Enumerating Parametric Global Minimum Cuts by Random Interleaving

David R. Karger*
MIT Computer Science and AI Laboratory
Cambridge, MA 02138
karger@mit.edu
http://people.csail.mit.edu/karger

ABSTRACT

Recently, Aissi et al. gave new counting and algorithmic bounds for *parametric* minimum cuts in a graph, where each edge cost is a linear combination of multiple cost criteria and different cuts become minimum as the coefficients of the linear combination are varied. In this article, we derive better bounds using a mathematically simpler argument. We provide faster algorithms for enumerating these cuts. We give a lower bound showing our upper bounds have roughly the right form. Our results also immediately generalize to parametric versions of other problems solved by the *Contraction Algorithm*, including approximate min-cuts, multi-way cuts, and a matroid optimization problem. We also give a first generalization to *nonlinear* parametric minimum cuts.

Categories and Subject Descriptors

F2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on Discrete Structures*; G2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial Algorithms, Counting Problems*; G2.2 [**Discrete Mathematics**]: Graph Theory—*Graph Algorithms*

Keywords

Minimum cuts, Randomization, Parametric optimization, Enumeration

1. INTRODUCTION

Recently, Aissi et al. [AMMQ15] gave new bounds on the number of *parametric* minimum cuts in graphs and hypergraphs. Their problem formulation considers a set of k distinct edge cost "criteria" $c_i(e)$ and defines an overall combined cost $c(e) = \sum \mu_i c_i(e)$ using linear coefficients μ_i. As the μ_i vary so does each $c(e)$, which causes the min-cut of the graph to vary. It is natural to ask how many different cuts can become min-cuts and how to enumerate all of them quickly. There is an extensive literature on and interest in such parametric optimization.

In a graph with negative edge costs, even the standard single-parameter minimum cut problem is NP-hard. And the empty graph (equivalently, all zero cost edges) has all exponentially many cuts minimum even without varying parameters. But it is well known [DKL76, KS96, Kar00] that any connected (by its positive cost edges) n-vertex graph has at most $\binom{n}{2}$ min-cuts, with this bound being achieved by the cycle on n vertices. Thus work focuses on graphs where all edge costs are non-negative (although the individual criteria may be negative) and where the graph is connected by its positive cost edges. Mulmuley [Mul99] gave an $O(n^{19})$ bound on the number of 2-parameter min-cuts in this case.

Aissi et al. show that in a k-parameter linear combination, the number of cuts that become minimum at some point in the parameter space is $O(m^k n^2 \log^{k-1} n)$. In the case of hypergraphs there is an additional exponential dependence on the hypergraph rank. They also derive algorithms with $n^{O(k^2)}$ running times for enumerating all such cuts. For the special case of 2 parameters they give a better bound of $O(n^3 \log n)$ cuts that can be enumerated in $\tilde{O}(m^2 n^4)$ time.

1.1 Results

In this article, we give a different analysis of parametric min-cuts that uses significantly less math. We require stronger conditions than Aissi et al. to apply our results, but the results are significantly stronger for those conditions and generalize more broadly in various directions.

In particular, we require that *each* of the k criterion functions $c_i(e)$, as well as the μ_i, be nonnegative, instead of just requiring that the overall linear combination be nonnegative. With this stronger condition, we improve the bound on the number of k-parameter minimum cuts from $\tilde{O}(m^k n^2)$ to $O(n^{1+k})$. We give an algorithm for enumerating all of them in $\tilde{O}(mn^{1+k})$ time, improving the Aissi et al. bound of $n^{O(k^2)}$. We also give better bounds of $O(n^3\sqrt{m})$ for enumerating 2-parameter min-cuts and $\tilde{O}(n^4)$ for enumerating 3-parameter min-cuts. We give a related lower bound, showing a k-parameter graph with $\Omega(n^{k/2})$ min-cuts over the parameter space.

Our techniques generalize to prove similar bounds on the number of *hypergraph* min-cuts in the parametric setting (again, improving on bounds by Aissi et al. for this problem) as well as the number of *approximately minimum* parametric graph cuts and hypergraph cuts (for which there were no previous results) and a more general "matroid min-cut" opti-

*Supported by NSF Award number 1117381

STOC'16, June 19–21, 2016, Cambridge, MA, USA
ACM. 978-1-4503-4132-5/16/06...$15.00
http://dx.doi.org/10.1145/2897518.2897578

mization problem. Our results are modular: they essentially state that in all these problems, the number of k-parameter solutions is $O(n^{k-1})$ times the number of standard solutions, and that a variant of the Contraction Algorithm [KS96] finds them all. We also provide efficient algorithms for enumerating the cuts in question.

We also generalize to the first result on *nonlinear* parametric min-cut problems. We show that when edge costs have the form $\sum_{i=1}^{k} c_i(e)\mu^i$ for positive $c_i(e)$—that is, a *polynomial* combination of k criteria via a single parameter μ—the number of min-cuts is $O(kn^3)$. It seems it is not linearity but limited degrees of freedom that constrains the number of min-cuts. We also give an $O(n^3\sqrt{m})$-time algorithm for enumerating these cuts—i.e., spending roughly $\sqrt{m}$ time per potential cut. These results generalize to a broad class of parametric functions that don't repeatedly cross each other.

1.2 Method

We prove our result by applying the Contraction Algorithm [Kar93], which finds min-cuts by contracting edges in random order. The new trick we apply here is to describe that random choice process over a parametric cost function as a series of decisions that we can make in two different orders—much like exchanging the order of a summation. Intuitively, we argue that running the Contraction Algorithm on a parametric combination of weight functions is equivalent to running a *parametric combination of Contraction Algorithms*, and that the increase in the number of parametric min-cuts is determined entirely by the way these different Contraction Algorithms can interleave their executions, or more precisely by the relative "rates" at which these Contraction Algorithms execute.

2. THE ANALYSIS

For brevity, we call a particular cut *minimish* if there is *some* setting of the parameters for which it is a min-cut. We wish to enumerate the minimish cuts. We generalize our original use of the Contraction Algorithm [Kar93] to bound the number of (standard) minimum cuts. Our core result is the following:

THEOREM 2.1. *The number of cuts that become minimum over all nonnegative combinations $\sum \mu_i c_i(e)$ of k parameters is $O(n^{k+1})$.*

2.1 The Contraction Algorithm

We define the Contraction Algorithm as follows:

- Generate a random *weighted permutation* of the edges of G by repeatedly choosing an unchosen edge with probability proportional to its weight until all edges have been chosen
- Contract edges in permutation order until only 2 vertices remain, defining the output cut

Usually, we describe the Contraction Algorithm [KS96] as choosing and contracting non-loop edges. In the above variation we don't notice self loops until after we build the permutation and begin contracting. But these loops have no impact on the output cut, so the variant produces the same output as the standard Contraction Algorithm. It is known [KS96] that this process produces any particular min-cut with probability at least $\binom{n}{2}^{-1}$.

2.2 Proof Outline

We begin our analysis with the following slightly modified description of the Contraction Algorithm's application to (standard) min-cuts:

- The Contraction Algorithm outputs one random cut
- It outputs any given min-cut with probability at least $\binom{n}{2}^{-1}$ (assuming the graph is connected).
- Thus if there are K min-cuts, the expected number output is at least $K\binom{n}{2}^{-1}$.
- But only 1 cut (minimum or not) is ever output, so $K\binom{n}{2}^{-1} \le 1$, which shows that $K \le \binom{n}{2}$.

For the parametric problem, we make a minor change:

- We modify the Contraction Algorithm to output at set of C cuts.
- We show that every minimish cut is among those output with probability at least $\binom{n}{2}^{-1}$.
- So if there are K minimish cuts, the expected number output is at least $K\binom{n}{2}^{-1}$.
- But only C cuts (minimish or not) are ever output, so $K\binom{n}{2}^{-1} \le C$, which shows that $K \le C\binom{n}{2}$.

Finally we bound C by n^{k-1}, which concludes our analysis.

2.3 A Special Case

To convey the main idea, we'll first consider a special case of the problem. Take two edge sets E_1 and E_2 on the same vertex set G, with m_1 and m_2 edges respectively. Each E_i spans G. Parallel edges may exist. Consider the parametric problem where we assign weight μ_1 to every edge in E_1 and weight μ_2 to every edge in E_2, and consider the evolving min-cuts as we vary μ_1 and μ_2.

For this parametric problem, once we have fixed the μ_i to get a single cost function, we can apply the Contraction Algorithm, contracting according to a permutation weighted by the scaled costs $\mu_i c_i(e)$. In our special case graph, each edge of the permutation comes from either E_1 or E_2. In other words, the permutation can be described by specifying (i) its subsequence π_1 of edges from E_1, (ii) its subsequence π_2 of edges from E_2, and (iii) an *interleaving* S indicating, at each step, whether we take our next edge from π_1 or π_2.

Critically, the sequences π_1 and π_2 are *uniformly random, independent* sequences of their respective edge sets. This is because each step in building the permutation can be described as first deciding randomly from which E_i to choose an edge and then deciding randomly which edge to take from the selected E_i. Since all edges of E_i have the same weight μ_i, each edge choice from E_i is uniformly random.

It follows that we can describe the construction of our permutation in a different way: first, we generate uniformly random permutations of E_1 and E_2. Then, we randomly interleave these two permutations via some random process that depends on the μ_i and π_i.

So far we have only changed the description of the algorithm, not its operation. Thus, given the specific μ_i the procedure just described outputs any particular min-cut (under cost $\mu_1 c_1 + \mu_2 c_2$) with probability at least $\binom{n}{2}^{-1}$. Now we make a change: instead of using this permutation to output *one* cut, we consider *all possible interleavings* of the (uniformly random) π_i and output all the cuts produced by contracting along these different interleavings.

Our original procedure that used a random interleaving (from the proper distribution) output our particular minimish cut with probability $\binom{n}{2}^{-1}$. Our revised procedure, which tries all interleavings, will therefore still output this cut with at least the same probability.

Now note that this argument applies to *each* minimish cut. That is, the all-interleavings algorithm outputs *each* minimish cut with probability $\binom{n}{2}^{-1}$. But this modified algorithm no longer depends on the μ_i. Those μ_i affected *which* interleaving of the uniformly random π_i was generated in the original Contraction Algorithm, but our modified algorithm considers all interleavings.

In summary, we designed an algorithm that outputs some number C of cuts such that every minimish cut is output with probability at least $\binom{n}{2}^{-1}$. As outlined in Section 2.2, it follows that if there are K minimish cuts, then the expected number of minimish cuts output by our algorithm is at least $K\binom{n}{2}^{-1} \leq C$, which implies that $K \leq C\binom{n}{2}$.

2.4 Bounding the Number of Distinguishable Interleavings

We now bound the number of cuts C output as we consider all interleavings. A priori, there are $\binom{m_1+m_2}{m_1}$ distinct interleavings, but we will see that most of them produce the same cuts.

In particular, once we fix the π_i, the exact interleaving order doesn't matter. The Contraction Algorithm terminates when certain interleaved prefixes of π_1 and π_2 have been contracted to reduce G to 2 vertices. The exact order in which those contractions are done doesn't matter.

It follows that without loss of generality we can assume the interleaving places all edges contracted from π_1 before all those from π_2. This means that we need only specify the number n_1 of (non-self-loop) contractions that occur from the prefix of π_1, since then there must be exactly $n - n_1 - 2$ contractions from π_2 to reduce G to 2 vertices and output a cut.

We have shown that every output cut corresponds to a value of n_1 chosen from 0 to $n-2$, so there are at most $n-1$ distinct output cuts. Incorporating this refinement into our argument, we find the number of minimish cuts is at most $n\binom{n}{2}$.

Our analysis assumes that each interleaving yields only one cut, which is true if the positive-cost edges span G. This certainly holds if each of the E_i span G as we assumed at the beginning; alternatively, it is sufficient for the *union* of the E_i to span G if we require to all $\mu_i > 0$ instead of $\mu_i \geq 0$. There may be other parameter values $\mu_i < 0$ for which the *overall* cost function is nonnegative and the positive edges span G; however, our proof does not hold in this case because we treat the μ_i as probabilities, which only makes sense if the μ_i are nonnegative. Some of the results of Aissi et al. apply to the more general parameters so long as the *combined* cast is positive.

2.5 The General Case

The basic argument above generalizes, unchanged, to arbitrary graphs with two-parameter linear weight functions. We considered a parametric combination of "unweighted" graphs (all edge weights 1). But we can instead consider a weight function $\mu_1 c_1(e) + \mu_2 c_2(e)$. For (nonnegative) integer valued $c_i(e)$ we can replace the edge of weight $c_i(e)$ with $c_i(e)$ parallel edges of weight 1 and reduce to our special case. For (nonnegative) rational $c_i(e)$ we can scale up by the least common denominator to reduce to the integer-weighted case. And nonnegative reals $c_i(e)$ are a limit of rationals.

The resulting graphs have unbounded numbers of edges but this doesn't matter for the counting arguments. In fact, given the equivalence we can simulate the algorithms directly on the weighted edges: we generate individual *weighted permutations* π_i by sampling edges e with probabilities proportional to $c_i(e)$, then interleave them as before.

We can also generalize to higher-order linear parametric weight functions. A graph with weight function $\sum_{i=1}^{k} \mu_i c_i(e)$ can be seen as a combination of k graphs with weights $c_i(e)$. We know that running the Contraction Algorithm on the graph outputs each min-cut (at this specific μ-weighting) with probability $\binom{n}{2}^{-1}$ as before. But we can represent this Contraction Algorithm as an interleaving of a contraction sequence for each of its k component graphs. As was argued above, the exact interleaving doesn't matter. Instead, we need only choose some number of contractions n_1 to perform using a prefix of π_1, then some number n_2 of contractions from a prefix of π_2, and so on. In total we must perform $\sum n_i = n - 2$ contractions to reduce G to 2 vertices and define a cut.

We have thus shown that every possible cut that can be produced by interleaving the π_i can be identified by specifying values n_i that sum to $n-2$. There are obviously at most n^{k-1} ways to do this (the final n_k is fixed by the others). The well-known "stars and bars" theorem [Fel68] gives a better bound of $\binom{n+k-1}{k-1}$.

It follows that the number of cuts that can be output by interleaving any k given permutations is at most n^{k-1}, which means that the number of minimish cuts overall is $O(n^{k+1})$.

In Section 6, we will extend our bounds to *nonlinear* parametric cost functions. First, we turn to lower bounds and enumeration algorithms for the linear case.

3. LOWER BOUNDS

We now show our results to be relatively tight by giving a lower bound of $n^{k/2}$ (versus the n^{k+1} upper bound) on the number of k-parameter minimish cuts.

We begin with the 2-parameter problem. Consider the graph $K_{1,n-2,1}$ consisting of two vertices s and t along with $n-2$ vertices v_i, $i = 1, \ldots, n-2$, each connected to s and t. We assign weights so that varying parameters forces some of the v_i to merge with s and the others to merge with t in any minimish cut.

We require a sufficiently large quantity M that we'll define later. Our first weight function sets $c_1(s, v_i) = M^i$ and $c_1(v_i, t) = 0$. Our second weight function assigns sets $c_2(s, v_i) = 0$ and $c_2(v_i, t) = M^{n-i}$. We consider the bicriterion weight function $c = \mu_1 c_1 + \mu_2 c_2$. Suppose we set $\mu_1 = M^{-j}$ while setting $\mu_2 = M^{j-n+1}$. It follows that $c(s, v_i) = M^{i-j}$ is at least M for $i > j$ and is at most 1 for $i \leq j$. Similarly, $c(v_i, t) = M^{j-i+1}$ is at most 1 for $i > j$ and at least M for $j \geq i$. In other words, exactly one of (s, v_i) and (v_i, t) is at most 1 and the other is at least M.

From this we can see that there is a cut of value $n-2$ in the graph: removing all edges of weight at most 1 removes one each of (s, v_i) and (v_i, t) and therefore separates s and t. This shows that the min-cut has value at most $n-2$. Assuming $M > n$, this means no edge of weight M crosses the min-cut. We can therefore contract all such edges without affecting the min-cut. This merges all v_i into either s or t, leaving a graph with 2 vertices and only one cut, which must be the min-cut; this shows that the cut we originally identified, consisting of all edges of weight at most 1, is in fact the min-cut given these values of μ_1 and μ_2.

It follows that if we vary j through the integers from 1 to $n-1$ to set values for the two parameters we get $n-1$ distinct min-cuts identified by how many of the v_i are on the s and t sides respectively. This shows that there can be as many as $n-1$ minimish cuts. Which is not interesting in itself, since the cycle has $\binom{n}{2}$ min-cuts without any parametric weight function at all.

However, we can generalize this construction. Make r distinct copies of the graph described above, each with its own 2 vertices s and t, its own $(n-2)/r$ middle vertices v_i with edges to s and t, 2 edge weight functions, and 2 parameters. Then identify all the s vertices from the different copies into a single vertex, and do the same for t, and extend each weight function to assign weight 0 on edges in copies of the graph other than its own. This essentially gives us an instance of $K_{1,n-2,1}$, but with r "groups" of edges each being driven by their own 2 parameters. By setting each pair of parameters separately according to the previous 2-parameter construction, we can independently choose how many of the $(n-2)/r$ middle vertices v_i in each copy must be merged with each of s and t in the min-cut of the combined graph. It follows that by varying these choices independently we can cause $((n-2)/r)^r$ distinct cuts to become the (unique) min-cut of this graph. In other words, if we are given $k = 2r$ parameters we can force up to $(2(n-2)/k)^{k/2}$ distinct minimish cuts, as compared to the $O(n^{k+1})$ upper bound.

One might object that the individual cost functions we are using are not "fair" since each cost function individually fails to connect the graph using its positive-weight edges. However, we can eliminate this objection by adding an infinitesimal cost to every zero-cost edge without changing the argument above (or the resulting min-cuts).

Just like our upper bound, our lower bound is based on an interleaving of permutations. Each of our k geometric weight functions defines a permutation (order by weight) on its edges; our choice of multipliers for the individual criteria the determines how those permutations interleave.

While our bound is meaningful for large k, it is useless for $k < 6$. A small tweak on the above construction, adding an n-vertex path from s to t, can improve the 5-criterion bound to $\Omega(n^3)$. But for 2–4 criteria we have no interesting separation from the single criterion case better than $k\binom{n}{2}$ (achieved by combining k distinct cycles).

4. PROBLEM GENERALIZATIONS

Our analysis above uses nothing about the specifics of the problem being solved; it relies only on an argument that the Contraction Algorithm "succeeds" with some probability and can be interleaved to run on parametric problems. Thus, any use of the Contraction Algorithm for a standard single-parameter problem has an immediate extension to parametric problems:

THEOREM 4.1. *Let $S(c)$ be a particular set of "interesting" cuts, such as the min-cuts, of an n vertex graph G with edge cost function c. If every cut of $S(c)$ is output by the Contraction Algorithm with probability $1/N$, then the size of $S(c)$ is at most N. Furthermore, given a linear parametric cost function $\sum \mu_i c_i(e)$ over k positive criteria $c_i(e)$, the total number of cuts in $S(c)$ over all nonnegative values μ_i is at most $\binom{n+k-1}{k-1}N$.*

PROOF. The proof of this theorem follows the analysis of Section 2. We showed that there are $\binom{n+k-1}{k-1}$ distinct interleavings, and that for any particular μ_i, one of those interleavings corresponds to the permutation we would get running the Contraction Algorithm with cost $\sum \mu_i c_i$ where each cut of $S(c)$ is output with probability $1/N$. $\square$

COROLLARY 4.2. *Given a k-parameter cost function the number of cuts of value α times the min-cut (at some parameter value) is $O(n^{k-1+2\alpha})$.*

PROOF. For the single-parameter problem the Contraction Algorithm yields any α-minimum cut with probability $\Omega(n^{-2\alpha})$ [KS96]. $\square$

COROLLARY 4.3. *Given a k-parameter cost function the number of r-way minimish cuts is $O(n^{k-1+2r})$*

PROOF. For the single-parameter problem the Contraction Algorithm yields any α-minimum r-way cut with probability $\Omega(n^{-2r})$ [KS96]. $\square$

COROLLARY 4.4. *Given a k-parameter cost function the number of α-times minimish r-way cuts is $O(n^{2\alpha(r-1)+k-1})$.*

PROOF. For the single-parameter problem the Contraction Algorithm yields any α-minimum r-way cut with probability $\Omega(n^{-2\alpha(r-1)})$ [KS96]. $\square$

COROLLARY 4.5. *Given a k-parameter cost function the number of α-times minimish* hypergraph *minimum cuts is $O(2^{\alpha r} n^{2\alpha+k-1})$*

PROOF. Kogan and Krauthgamer [KK15] proved that with a single parameter cost function each α-minimum hypergraph cut is found by the Contraction Algorithm with probability $\Omega(2^{-\alpha r} n^{-2\alpha})$. $\square$

Our final corollary is about the generalization of the min-cut problem to matroids. In a matroid, a *quotient* is the complement of any closed subset of the matroid (in particular, a cut of the graphic matroid). The minimum quotient is the quotient of minimum cardinality (or total weight, if each matroid element is given a weight). The Contraction Algorithm outputs any particular minimum quotient of an m-element rank-r matroid with probability $1/mr$ [Kar98].

COROLLARY 4.6. *Given a k-parameter cost function the number of minimish quotients of an m-element rank-r matroid is at most $m^k r$.*

5. BASIC ALGORITHMS

We have used an algorithmic thought experiment to count cuts, but now that we know this number is polynomial, we can use the same algorithmic idea *find* them all. We've proven that choosing random permutations for each parameter and considering all interleavings outputs each minimish cut with probability $\binom{n}{2}^{-1}$. Since there are $O(n^{k+1})$ such cuts, doing this $O(kn^2 \log n)$ times will select all minimish cuts with high probability. Each trial requires generating individual permutations π_i then attempting all interleavings.

For the thought experiment we could vague about the output of our algorithm, settling for a description of algorithms that "encounter" every potentially minimish cut with high probability. For certain search applications this may be suitable. For example, we may have an application that seeks a certain cut that it knows will be minimish, but that it can identify by other means if given the cut.

But for the problem of listing or accurately counting the minimish cuts of a particular graph, what we have so far is inadequate. We enumerate a set that *includes* all the minimish cuts, but some of them might not actually be minimish. So in addition to implementing the generation of all candidate cuts, we explain how to determine which of the cuts we visit actually is minimish, which enables us to count or list *only* the minimish cuts.

In this section, we give basic algorithms both for enumerating the candidate minimish cuts and for determining which of them actually do become minimum for certain parameters. In Section 7, we give better but more complex algorithms. We summarize here the runtimes we will ultimately achieve:

THEOREM 5.1. *For 2 parameter linear costs the $O(n^3)$ minimish cuts can be enumerated in $O(n^3\sqrt{m})$ time.*

THEOREM 5.2. *For 3 parameter linear costs the $O(n^4)$ minimish cuts can be enumerated in $O(n^4 \log^2 n)$ time, and for 4 parameter linear costs the $O(n^5)$ minimish cuts can be enumerated in $O(n^5 \log n)$ time.*

THEOREM 5.3. *For $k \geq 5$ parameter linear costs it is possible to enumerate a set* containing *all $O(n^{k+1})$ minimish cuts in $O(n^{k+1} \log n)$ time. Enumerating* only *the minimish cuts in this set can be done in $\tilde{O}(k^2 mn^{1+k})$ time via Ellipsoid or in $O(k(n \log n)^{2+k})$ strongly polynomial time.*

Note the odd fact that for 3 and 4 parameters our time bound is nearly optimal with respect to our bound on the number of cuts—we cannot hope to improve it significantly without improving the cut counting bound. For 2 and 5 or more parameters, we are off by factors of $\sqrt{m}$ and m respectively for different reasons we discuss below.

5.1 On the Input Size

There are some choices in defining the input size. The graph has some number of edges, and we could define m to be this quantity. But each edge can have up to k distinct costs associated with the different parameters. In a "sparse" representation, however, we might only represent the nonzero costs on each edge. We allow this sparse representation and define m to be the total number of nonzero coefficients over all the edges. The input will have this size when we record each nonzero cost coefficient explicitly. A natural way to do this is to use a distinct parallel edge for each distinct criterion in the parametrized function. We can still sum edges bearing the same criterion, but the graph may now have as many as kn^2 distinct edges.

There might be cases where the input problem can be smaller because coefficients are represented implicitly. Regardless, when k is a constant, there is no asymptotic difference from the standard edge count.

5.2 A Basic Interleaving Algorithm

We begin with a direct implementation of the abstract algorithm used to bound the number of minimish cuts. Given the permutations π_i, only n edges in each π_i can actually matter as the others are contracted by the time we encounter them in the permutation. Which n edges these are can be determined by contracting the edges of π_i alone in order, i.e. by running an instance of the standard one-parameter contraction algorithm. This is equivalent to computing a minimum spanning tree on the edges of the permutation, so can be solved over all k parameters in $O(m\alpha(n))$ time using Kruskal's algorithm (since the edges are already sorted) or $O(m)$ time with more sophisticated ones [KKT95].

Having thinned each permutation to n edges, we try all $O(n^{k-1})$ interleavings of the thinned permutations. For each of the $O(n^{k-1})$ lengths of prefixes of the first $k-1$ permutations, we collect all the edges in those prefixes and compute connected components to contract G in $O(m)$ time. We then contract enough of the final permutation to give us 2 vertices which correspond to the cut. In this case we need only the ending cut defined by the largest MST edge and not all the edges that matter in the contraction, so we can find it using binary search instead of a full MST algorithm. This gives a total of $O(mn^{k-1})$ time per trial. We must perform $O(kn^2 \log n)$ trials to find all minimish cuts with high probability, for total work of $O(kmn^{k+1} \log n)$. Similar algorithms apply for the various generalized problems.

This enumeration time bound is a factor of $km \log n$ greater than our bound on the number of minimish cuts. In Section 7, we show how to eliminate most of this factor.

5.3 Winnowing Candidate Cuts

Our interleaving algorithm has output a set of linear cost functions S such that whatever the minimum cut value is at some parameter setting, it is achieved by a function in S. But the converse need not be true: some functions in S might never be minimum. To output (only) the minimish cuts, we need to identify the functions that do become minimum at some parameter setting. Note that any minimum over S is minimum over all cuts with high probability given our proof that the minimum cut is in S for all parameters.

Consider the k-dimensional polytope P consisting of all parameter vectors μ for which the minimum cut in G exceeds 1. This polytope is frequently described as the intersection of 2^{n-1} constraints, one for each cut in G, which requires that the parametric value of that cut be at least one. However, we have previously given an algorithm that enumerates a set S of $O(n^{k+1})$ cuts that contains all minimish cuts—i.e., all cuts that can be minimum under some parametrization. Thus, if any cut in the graph has value below 1, some cut in S will have value below one. Conversely, to ensure G has minimum cut 1, it is sufficient to require that every cut in S have minimum cut 1. In other words, the polytope can be written more simply as the intersection of the constraints defined by the cuts in S. This means that this polytope has $O(n^{k+1})$ facets.

We wish to select all the minimish cuts out of S. A cut $C \in S$ is minimish if there is some parametrization μ for which C becomes a minimum cut. We can scale this μ so that the value of C is 1 while that value of every other cut is at least 1. This gives us a point μ that is in P that is *tight* for the constraint that the value of C is at least 1. Conversely, any cut C for which such a point μ exists is a minimum cut for that value of μ and therefore a minimish cut.

In summary, we have reduced the problem of selecting minimish cuts from S to that of determining, for each S, whether there is some $\mu \in P$ which assigns value 1 to C. Equivalently, we can see this as the problem of minimizing the value of C over the polytope P.

Since we are dealing with a set of linear constraints, we can solve this problem using linear programming. We have a polynomial number of constraints—one constraint per cut for a total of $O(n^{k+1})$ constraints—so can use any polynomial time linear programming algorithm. Better, since our problem involves only k variables μ_i, we can makes use of algorithms optimized for *fixed dimension* linear programming. The currently best [HZ15] takes $2^{O(\sqrt{k})}N$ time to solve a linear program with N constraints, so takes $2^{O(\sqrt{k})}n^{k+1}$ time per candidate cut in our problem. We need to apply it to every candidate cut, yielding an overall time bound of $2^{O(\sqrt{k})}n^{2(k+1)}$ time.

5.3.1 Linear Winnowing via Ellipsoid

We can get a better runtime by leveraging the *Ellipsoid Algorithm* [GLS88] which copes well with large numbers of constraints.

The Ellipsoid Algorithm can find a feasible point in a polytope given a *separation oracle*. The separation oracle, when given a query point (value of μ) outside the polytope, must return a constraint that is satisfied by all points in the polytope but not by the query point.

Given such a separation oracle, Ellipsoid will find a feasible point in a k-dimensional polytope if one exists, using $O(k^2 L)$ calls to the separation oracle, where L is the number of bits in the coefficients of the constraints defining the polytope. In our particular application, given the constraint C corresponding to the cut that we wish to test, we seek a feasible point in the polytope $P \cap C$. The coefficients in the constraints are simply the $c_i(e)$.

Since we have an explicit listing of the polytope's constraints S, there is an immediate separation oracle—to check for violated constraints. But we can do far better by returning to the origins of P as the cut polytope. A point not in P is a μ which either violates C or produces a graph whose minimum cut is less than 1. We can directly check whether C is violated; if it is not then we can look for a cut in G of value less than 1 given the parameters μ. This can be done by finding a minimum cut in G, which can be done in $O(m \log^3 n)$ time using a randomized algorithm of Karger [Kar00] after we spend $O(m)$ time computing the numeric value of edges given the parametrization μ. This is much faster than testing each cut in S explicitly. Once we have the violating cut X we can compute its parameter vector $x = \sum_{e \in X} c(e)$ in $O(m)$ time; the constraint $x \cdot \mu$ is the separating hyperplane we return to the oracle.

It follows that we can check whether any cut in S is minimish in $O(k^2 m \log^3 n \log U)$ time, where U is the ratio of maximum to minimum value of the coefficients $c_i(e)$. Since we have bounded the number of minimish cuts by $O(n^{1+k})$, this yields an $\tilde{O}(k^2 m n^{1+k})$ time bound for enumerating all minimish cuts.

While Ellipsoid is significantly faster then naive linear programming, it is not strongly polynomial. In section 7, we will give a more complicated algorithm that nearly matches the Ellipsoid approach and is strongly polynomial.

5.4 Algorithms for the Generalized Problems

Our algorithms apply to the generalizations of Section 4 with caveats. For r-way cuts, which we find using an early halt of the Contraction Algorithm [KS93], the application to minimish cut counting is immediate and we achieve similar time bounds to 2-way cuts for all values of k. We can implement the candidate enumeration algorithm as before and apply linear programming to winnow candidates. The explicit LP approach is unchanged. For Ellipsoid, we no longer have a linear-time separator oracle, but the Recursive Contraction Algorithm can be used to solve the minimum r-way cut problem and provide a separator oracle in $\tilde{O}(n^{2r})$ time for an overall time bound of $\tilde{O}(n^{2r}n^{k-1+2r}) = \tilde{O}(n^{k-1+4r})$.

We can similarly enumerate α-minimish cuts. Again, to enumerate candidate cuts, we use the standard Contraction Algorithm in the single parameter case with an early halt) [KS93] to enumerate all candidate cuts. The Ellipsoid algorithm also applies: given a candidate cuts, we can write a linear program seeking a setting for μ that makes the value of that cut less than α while the minimum cut is at least 1. Again, we use a a suitable cut algorithm as a separation oracle. This gives a bound of $\tilde{O}(mn^{2\alpha+k-1})$ time to enumerate α-minimish 2-way cuts using the linear-time min-cut oracle and $\tilde{O}(n^{2r}n^{2\alpha(r-1)+k-1})$ time to enumerate α-minimum r-way cuts using the Recursive Contraction Algorithm.

For hypergraph cuts and matroids, our techniques apply in general but certain details change—for example, it is no longer true that the number of (hyper)edges can be bounded by n^2. This worsens our time bounds somewhat. Details will be provided in the full paper.

6. NONLINEAR PARAMETRIZATIONS

Although it is presented with k parameters, the linear parametrization we have been considering actually offers only $k-1$ degrees of freedom, since $\sum \mu_i$ can always be scaled to 1 without changing the set of min-cuts. Thus, our $n^{k-1}\binom{n}{2}$ bound shows one factor-n increase in minimish cuts per degree of freedom. We prove this more generally. Consider, for example, counting the set of minimish cuts under parametric weights $c_0(e) + t \cdot c_1(e) + t^2 c_2(e)$. The above analysis can only treat t and t^2 as two entirely independent parameters and derive a bound of $O(n^4)$ minimish cuts. But we will now prove that there are $O(n^3)$.

THEOREM 6.1. *If every edge cost is a degree-k polynomial in t with positive coefficients, then as t ranges over the positive reals (and the positive-cost edges span G) the number of distinct minimish cuts is $O(kn^3)$ and they can be output in $O(kn^3\sqrt{m})$ time.*

6.1 Constructing Weighted Permutations

To prove the theorem we start with the 2-parameter case. In order to analyze this case, we consider a specific method for generating the Contraction Algorithm's weighted permutations based on a *Poisson process*. This idea, of defining order via a Poisson process, was used by Lomonosov and Poleskii in their analysis of network reliability [Lom94]. Given a graph with edge weights $c(e)$, we build a weighted permutation by generating a *score* s_e for edge e distributed exponentially with rate $c(e)$, and then sorting edges by increasing score. In other words, $\Pr[s_e > t] = e^{-c_e t}$. Equivalently, we can generate s_e by generating an independent exponential variable z_e distributed as $\Pr[Z > t] = e^{-t}$, then set $s_e = z_e/c(e)$.

It is well known [Fel68] that in such a Poisson process, the first edge in the order is edge e with probability proportional to $c(e)$. Furthermore, the remaining edge ordering is "memoryless": conditioned on the identity and score of the first edge, the (additional) scores of the remaining edges are once again exponentially distributed with the same rates. Thus, the order of edges by score satisfies exactly the properties we demand of the weighted permutation for the Contraction Algorithm.

Given the permutation based on the scores, we implement the Contraction Algorithm by going through the edges in score order and contracting each edge that does not form a self loop, stopping when we have contracted to two vertices and the next edge we contract would reduce to one. This is equivalent to running Kruskal's minimum spanning tree algorithm (on edge *scores*, not weights) up to the final (largest) MST edge. In other words, the cut defined by scores is precisely the cut produced by taking the MST of the scored edges, then removing the edge of largest score.

6.2 Counting Interleavings

To apply this idea to the parametric problem, we consider our quadratic-parametrized graph as a union of three graphs G_i where G_i has edge weights $c_i(e)$ which are then multiplied by t^i. For now, assume each G_i is connected. As just described, we generate an independent exponential variable $z_{i,e}$ for each edge in G_i, then divide it by $t^i c_i(e)$ to yield the properly weighted score, $s_{i,e} = z_{i,e}/t^i c_i(e)$. Sorting by these $s_{i,e}$ produces a weighted permutation of the edges of each G_i, all interleaved to form the complete permutation.

We seek the cuts defined by contracting in the permutation order, as we vary the parameter t. Varying t does not affect the $z_{i,e}$ but does change the scaling of each $z_{i,e}$ by the relevant t^i. Notice, however, that varying t does *not* change the order π_i of edges in a single G_i, since each edge is scaled by the same quantity t^i. So the order of edges in G_i is determined by the quantities $z_{i,e}/c_i(e)$. This immediately tells us that any edge of G_i that is *not* in the MST of G_i also cannot be in MST(G), since such an edge will always be spanned by a path of lower-score edges in G_i. Thus, as in the linear case, we can immediately discard all but n edges of each π_i. However, as we vary t, the interleaving of the π_i will change, so MST(G) is some combination of edges from the MST(G_i).

We now assess the number of permutations that emerge starting at a large value of t and slowly reducing t to 0. As discussed above, the cuts correspond to minimum spanning trees defined by the scores, so we will actually assess the number of distinct minimum spanning trees that emerge as the parameter t varies. The scores $s_{i,e}$ for edges in G_i are $O(1/t^i)$. So for sufficiently large t, all scores in G_2 are smaller than those in G_1 and G_0, which means that all edges in G_2 precede all edges in G_1 and G_2. Thus, the MST of G is just the MST of G_2 (which is unchanging as t varies since all edges of G_2 vary proportionately). As we shrink t, the edge weights in G_0 are unchanged, while the edges of G_1 grow past them, and the edge weights of G_2 grow even faster to ultimately pass those of G_1.

Intuitively, as t shrinks the edges of G_2 in the MST will slowly be replaced by edges of G_1 and then edges of G_0 (as $t \to 0$, the MST of G must ultimately become the MST of G_0). Since there are only $n-1$ edges in the MST of G_2, which can be replaced by only $n-1$ edges of the MST of G_1 and then by $n-1$ edges of the MST of G_0, this suggests that there can be only $2n-2$ changes to the MST. But we need to be a little bit careful, as it seems possible for an edge to leave and re-enter the MST several times as other edges pass it in opposite directions.

We therefore define the *potential* of edge $e \in G_i$ to be i, and define the potential of the (current) MST to be the sum of potentials of its edges. The MST potential is at most $2n-2$ and at least 0. We will show that each change to the MST decreases the potential by at least 1, which demonstrates that there can be only $2n-2$ changes to the MST.

As t shrinks, the MST changes whenever some edge leaves the MST and another enters to replace it. The entering edge creates a cycle with the leaving edge, which breaks the cycle when it leaves. At this moment of transition, the edges are (both) the maximum score edges[1] in this cycle—this is the *red rule* which says that an edge is excluded from the MST iff it is the largest edge on some cycle [Tar83]. The leaving edge must be increasing faster than the entering edge as t decreases. So if the leaving edge is in G_i, the entering edge (with a more slowly increasing score) must be in some G_j with $j < i$. In other words, the new MST edge has lower potential than the old. This demonstrates our claim, that every change in the MST decreases its potential.

It follows that there are at most $2n$ changes to the MST. This means that at most $2n$ distinct cuts are defined by the interleaving as we vary t. Continuing the argument as in the basic analysis, we conclude that the total number of minimish cuts is at most $2n \cdot \binom{n}{2} = O(n^3)$.

We initially assumed for simplicity that each G_i was connected, but as before it is only necessary that the combined graph be connected. If the G_i are not connected, their minimum spanning *forests* will simply have fewer edges and thus fewer edge crossings than our analysis pessimistically assumed. Equivalently, we can add connecting edges to these graphs with infinite scores so they never enter the MST.

6.3 Generalization

There is a clear generalization to higher degrees. For a degree-k polynomial parametrization, where our basic analysis would suggest a bound of $O(n^{1+k})$ min-cuts, our new potential-function analysis shows that the number of minimish cuts is $O(kn^3)$. And it generalizes beyond polynomials: the only fact we used was that the different additive terms in the parametric function had a consistent order by rate of increase. In other words, any parametric function of the form $\sum_{i=1}^{k} c_i f_i(t)$ for positive c_i and f_i yields $O(kn^3)$ minimish cuts so long as each ratio $f_i(t)/f_j(t)$ is monotonic, since this ensures that when we consider the graph as a union of k evolving graphs, two edge scores can only cross once as

[1]As is usual, we can use lexicographic tie-breaking to ensure that no 3 edges tie in value. Although in this case that is in fact unnecessary, as one can show that the probability that the random edge coefficients yield a 3-way tie is 0.

t changes. Equivalently, the quantities $f_i(t)/f_i'(t)$ must be non-crossing, where f' is the derivative.

As another application of this generalization, notice that each edge could have its own, arbitrary parametric capacity function $c_e(t)$. So long as the monotonicity criterion just given holds—that for any pair of edges e and f, the function $c_e(t)/c_f(t)$ is monotonic—we are in the situation of the previous paragraph but with m distinct functions. A corollary follows:

Corollary 6.2. *Suppose the capacities of edges e on graph G are arbitrary nonnegative functions $c_e(t)$ such that for every pair of edges e and f the function $c_e(t)/c_f(t)$ is monotonic. Then as t varies, the number of distinct cuts of G that become minimum is at most $mn\binom{n}{2}$.*

Proof. Our analysis above derived an $O(kn\binom{n}{2})$ bound for k classes of "once-crossing" edge functions. Here we simply think of each edge as forming its own class, so $k = m$. In more detail, we run the Contraction Algorithm where permutation score for edge e is $s_e/c_e(t)$ where s_e is an independent exponential random variable. The permutation changes as we vary the t (and thus the $c_e(t)$, but the monotonicity of the $c_e(t)/c_f(t)$ means two edge scores only cross once. Thus our analysis is the same as before. □

6.4 Enumerating Interleavings

Our proof has a straightforward algorithmic implication. As in the linear case, we first need to efficiently enumerate all the candidate cuts that arise during interleavings, and then determine which of these cuts actually becomes minimum for some value of the (single) parameter.

To enumerate minimish cuts, we need to track the MST as it evolves with changing t, and in particular the two pieces it forms when the heaviest edge is removed. These define the candidate minimish cuts. The MST depends only on the relative order of edges, not their values. Thus, it is sufficient for us to track changes in the edge *order* as t varies, and recompute the MST each time the order changes. The bound on the number of changes was the core of our bound on the minimish cuts.

We can track the order of edges using a "sweep line" method from computational geometry, which was used by Sarnak and Tarjan [ST86] in a point location data structure. The key observation is that ordering changes are local: they occur through the exchange of two adjacent edges in the order.

We thus start with edges ordered as they should be at $t = \infty$ (breaking ties in any consistent way). We then compute, for every edge, the time at which its numeric value crosses each of its neighbors'. The first crossing time (over all edge pairs) is the first time the MST changes. At that moment, we can compute the implied cut and its value. We also swap the edges whose order has switched.

Determining the next crossing requires a scan of the edges, and so does computing the value of the implied cut. Thus, we can find and assess each cut in $O(m)$ time, yielding an overall time bound of m times our bound on the number of MST changes. For example, under the quadratic parametrization all n interleavings can be found in $O(mn)$ time. This yields a bound of $O(mn^3)$ to find the (possible n^3) candidate minimish cuts.

It seems excessive to spend m time identifying each cut, and we can speed up part of the work. In addition to storing the edges in order, we can store neighboring edge pairs in a priority queue keyed by crossing time. Extracting the next crossing time takes $O(\log n)$ time; computing the crossing times for the new neighbors also takes $O(\log n)$ time. Having identified the necessary ordering changes in $O(\log n)$ time, we can invoke a dynamic MST data structure [HdLT01] that can maintain an MST during edge insertions and deletions at a cost of $O(\log^2 n)$ per insertion or deletion. We use the rank of each edge in the array as the key for this MST data structure; when two edges cross, we delete them with their old ranks and insert them with their new ranks.

In summary, we have shown how to maintain the evolving MST that corresponds to our candidate minimish cuts at a cost of $O(\log^2 n)$ per minimish cut (plus an $O(m \log^2 n)$ cost to insert the initial neighbor pairs). But to assess the cuts values, we also need to sum the values of the edges crossing the cuts. Naively, this will take $O(m)$ time per cut.

6.5 Faster Interleaving

We can improve the interleaving step by using a contraction batching technique. We have just shown how to identify the sequence of MST changes—each an insertion and deletion of one edge—that defines the cuts we need to evaluate. Consider a series of d of the changes. From the starting MST we delete only d edges, which means that over the course of the d deletions, the remaining MST edges are *never* deleted. Thus, before we consider this sequence of changes, we can contract all the undeleted edges, leaving us with a graph that has only $d + 1$ vertices. By merging edges we can also ensure that the number of edges is $O(d^2)$.

We apply this idea recursively. Suppose we have an n vertex graph with a given starting MST and we wish to compute the (parametric) values of cuts creating during a series of at most n changes. Divide the sequence of changes into a first sequence of $n/2$ and a second sequence of $n/2$. Construct the MST as it exists at the midpoint in $O(m)$ time by looking at the edge insertion and deletion times. We can now solve two subproblems involving a starting tree and a sequence of $n/2$ changes. As argued above we can contract each problem to a size $n/2$ graph before we begin. This yields a recurrence in n, the size of both the graph and the number of changes, where $T(n) \leq 2(\min(m, n^2) + T(n/2))$.

To solve this recurrence, note that in particular $T(m, n) \leq kn^2 + 2T(n/2) = O(kn^2)$. Looking more closely, observe that at each layer of the recursion the number of subproblems doubles while the work per subproblem remains m, until we reach a layer where there are $n/\sqrt{m}$ subproblems of size $\sqrt{m}$. At this point our simpler $O(kn^2)$ bound gives an $O(km)$ bound on the work in each subproblem. This dominates the prior work, yielding a total bound of $O((n/\sqrt{m}) \cdot km) = O(kn\sqrt{m})$ time per interleaving.

If our sequence contains some larger quantity $d \geq n$ of changes, we can break it up into d/n subproblems each with n changes, compute the MST at each break-point, and apply our algorithm for n changes to each subproblem from the starting MST. This yields a runtime of $O((d/n)n\sqrt{km}) = O(d\sqrt{km})$. In other words, we can evaluate the cut value functions at a cost of $O(\sqrt{km})$ time per cut. Thus, for example, listing candidate minimish cuts with the quadratic parametrization can be solved in $O(n^3\sqrt{km})$ time.

6.6 Winnowing Minimish Cuts

Our enumeration procedure produces a list of candidate cuts, each with a quadratic function which sums the quadratics of the edges in the cut and represents the cost of the cut as a function of the parameter t. We know that every minimish cut is in the set, but some of the cuts might not be minimish. We need to determine which cuts truly become minimum for some value of the parameter t.

For the case of quadratic parametrization, this is equivalent to determining the *lower envelope* of the set of parabolas. For the single parameter problem, this can be done in $O(n \log n)$ time using a simple divide and conquer procedure [SA95].

More generally, the time to find the lower envelope of n curves is $O(\lambda_s(n) \log n)$ where s is the number of times any two curves may intersect (two in the case of our parabolas) and $\lambda_s(n)$ is the maximum length of an (n, s)-*Davenport-Schinzel* sequence. We have $\lambda_2(n) = O(n)$ which yields the $O(n \log n)$ bound we claimed. If we consider cubic functions with 3 potential crossing per pair, $\lambda_3(n) = O(n\alpha(n))$. The quantity grows with larger s but remains very close to $O(n)$ [ASS89]. Thus, for any fixed s, the time to find the lower envelope given the candidate cuts is dominated by the $O(n\sqrt{m})$ time we take to find the candidates.

7. FASTER INTERLEAVING

In Section 5 we gave bounds of roughly $O(m)$ times the number of minimish cuts for actually finding them. These bounds were "balanced" in the sense that it cost us roughly a factor m slowdown to enumerate each candidate cut and also a factor of m to check if each candidate cut was actually minimish using the Ellipsoid algorithm. In this section, we give improved interleaving algorithms and in the following we give improved winnowing algorithms. We show that for all $k \geq 3$ we can enumerate candidate cuts in $\tilde{O}(1)$ time per cut. Conversely, we show that for 2–4-parameter problems we can winnow all candidates in $\tilde{O}(1)$ time per cut. Combining these two facts gives us a faster enumeration algorithm for 2-parameter cuts and a near-optimal (relative to the cut bound) enumeration algorithm for 3- and 4-parameter cuts. We also give a strongly polynomial winnowing algorithm to complement our Ellipsoid-based approach.

7.1 On the Output Size

If we're going to beat a runtime of $O(m)$ per cut, we have to find a compact output representation. Outputting a cut as a vertex partition or edge set will take time n or m respectively. Thus, we focus simply on outputting the *value* of a cut. Here, that value is not a number but a parametric function, representing the sum of all the functions of edges crossing the cut.

More precisely, the interleaving algorithms repeatedly contract the graph to a pair of vertices. As vertices are contracted, parallel edges can be merged by addition. Each edge that we are summing carries a vector of k coefficients representing its linear-parametric cost. Thus, when we finish contracting to a pair of vertices, we also have a single edge connecting them, bearing a k-coordinate vector representing the cost of the selected cut as a linear function of the parameters. It follows that the output of our enumeration is a set S of k-dimensional vectors representing linear functions. Our analysis has shown that with high probability, for any possible (positive) setting of the parameters, one of these linear functions defines the minimum cut of the graph at that setting of the parameters.

We may be concerned about outputting cuts more than once. As was shown elsewhere [Kar94], we can "fingerprint" each cut by assigning a random integer to each vertex, and exclusive-or-ing the integers of any vertices we merge. A cut (pair of vertices) now corresponds to a pair of integers. With polynomial integers, the probability that two cuts get the same fingerprint is negligible. Storing these fingerprints in a hash table as we encounter them lets us output each cut only once. This is useful if, for example, we wish to exactly count the number of minimish cuts in a given graph without spending the time to determine the vertex partition of each cut.

7.2 Batched Interleaving

We can improve our naive interleaving algorithm by batching the contractions as we did in the quadratic parametrization. Suppose that in the 2-parameter problem we have generated two permutations π_i and thinned them to n edges each and wish to output the cuts corresponding to their interleavings. The naive approach above required $O(mn)$ time for one permutation so $O(mn^3 \log n)$ time overall.

As an alternative, we observe that in an n vertex graph, each interleaving must perform at least $n/2$ contractions using either π_1 or of π_2. Thus, we can generate two recursive subproblems by contracting a length-$n/2$ prefix of each π_i, then recursively output the cuts formed by interleavings in the two subproblems. These contractions initially need not decrease the number of edges below m, but as contractions occur we can merge parallel edges and bound the total number of edges by $O(n^2)$. We also thin the interleavings so their size is bounded by n. This leads to a recurrence $T(m, n) \leq \min(m, n^2) + 2T(n/2)$.

We already solved this recursion for the nonlinear parametric problem; it yields $T(m) = O(n\sqrt{m} \log n)$. This in turn reduces the overall minimish cut enumeration time from $O(n^3 m \log n)$ to $O(n^3\sqrt{m} \log n) = O(n^4 \log n)$.

7.3 Four or More Parameters

With more parameters the same batching technique yields an even better bound. Consider the 4-parameter problem on a graph of n^2 edges (we skip over the 3-parameter problem and return to it in Section 7.4). Choose a large integer q. Given the four thinned permutations π_i, divide the enumeration of all interleavings into subproblems, where a given subproblem focuses on interleavings where the number of edges contracted in each π_i is at least $a_i n/q$ and less than $(a_i + 1)n/q$ for some integers a_i with $0 \leq a_i < q$. Since the total number of contractions to produce a cut is $n - 2$ we must have $\sum a_i < q$ and $\sum a_i > q - 4$, which means there are $O(q^3)$ subproblems. For the subproblem indexed by a_i we know that there are at least $a_i n/q$ edges contracted from each π_i so we can contract this many *before* we recursively explore the rest of the edges to contract. At this point we know there remain at most n/q contractions to be done in each π_i to get a cut (2-vertex graph), which means the size of the contracted graph is at most $4n/q$. Thus we can discard any graphs larger than this, then recursively enumerate all cuts produced by contracting at most an additional n/q edges from each π_i. Doing this for all q^3 subproblems gives a recurrence $T(n) = O(q^3(n^2 + T(4n/q)))$. If we now set $q = n^{1/4}$ we find $T(n) \leq kn^{11/4} + n^{1/4}T(4n^{3/4})$. By induction $T(n) = O(n^3)$ since the recursive term dominates.

We need to perform $O(n^2 \log n)$ iterations of this experiment to find all cuts with high probability, for an overall running time of $O(n^5 \log n)$. Relative to our $O(n^5)$ bound on the number of 4-parameter minimish cuts, this suggests we are producing each cut in nearly constant time. The same technique applied to $k \geq 4$ yields an $O(k^2 n^{k+1} \log n)$ time bound for enumerating cuts with any $k \geq 4$ parameters.

7.4 The 3-Parameter Problem

Our recursive algorithm is near-optimal relative to our counting bound for $k \geq 4$ parameters, but doesn't quite work for the 3-parameter case. We'd like to achieve an $O(n^2)$ bound for a 3-parameter interleaving, which forces us to set $q = O(1)$ in the analysis above. But this only yields $T(n) = O(n^{2+\epsilon})$. Here we show how to remove the n^ϵ term. This waste term arises from the basic recursive algorithm because we solve certain subproblems multiple times. We address this issue by memoizing the subproblems so that each only needs to be solved once. The resulting algorithm generalizes to any number of parameters, but is not worth the added complexity for larger k since the simpler divide and conquer algorithm is already near optimal.

We are analyzing a series of edge contractions. Note that as we contract edges, some of the other edges in the π_i become "inactive" since they spanned by the previously contracted edges, so the edge's endpoints are merged by the time we reach it.

Consider the 3-parameter version of the problem. We are given 3 permutations π_i and generate a cut by contracting some prefix p_i of each of the π_i to produce a 2-vertex graph. We will describe a "canonical" way to perform these contractions of the p_i that will help us create them all efficiently. We contract in batches whose sizes are powers of 2. Starting with $j = \log n$ and decreasing, we consider each π_i in order. For each π_i, if contracting the remaining edges of p_i will reduce the size of G by more than 2^j, then contract enough of p_i to reduce the size of G by *exactly* 2^j.

Observe that by induction, after we execute the phase for a particular value of j, the maximum amount by which contracting p_i can reduce the size of G will be 2^j. This is clearly true when $j = \log n$. In phase j, we know by induction that the maximum possible reduction is 2^{j+1}. If it exceeds 2^j then we contract more of p_i, reducing the size by exactly 2^j, which means that the maximum remaining reduction is at most $2^{j+1} - 2^j \leq 2^j$. Contractions done using the other permutations can only further reduce the amount of "available reduction" in the permutation we are considering, as some of its edges become redundant self loops.

We've described the canonical order in which we contract the p_i. Using this idea, we show how to generate the graphs that result from the canonical-order contractions of *all* n^2 *interleavings* that yield the candidate cuts we want.

In particular, suppose that we have already generated all the graphs and "residual prefixes" that arise after phase $j+1$ of the canonical contraction order for *all* interleavings we care about. We show how to generate all the results of phase j. Consider a particular interleaving with prefixes p_i whose graph at the end of phase $j+1$ is H. This interleaving's canonical order now performs 0 or 2^j additional contractions on each of the remaining p_i in order. We do so, starting from H, to produce a new intermediate result H'. We then go through the remaining parts of the prefixes p_i and remove any edges whose endpoints were already contracted, as these are irrelevant.

We can bound the size of each intermediate result after phase j. Note that in H', each remaining p_i can reduce the size by at most 2^j. Since by assumption the p_i taken together must reduce H' to 2 vertices, this implies that the maximum possible size is of H' is $2 + 3 \cdot 2^j = O(2^j)$. This in turns means that the length of each p_i is $O(2^j)$. Also, since we can merge parallel edges in H', we conclude that the number of edges in H' is $O(4^j)$.

We can also bound the *number* of intermediate results after phase j. Naively, we have constructed each intermediate result by making three binary choices per phase over $\log n - j$ phases; it follows that the total number of outcomes is $2^3(\log n) - j = n^3/8^j$. But the size of the remaining graph is at most $3 * 2^j$ as discussed above. Thus, we know that the sum of the chosen powers of 2 must be between $n - 3 * 2^j$ (since we cannot contract less and get a graph of the correct size) and n (since we cannot contract more). This means the third value (total length of p_3), which must be a multiple of 2^j, is essentially determined by the first two, which implies that the number of combinations that satisfy this constraint is $O(n^2/4^j)$.

How long does it take to construct these intermediate results? We have just argued that there are $O(n^2/4^j)$ intermediate results at the start of phase j. To generate the level-j intermediate results that follow from a given one, we need to try 8 possible combinations of 0 or 2^j contractions of each p_i. Each starts from a graph of size $O(2^j)$ which means that the work of contracting (and computing the new graph with its merged parallel edges and reduced p_i) is $O(4^j)$. Thus the total work for doing all the extensions is $O(4^j \cdot n^2/4^j) = O(n^2)$. We can then discard all the results whose total size is greater than the derived bound of 2^j (because these cannot be intermediate results that proceed to contract fully to yield a cut).

In summary we have shown how to extend from each phase $j+1$ to phase j in $O(n^2)$ time. Thus, starting from phase $\log n$ where the only result is the original graph and computing to phase 0 where we have the results of contracting any arbitrary set of prefixes to produce 2 results takes time $O(n^2 \log n)$. It follows that the time to try this with the $O(n^2 \log n)$ necessary random permutations is $O(n^4 \log^2 n)$, a factor of $O(\log^2 n)$ greater than our $O(n^4)$ bound on the possible number of 3-parametric minimish cuts.

8. BETTER WINNOWING

With near-optimum algorithms for enumerating candidate minimish cuts, there is even more motivation to improve the time to verify the candidates. In this section we describe two approaches. Convex hull algorithms give near-optimal winnowing algorithms for 2–4 parameters. For larger k, we give an algorithm that nearly matches the ellipsoid algorithm's time bound but is strongly polynomial, costing roughly n^2 per candidate cut.

8.1 Convex Hulls

One way to winnow faster is to take consider the dual of our cut polytope. Given our set S of candidate k-parameter vectors, identifying the minimish cuts is equivalent to asking which of the k-vectors in S are on the convex hull of S, since one definition of a convex hull point is that it is a point minimizing $\mu \cdot x$ over $x \in S$ for some μ. In our problem, we are also adding the requirement that $\mu \geq 0$. To exclude cases where $\mu_i < 0$, we choose a sufficiently large value M and, for each standard basis vector e_i, add the vector Me_i to S. Now whenever $\mu_i < 0$ the quantity $\mu \cdot x$ will be minimized by Me_i, so any other $x \in S$ on the convex hull must correspond to a $\mu \geq 0$.

Convex hull algorithms have been studied intensively. For 2 and 3 dimensions (parameters) algorithms are known with runtime $N \log N$ for N points [dBSvKO00]. In our case, the number of points is the number of candidate cuts, $O(n^3)$ for 2 parameters and $O(n^4)$ for 3 parameters, so the time to find the convex hull (and thus the minimish cuts) is $O(n^3 \log n)$ for 2 dimensions and $O(n^4 \log n)$ for 3 dimensions.

Our 4-parameter problem can also be solved this way. Any parameter μ making a given cut minimum can be scaled to have coordinate sum 1 and still make the same cut minimum. Thus, we can add the constraint that $\sum \mu_i = 1$ and reduce the dimensionality of our 4-parameter problem to 3. In implementation, this means that we rewrite our 4-parameter cut function as a linear function of just 3 parameters $c_1\mu_1 + c_2\mu_2 + c_3\mu_3 + c_4(1 - \mu_1 - \mu_2 - \mu_3$, and find the convex hulls of the resulting 3-coordinate vectors in $O(n^5 \log n)$ time. This proves Theorems 5.1 and 5.2.

In higher dimension, the best known approach to convex hulls is essentially the linear programming approach we already described in Section 5. We used the Ellipsoid algorithm to solve this problem quite quickly in Section 5.3.1. It doesn't match the linear speed of the 2- and 3-parameter convex hull algorithms. However, an interesting advantage of the Ellipsoid approach over the convex hull algorithms is that it is *incremental*: given any one candidate cut, it can determine whether it is minimish, and if so find a proof that it is, in $\tilde{O}(m)$ time. The convex hull algorithms require a superset of *all* the candidate cuts in order to determine which are actually minimish.

Our convex hull approach also applies to some of the generalized problems of Section 4. In particular for 2 to 4-parameter problems we can use the convex hull approach to get $O(n^{1+2r}\sqrt{m})$ and $O(n^{2+2r}\log^2 n)$ time enumeration algorithms. It does not apply to the α-minimish cut problem as some α-minimish cuts will not be on the convex hull of the polytope—they'll instead be "within α" of it. We can still apply the Ellipsoid technique for α-minimish cuts.

8.2 Strongly Polynomial Winnowing

The Ellipsoid Algorithm provides a fast way to certify minimish cuts. But one drawback is that it is not strongly polynomial. Its runtime is affected by the number of bits used to represent the input coefficients. Our lower bound graph is an example of one that uses exponentially large coefficients and would therefore require substantial time to solve using ellipsoid. In this section, we give a slightly slower but strongly polynomial algorithm.

We seek a parameter setting μ^* that makes a particular cut C minimum. Our approach comes in two parts. First, we show that if we are able to get "close" to a valid μ^*, we can find a set of $O(n^2)$ "important" cuts and find μ^* exactly by linear programming over the resulting $O(n^2)$ constraints. We then show how to get close to the desired μ^* by partitioning the parameter space into a small number boxes (products of intervals) that are each sufficiently small.

Through this section, we will assume without loss of generality that the μ^* we are seeking sets the value of the minimum cut to 1, since we can always scale μ^* to change all cut values proportionately. In fact, we will *enforce* this restriction by adding a constraint $\sum \mu_i = 1$ to our linear program.

8.2.1 *Rounding from a Close Guess*

Our first insight is that if we are "close" to the value μ^* which demonstrates a cut C to be minimish, then we can quickly find μ^*. In particular, suppose we are given a *box of ratio* $1 + \epsilon$, which is a set of k intervals $[\alpha_i, \beta_i]$ such that $\alpha_i \leq \mu_i* \leq \beta_i$ and also $\beta_i/\alpha_i \leq 1 + \epsilon$.

Given the box, we can find μ^* as follows. At μ^* the cut C is a minimum cut of value 1. As with the Ellipsoid approach, the problem of finding μ^* is a linear program. Let T be the set of other cuts that define tight constraints at μ^*. This means the cuts of T are also minimum cuts of value 1 at μ^*. Since within the box each μ_i can change in value by a relative factor of at most $1 + \epsilon$, we know that the value of each cut can change by at most $1 + \epsilon$, which means that the minimum cut *anywhere* in the box is at least $1/(1 + \epsilon)$. At the same time, we know that every cut in T takes on value *at most* $1 + \epsilon$ everywhere in the box. It follows that at any point in the box, every cut of T is a most $(1 + \epsilon)^2$ times the minimum cut.

In earlier work [Kar00], we showed that there are $O(n^2)$ cuts of value less than $3/2$ the minimum and they can all be found in $\tilde{O}(n^2)$ time. Setting $\epsilon = .2$ so that $(1 + \epsilon)^2 = 1.44$, let us find this set of $O(n^2)$ cuts. By the argument above, all cuts of T will be in this set of n^2 cuts that we find. These cuts in T are sufficient to constraint μ to its optimum μ^*. Thus, if we solve our linear program, requiring that C have minimum cut at most 1 while all the other cuts in S have value at least 1, then the linear program will yield μ^*. (We also include the constraints $\alpha_i \leq \mu_i \leq \beta_i$ in order to force the solution to be inside our box). Thus, we can use a constant-dimension *strongly polynomial* linear programming algorithm [HZ15] to find the optimum in $2^{O(\sqrt{k})}n^2$ time. Given the solution, we can verify that it does indeed make C minimum in $\tilde{O}(m)$ time using a standard minimum cut algorithm [Kar00].

Notice that our procedure needs to give the linear programming solver the constraints defined by $O(n^2)$ cuts. Constructing each constraint separately from the graph might require $O(m)$ time per cut to sum the parameter vectors of all edges crossing the cut. Note, however, that we find these cuts by executing a min-cut algorithm at some point in the box. This algorithm necessarily computes the numeric costs of (near) min-cuts as it is identifying the cuts; we can easily piggyback the summing of the edges' parameter vectors onto the summing of the numeric edge costs, taking at most a factor k slowdown in the algorithm.

8.2.2 *Making a Close Guess*

We've just shown that once we are close to μ^* in relative terms we can find μ^* in $O(n^2)$ time. We now show how to get close.

It would be nice to do this for each cut separately, but we haven't yet found a way. Instead, rather than designing an algorithm to verify each candidate cut separately, we return to our original interleaving idea but produce a different interleaving procedure designed to certify each cut as we find it. We again consider our parametric problem as taking k distinct graphs G_i with edge costs $c_i(e)$ and combining the k graphs with parameters μ_i. This time, instead of choosing a random order and considering interleaving by rank in the order, we order all edges in each graph by cost and use interleavings to balance the *numeric values* of the interleaved edges.

We will discuss our algorithm with respect to a particular minimish cut C and describe an "oracle" that will identify and certify C by making certain guesses. This will show that every minimish cut can be certified by trying all possible combinations of guesses. We will then bound the number of possible combinations of guesses which will bound our overall runtime for enumerating all minimish cuts. This will also give an alternative (deterministic) proof of a bound on the number of minimish cuts; however, it will be less tight and more complicated than the randomized thought experiment of Section 2.

So consider a particular minimish cut C and let μ^* be some setting of the parameters the makes C a minimum cut with cost 1 (we can always scale μ^* to make this last fact true). Given this μ^*, define the *effective cost* of edge $e \in G_i$ to be $\mu_i^* c_i(e)$. This is the cost of e at parameter setting μ^*. We will call an edge *heavy* if its effective cost exceeds 1 (the minimum cut) and *light* otherwise. For graph G_i, define the *pivot edge* p_i to be the minimum cost heavy edge. This means that all edges with costs at or exceeding p_i have effective cost exceeding 1. We now posit that an oracle has informed us of all the p_i.

Note that if e is heavy it cannot cross the minimum cut. To take advantage of this, we add the constraints $\mu_i \cdot c_i(p_i) \geq 1$ for each i, which is true at μ_i^* by the definition of p_i, and we contract all heavy edges for *any* graph. We claim that if we can make C be a min-cut (of value 1) in this contracted graph with the new constraints, then we have made C minimum in G. To see this, suppose we have found μ' that makes C a min-cut of value 1 in the contracted graph. This means C is smaller than any cut in the contracted graph. Any cut of G *not* in the contracted graph is missing because a heavy edge crossing it was contracted. But we added constraints forcing such heavy edges to have effective cost 1; thus any cut with a heavy edge crossing will have total cost at least 1 so will be larger than C.

We can therefore assume without loss of generality that G has no heavy edges. In other words, μ^* induces effective cost at most 1 on every edge of G. We now subdivide our cases further. We specify a series of shrinking intervals $[(1-\epsilon)^r, (1-\epsilon)^{r-1}]$ for integer powers r, and use our oracle to guess which intervals contains the effective cost of the heaviest uncontracted edge in G_i. Recall also that $\epsilon = .2$ from the previous section. We know that μ^* is in one of these boxes since the effective cost of each edge is at most 1 by our contraction of heavy edges. Since each interval has ratio $(1-\epsilon)$ between its endpoints, we can use the "rounding" algorithm of the previous section to find the optimum μ^* in $O(n^2)$ time.

8.2.3 *Bounding the Number of Guesses*

So far we have defined an infinite space of intervals for all integer r, which makes guessing difficult. But we now argue that we can truncate our list of intervals at $R = O(\epsilon^{-1}\log(km/\epsilon))$. We capture the entire remaining space of μ_i in a the single interval $[0, (1-\epsilon)^R] = [0, \epsilon/km]$. At this value of R, the effective cost of every (uncontracted) edge in G_i is ϵ/km. It follows that varying μ_i anywhere in this interval changes any cut by a total value of $m \cdot \epsilon/km = \epsilon/k$, which means that even doing it for all k intervals changes any cut by a total value of ϵ.

We now reconsider the box algorithm of the previous section. We used the fact that moving within intervals $[(1-\epsilon)^r, (1-\epsilon)^{r-1}]$ does not change the value of any cut by more than a $(1\pm\epsilon)$ relative factor, so that we can find all important cuts as near-minimum cuts. Our new argument shows that in the "tail" interval $[0, (1-\epsilon)^R]$, the minimum cut changes by at most ϵ in *absolute* terms. Since we have constrained the minimum cut to be 1, this means that moving within the tail intervals also does not change the minimum cut by more than an ϵ relative factor, so our box algorithm works in these intervals as well.

We have now ensured a bound on the number of boxes. Given our choice of pivots, we need to specify an appropriate r between 0 and R for each parameter's interval, yielding an overall total of $O(\log^k km)$ boxes.

How many choices of pivots are there? A naive answer is m per parameter since we can choose any pivot edge. But there is a better answer. When we choose a pivot, we contract all edges of greater cost. But consider contracting edges from most to least expensive. As we consider more edges, only n of them cause a change in the graph. Because each contraction reduces the number of vertices by 1, all but n of edges in G_i will be contracted at the point we consider them as pivots. We can skip these contracted edges—instead, the only pivots we need to consider are the n edges of the maximum spanning tree of each G_i. In other words, there are only n^k choices of pivots.

Putting this all together, we take n^k choices of pivots. For each choice, we consider $O(\log^k km)$ boxes and, in each, enumerate all 1.44-minimum cuts, solve a linear program in $2^{O(\sqrt{k})}n^2$ time, and test the validity of the resulting solution in $\tilde{O}(m)$ time. Taken all together, this gives us a runtime of $2^{O(\sqrt{k})}n^2(n\log km)^k = n^{k+2+o(1)}$ time for finding all n^{k+1} minimish cuts with k parameters. This nearly matches the $O(mn^{1+k})$ time bound of Ellipsoid, and is strongly polynomial.

8.3 Algorithm Summary and Discussion

With these two sections on faster algorithms algorithms, we have given bounds of $O(n^3\sqrt{m}\log^2 n)$ for enumerating 2 parameter minimish cuts, $O(n^4\log^2 n)$ for enumerating 3-parameter minimish cuts, $O(n^5\log^2)$ for 4 parameters, and $\tilde{O}(k^2mn^{1+k})$ for enumerating all k-parameter minimish cuts when $k \geq 5$. For $k \geq 5$ we actually produce a superset of *candidate* minimish cuts in $O(n^{k+1}\log n)$ time which is near optimal given our counting bound; however, our enumeration bound is dominated by the hard work of constructing the convex hull of the candidates in order to decide which truly are minimish. For $k \leq 4$ the convex hull work is negligible.

For the 2-parameter problem, our $O(n^3\sqrt{m}\log n)$ time bound to find the minimish cuts is a factor $\sqrt{m}$ larger than our $O(n^3)$ bounds on the number of cuts. Why this odd outlier? For 3 or more parameters, the at least n^2 interleavings we must consider for one set of permutations means that the at least constant work per interleaving dominates the m work we must spend to perform contractions according to an interleaving. For 2 parameters, with only n cuts per interleaving, this domination is incomplete, and the contraction work takes us beyond the counting bound.

We are thus in a strange situation where for 3 and 4 parameters only, we can enumerate all minimish cuts in time which is only a polylogarithmic factor times our bound on their number—a nearly optimal result. For 2 or $k \geq 5$ parameters, we seem far less efficient.

Although the original Contraction Algorithm [Kar93] used $O(mn^2)$ time to enumerate all min-cuts, the Recursive Contraction Algorithm [KS96] was able to combine work from the many iterations of contraction to reduce the overall runtime to $\tilde{O}(n^2)$. It is an interesting open problem to achieve the same speedup for our 2-parameter minimish cut algorithm. The problem is that we need to explore all possible interleavings, which interferes with the Recursive Contraction Algorithm's re-use of partial contractions at high levels of the recursion.

For $k \geq 4$, the dominant work is winnowing. Although improving general convex hull seems difficult, our separator-based linear program shows that we can exploit the special structure of our particular problem. Perhaps this approach can be refined further. In particular, it may be possible to combine the Ellipsoid technique with our strongly polynomial algorithm to get one that is both fast and strongly polynomial.

9. DISCUSSION AND OPEN PROBLEMS

There are three obvious points of improvement in our results. First is the gap between the lower bound $\Omega(n^{k/2})$ and the upper bound $O(n^{k+1})$ on minimish cuts with k-parameter linear costs. The upper bound feels very natural so we conjecture that the lower bound can be improved. Second and third are the gaps between our bound on the number of cuts and the time to enumerate them. We are in the odd state of having an apparently optimal algorithm *only* for the 3- and 4-parameter problems. For 2 and $k \geq 5$ parameters there are significant gaps.

For 2 parameters the excess factor of $\sqrt{m}$ in the time to enumerate 2-parameter minimish cuts, $\tilde{O}(n^3\sqrt{m})$, as compared to the number of minimish cuts, $O(n^3)$, arises from the difficulty of interleaving two permutations efficiently. Once given the $O(n^3 \log n)$ candidates from the interleavings, we can winnow them in linear time. Thus, we seek a better interleaving algorithm.

For $k \geq 5$ parameters the gap between the $O(n^{k+1})$ counting bound and the $O(k^2mn^{k+1})$ enumeration bound is driven entirely by the need to determine which in a set of candidates is actually minimal. We are already leveraging the origins of these vectors as graph cuts to solve the this problem quite quickly using the Ellipsoid Algorithm; perhaps even more could be achieved.

Aissi et al. prove that their bound of $m^k\binom{n}{2}$ holds over the entire parameter space for which $c(e) \geq 0$. We require more: our use of the Contraction Algorithm separately with each $c_i(e)$ is only meaningful when *each* $c_i(e) \geq 0$ and *each* $\mu_i \geq 0$. It is an interesting open question whether our bounds can be extended to the case of negative cost functions, or whether negative cost functions are truly more complex. Note that in general the cost space considered by Aissi et al. is a $k-1$-dimensional affine subspace of the m-dimensional space of edge costs, intersected with the positive orthant. For Aissi, any basis of k cost functions yields the same cost space and the same bounds. We on the other hand require expressing the parametric cost as a *convex combination* of nonnegative *vertices* of this polytope in order for our results to hold. For $k = 2$ (the cost space is a line segment) we can always find 2 vertices to serve this purpose (the endpoints of the segment). However, in higher dimension, the number of vertices of the polytope can be much larger than k, suggesting cases where our stronger bounds will not apply.

We considered the problem of linearly *combining* cost functions. A related problem is to meet a bound on each cost function *separately.* This is known as the *multicritereon* min-cut problem. Armon and Zwick [AZ06] gave an $O(mn^{2k})$ time algorithm for the multicriterion *feasibility* problem: finding a cut that meets specific bounds on each criterion separately.

One can also consider enumeration of all multicriterion optima. Given the k cost functions, each cut corresponds to a set of k distinct costs, yielding a k-coordinate costs. The set of all cuts defines a point set. Our work (and that of Aissi) explore the *lower envelope* of this set—the interesting part of its convex hull. Any points on this envelope achieves optimality for some multicriterion objective. However, there are also *Pareto optimal* cuts on the *interior* of this set, which do not optimize any linear combination of costs but nevertheless are *non-dominated*: there is no other cut better than the given cut on all costs. They are thus multicriteria optima for certain cost bounds. Aissi et al. give bounds on the number of non-dominated cuts when there are two cost functions, but leave open the question of counting and enumerating such non-dominated cuts in general.

It is interesting to note that all the previous results on multicriteria and parametric minimum cuts [Mul99, AZ06, AMMQ15] relied on the fact that there are a small number of (single-criterion) *approximately* min-cuts. Our counting proof (for minimish cuts) is the first that does not make direct use of this fact. To bound (exact) minimish cuts, it uses only the bound on (exact) minimum cuts. It does, however, do so using the Contraction Algorithm which is the source of the result on approximately min-cuts used by the other work.

Mulmuley, in his prior work [Mul99] on min-cuts that are parametrized linearly by a single variable relied like we did on a connection between min-cut and minimum spanning trees; however, he explored it only for the case of low bit complexity and a single parameter and derived weaker bounds. Our work suggests that a deeper look at parametric minimum spanning trees might also shed further light on parametric minimum cuts.

We studied multi-parametric linear functions, and also gave a result on single parameter nonlinear parametric functions. It is natural to explore the combination, of multi-parametric nonlinear functions. Again, the majority of the work seems to center on understanding the behavior of nonlinear multi-parametric spanning trees.

10. ACKNOWLEDGMENTS

Thanks to the anonymous referee for some very helpful comments on the article, and to Jonathan Kelner for some emergency help with linear programming.

11. REFERENCES

[AMMQ15] Hassene Aissi, A.Ridha Mahjoub, S.Thomas McCormick, and Maurice Queyranne. Strongly polynomial bounds for multiobjective and parametric global minimum cuts in graphs and hypergraphs. *Mathematical Programming*, pages 1–26, 2015.

[ASS89] Pankaj Agarwal, Micha Sharir, and Peter Shor. Sharp upper and lower bounds on the length of general davenport-schinzel sequences. *Journal of Combinatorial Theory, Series A*, 52(2):228–274, 1989.

[AZ06] Amitai Armon and Uri Zwick. Multicriteria global minimum cuts. *Algorithmica*, 46(1):15–26, 2006.

[dBSvKO00] Mark de Berg, Otfried Schwarzkopf, Mark van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.

[DKL76] Efim A. Dinitz, Alexander V. Karzanov, and Micael V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Nauka Publishers, Moscow, 1976.

[Fel68] William Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley & Sons, New York, third edition, 1968.

[GLS88] Martin Grötschel, Làszló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1988.

[HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001.

[HZ15] Thomas Dueholm Hansen and Uri Zwick. An improved version of the random-facet pivoting rule for the simplex algorithm. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 209–218, New York, NY, USA, 2015. ACM.

[Kar93] David R. Karger. Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm. In *Proceedings of the* 4^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30. ACM-SIAM, January 1993. This work was merged with later work into Journal of the ACM43(4).

[Kar94] David R. Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, Stanford, CA 94305, 1994.

[Kar98] David R. Karger. Random sampling and greedy sparsification in matroid optimization problems. *Mathematical Programming B*, 82(1–2):41–81, June 1998. A preliminary version appeared in Proceedings of the 34^{th} Annual Symposium on the Foundations of Computer Science.

[Kar00] David R. Karger. Minimum cuts in near-linear time. *Journal of the ACM*, 47(1):46–76, January 2000. A preliminary version appeared in Proceedings of the 28^{th} ACM Symposium on Theory of Computing.

[KK15] Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Conference on Innovations in Theoretical Computer Science*, pages 367–376. ACM, 2015.

[KKT95] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, March 1995.

[KS93] David R. Karger and Clifford Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In Alok Aggarwal, editor, *Proceedings of the* 25^{th} *ACM Symposium on Theory of Computing*, pages 757–765. ACM, ACM Press, May 1993. Journal version appears in Journal of the ACM 43(4).

[KS96] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 43(4):601–640, July 1996. Preliminary portions appeared in SODA 1992 and STOC 1993.

[Lom94] Micael V. Lomonosov. On Monte Carlo estimates in network reliability. *Probability in the Engineering and Informational Sciences*, 8:245–264, 1994.

[Mul99] Ketan Mulmuley. Lower bounds in a parallel model without bit operations. *SIAM Journal on Computing*, 28(4):1460–1509, 1999.

[SA95] Micha Sharir and Pankaj Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.

[ST86] Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, July 1986.

[Tar83] Robert E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1983.