.

[3]This is true even when there is not a unique minimum s-t cut. The reasons, which we won't prove here, are: (i) all of the extreme points (a.k.a. vertices) of the feasible region are integral and correspond to s-t cuts; and (ii) linear programming algorithms always return an extreme point of the feasible region.

Formally, let C denote the random s-t cut generated by the randomized algorithm in Lemma 2.3, and consider the expected extent to which its cost exceeds that of C^*:

$$\begin{aligned}
\mathbf{E}[\Delta(C)] &= \sum_{e \notin F^*} c_e \cdot \mathbf{Pr}[C \text{ cuts } e] - \sum_{e \in F^*} c_e \cdot \mathbf{Pr}[C \text{ doesn't cut } e] && (5)\\
&= \sum_{e \notin F^*} c_e \hat{x}_e - \sum_{e \in F^*} c_e (1 - \hat{x}_e) && (6)\\
&= \Delta(\hat{\mathbf{x}})\\
&\leq 0, && (7)
\end{aligned}$$

where equation (5) follows from linearity of expectation, equation (6) from (3) and (4) in Lemma 2.3, and (7) follows from (2). Since $\Delta(C)$ is always nonnegative and is strictly positive unless $C = C^*$ (recall (1)), the derivation (5)–(7) implies that the randomized algorithm of Lemma 2.3 outputs the cut C^* with probability 1. Inspection of (3) and (4) shows that this occurs only if $\hat{x}_e \in \{0, 1\}$ for every $e \in E$. This completes the proof of Proposition 2.2.

Proof of Lemma 2.3: Let $(\hat{\mathbf{x}}, \hat{\mathbf{d}})$ be the optimal solution to (LP1). Intuitively, the randomized algorithm blows up a balloon around the source s up to a random size, with everything inside the balloon comprising the source-side of the cut. Formally:

1. Pick a "radius" $r \in (0, 1)$ uniformly at random.
2. Define $A = \{v \in V \,:\, \hat{d}_v \leq r\}$, and $B = V \setminus A$.

Since $\hat{d}_s = 0$ and $\hat{d}_t = 1$, the algorithm's output is an s-t cut with probability 1. We leave the verification that it satisfies (3) and (4) to Homework #4. ■

3 Stable Instances of the Minimum Multiway Cut Problem

Proposition 2.2 shows that in every minimum s-t cut problem with a unique optimal solution, the optimal solution can be recovered exactly in polynomial time by solving a linear programming relaxation. This section takes this result to the next level, by considering an NP-hard generalization of the minimum s-t cut problem, replacing the uniqueness assumption with a stronger stability assumption, and again showing how to recover the optimal (integral) solution using linear programming.

In the *minimum multiway cut problem*, the input is an undirected graph $G = (V, E)$ with positive edge costs c_e, and a set $t_1, t_2, \ldots, t_k \in V$ of *terminals*. A *multiway cut* is a partition of the vertex set V into k groups $S_1, \ldots, S_k$, with $t_i \in S_i$ for $i = 1, 2, \ldots, k$. In addition to being a natural generalization of the fundamental s-t minimum cut problem (which is equivalent to the $k = 2$ case), the problem also shows up in applications, such as image segmentation.[4] The problem is NP-hard for every $k \geq 3$ [2]. See Figure 3.

[4]In the most straightforward formulation, vertices correspond to pixels, edges correspond to neighboring

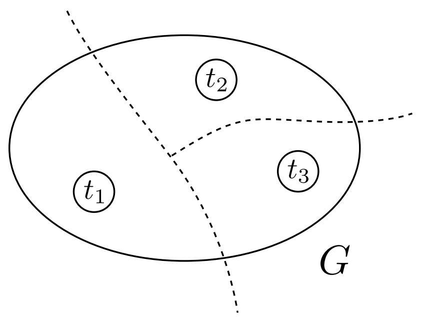

Figure 3: A multiway cut of a graph (with $k = 3$).

Since the problem is NP-hard, we do not expect to recover an optimal solution using linear programming in worst-case instances. But we *can* do this under a stability assumption, analogous to the one introduced last lecture for k-median instances.

Definition 3.1 For a parameter $\gamma \geq 1$, an instance of multiway cut is γ-*stable* if the optimal solution remains the same for every γ-perturbation, where each original edge cost c_e is replaced by $\sigma_e c_e$ for some $\sigma_e \in [1, \gamma]$.

The main result of this lecture is the following recent result; we'll define the linear programming relaxation (LP2) shortly.

Theorem 3.2 ([4]) *For every γ-stable multiway cut instance with $\gamma > 4$, the optimal solution to the linear program (LP2) is the (straightforward encoding of) the optimal multiway cut.*

This linear program (LP2) is a nice extension of (LP1), which can be viewed as the $k = 2$ special case. We again have a set of decision variables $x_e \geq 0$ for each $e \in E$, meant to indicate which edges get cut. We now have multiple variables $d_v^1, \ldots, d_v^k \geq 0$ per vertex, with the idea that if v is assigned to the ith group, then $d_v^i = 1$ and $d_v^j = 0$ for every $j \neq i$. The first set of constraints just asserts that each terminal is fully assigned to its own group:

$$d_{t_i}^i = 1$$

for each $i = 1, 2, \ldots, k$. For other vertices, we insist that they are fully assigned, perhaps fractionally, to the groups:

$$\sum_{i=1}^{k} d_v^i = 1$$

pixels, edge costs correspond to a prior guess about how likely two neighboring pixels are to be in the same region (e.g., based on how similar their colors are), and groups of the multiway cut correspond to the different regions of the image [1].

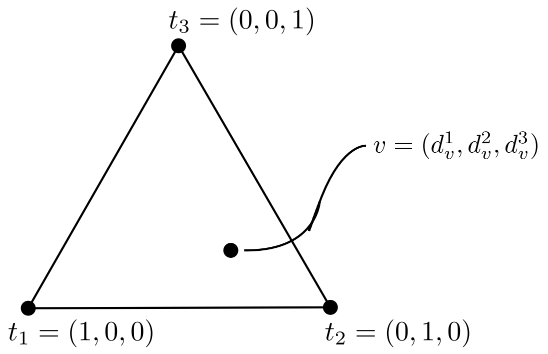

Figure 4: Pictorial representation of (LP2): the simplex when $k = 3$, drawn in two dimensions.

for every $v \in V$. Next, we give constraints stating that edges with endpoints in different groups must be cut. First, we have

$$\begin{aligned} y_e^i &\geq d_u^i - d_v^i \\ y_e^i &\geq d_v^i - d_u^i \end{aligned}$$

for every edge $e = (u, v) \in E$ and $i = 1, 2, \ldots, k$, where the y_v^i's are auxiliary variables. These constraints assert that the difference in the extent to which the edge's endpoints are assigned to group i should contribute to the extent to which e is cut. Finally, for every $e \in E$, we have the constraint

$$x_e = \frac{1}{2} \sum_{i=1}^{k} y_e^i, \tag{8}$$

which aggregates the group assignment differences across the coordinates. Note that if the endpoints of an edge e are assigned integrally to different groups (i and j, say), then $y_e^i = y_e^j = 1$; this motivates the "$\frac{1}{2}$" in (8).[5] Also note that every multiway cut naturally induces a feasible 0-1 solution to (LP2), according to the intended semantics of the decision variables. As another example, if $k = 3$, and the d-values of u and v are $(\frac{1}{2}, \frac{1}{2}, 0)$ and $(0, \frac{1}{2}, \frac{1}{2})$, then the x-value of edge $e = (u, v)$ must be at least $\frac{1}{2}$. Pictorially, we can think of (LP2) as assigning each vertex v a point $(d_v^1, \ldots, d_v^k)$ in the k-simplex (Figure 4), with corners of the simplex corresponding to the groups. Constraint (8) states that edge $e = (u, v)$ is cut at least in proportion to the "distance" between the points of the simplex to which u and v are assigned.

We now turn to Theorem 3.2. The proof follows the same template as that of Proposition 2.2. Let C^* be the unique (and γ-stable for $\gamma > 4$) multiway cut, cutting the edges F^*.

[5]If we were only concerned with integral solutions, the constraint $x_e = \max_{i=1}^{k} y_e^i$ would already force edge e to be cut whenever its endpoints are assigned to different groups. When the variables can be fractional, however, the constraint (8) rules out many more fractional solutions, thereby increasing the utility of (LP2) as an approximation to the problem that we actually care about, the minimum multiway cut problem.

For another multiway cut C, cutting edges F, define

$$\Delta_4(C) = \sum_{e \in F \setminus F^*} c_e - 4 \sum_{e \in F^* \setminus F} c_e. \tag{9}$$

Clearly $\Delta_4(C^*) = 0$. If $C \neq C^*$ then, by γ-stability, $\Delta_4(C) > 0$. (This follows by considering the perturbation that blows up costs of edges in F^* as much as possible, and leaving other edges the same; see Homework #4.) For the optimal solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{d}})$ to (LP2), as before we define

$$\Delta(\hat{\mathbf{x}}) = \sum_{e \notin F^*} c_e \hat{x}_e - \sum_{e \in F^*} c_e (1 - \hat{x}_e).$$

As before, since C^* induces a feasible solution to (LP2) and $\hat{\mathbf{x}}$ is optimal,

$$\Delta(\hat{\mathbf{x}}) \leq 0. \tag{10}$$

We next give the analog of Lemma 2.3. We can't expect an equally strong lemma for multiway cuts with $k \geq 3$, since this would give a polynomial-time exact algorithm for an NP-hard problem. Instead, we content ourselves with a version in which the identities (3) and (4) hold approximately.

Lemma 3.3 *There is a randomized algorithm that outputs a (random) multiway cut C such that, for every edge $e \in E$,*

$$\mathbf{Pr}[e \textit{ cut by } C] \leq 2\hat{x}_e \tag{11}$$

and

$$\mathbf{Pr}[e \textit{ not cut by } C] \geq \frac{1 - \hat{x}_e}{2}. \tag{12}$$

Note that (12) does not follow from (11); the implied lower bound of $1 - 2\hat{x}_e$ is not good enough for our purposes.

The proof that Lemma 3.3 implies Theorem 3.2 follows the proof of Proposition 2.2. We lose two factors of 2 in the argument from (11) and (12), but these get absorbed by the assumption of γ-stability for $\gamma > 4$. Formally, let C denote the random multiway cut generated by the randomized algorithm in Lemma 3.3. We have

$$\begin{aligned}
\mathbf{E}[\Delta_4(C)] &= \sum_{e \notin F^*} c_e \cdot \mathbf{Pr}[C \text{ cuts } e] - 4 \sum_{e \in F^*} c_e \cdot \mathbf{Pr}[C \text{ doesn't cut } e] && (13)\\
&\leq \sum_{e \notin F^*} c_e (2\hat{x}_e) - 4 \sum_{e \in F^*} c_e \tfrac{1}{2}(1 - \hat{x}_e) && (14)\\
&= 2\Delta(\hat{\mathbf{x}})\\
&\leq 0, && (15)
\end{aligned}$$

where equation (13) follows from linearity of expectation, inequality (14) from (11) and (12) in Lemma 3.3, and (15) follows from (10). Since $\Delta_4(C)$ is always nonnegative and is strictly positive unless $C = C^*$, the derivation (13)–(15) implies that the randomized algorithm of Lemma 3.3 outputs the cut C^* with probability 1. As we'll see when we describe the randomized algorithm of Lemma 3.3 below, this occurs only if $\hat{x}_e \in \{0,1\}$ for every $e \in E$. Modulo this point, this completes the proof of Theorem 3.2.

Proof of Lemma 3.3: Let $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{d}})$ be the optimal solution to (LP2). The randomized algorithm (originally from [3]) is simple, though unlike the one in the proof of Lemma 2.3, it's probably not the first one you'd think of:

1. Initially, all vertices are unassigned.
2. While there is at least one unassigned vertex:
   (a) Pick a threshold $r \in (0,1)$ uniformly at random.
   (b) Pick a group $i \in \{1, 2, \ldots, k\}$ uniformly at random.
   (c) For every unassigned vertex v, if $\hat{d}_v^i \geq t$, assign vertex v to group i.

The algorithm eventually terminates with a multiway cut — each vertex is assigned to exactly one group, and since $\hat{d}_{t_i}^i = 1$ and $\hat{d}_{t_i}^j = 0$ for $j \neq i$ for every i, every terminal is assigned to the correct group. We leave the verification of inequalities (11) and (12) to Homework #4.[6]
■

Observe that, if d_v^j is strictly between 0 and 1 for any vertex v and coordinate j, then $d_v^\ell \in (0,1)$ for some other coordinate ℓ as well (since $\sum_{i=1}^k d_v^i = 1$). In this case, the randomized algorithm of Lemma 3.3 will sometimes assign v to group j, and sometimes to group ℓ. We conclude that the randomized algorithm always returns the same multiway cut only if the optimal fractional solution $(\hat{\mathbf{d}}, \hat{\mathbf{x}}, \hat{\mathbf{y}})$ is integral. This completes the proof of Theorem 3.2, that in γ-stable multiway cut instances with $\gamma > 4$, a natural linear programming relaxation (LP2) is guaranteed to be exact.

An interesting open research question is to prove better upper bounds, or any lower bounds, on the minimum value of γ such that the optimal solution of γ-stable multiway cut instances can be recovered in polynomial time (by solving (LP2) or by other means).

References

[1] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2011.

[6]Here's an example to get a feel for the algorithm's behavior. Suppose the endpoints u, v of edge e have $\hat{d}$-values $(\frac{1}{2}, \frac{1}{2}, 0)$ and $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. If i is randomly chosen to be 3, then u is definitely not assigned to any group in this iteration, and v is assigned in this iteration to group 3 with probability $\frac{1}{3}$. In the latter case, edge e will definitely be cut at the end of the day. If i is chosen to be 1, then three things could happen: both u and v could be assigned to group 1 (if $r \leq \frac{1}{3}$), in which case e is definitely not cut; neither could be assigned (if $r > \frac{1}{2}$), leaving the edge's fate to a future iteration; or only u could be assigned to group 1 (if $r \in (\frac{1}{3}, \frac{1}{2}]$), in which case e is likely to eventually get cut (although there is still a chance v could be assigned to group 1 in a future iteration).

[2] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[3] J. M. Kleinberg and É.Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. *Journal of the ACM*, 49(5):616–639, 2002.

[4] K. Makarychev, Y. Makarychev, and A. Vijayaraghavan. Bilu-Linial stable instances of max cut and minimum multiway cut. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 890–906, 2014.

[5] J. Matousek and B. Gärtner. *Understanding and Using Linear Programming.* Springer, 2006.