10-2011

Algorithms for the Generalized Sorting Problem

Zhiyi Huang
University of Pennsylvania

Sampath Kannan
University of Pennsylvania, kannan@cis.upenn.edu

Sanjeev Khanna
University of Pennsylvania, sanjeev@cis.upenn.edu

Huang, Z., Kannan, S., & Khanna, S., Algorithms for the Generalized Sorting Problem, IEEE 52nd Annual Symposium on Foundations of Computer Science, Oct. 2010, doi: 10.1109/FOCS.2011.54

Algorithms for the Generalized Sorting Problem

Abstract
We study the generalized sorting problem where we are given a set of n elements to be sorted but only a subset of all possible pairwise element comparisons is allowed. The goal is to determine the sorted order using the smallest possible number of allowed comparisons. The generalized sorting problem may be equivalently viewed as follows. Given an undirected graph G(V,E) where V is the set of elements to be sorted and E defines the set of allowed comparisons, adaptively find the smallest subset E' E of edges to probe such that the directed graph induced by E' contains a Hamiltonian path. When G is a complete graph, we get the standard sorting problem, and it is well-known that Θ(n log n) comparisons are necessary and sufficient. An extensively studied special case of the generalized sorting problem is the nuts and bolts problem where the allowed comparison graph is a complete bipartite graph between two equal-size sets. It is known that for this special case also, there is a deterministic algorithm that sorts using Θ(n log n) comparisons. However, when the allowed comparison graph is arbitrary, to our knowledge, no bound better than the trivial O(n2) bound is known. Our main result is a randomized algorithm that sorts any allowed comparison graph using Õ(n3⁄2) comparisons with high probability (provided the input is sortable). We also study the sorting problem in randomly generated allowed comparison graphs, and show that when the edge probability is p, Õ(min{n/p2 , n3⁄2p √p}) comparisons suffice on average to sort.

Disciplines
Computer Sciences

Comments
Huang, Z., Kannan, S., & Khanna, S., Algorithms for the Generalized Sorting Problem, IEEE 52nd Annual Symposium on Foundations of Computer Science, Oct. 2010, doi: 10.1109/FOCS.2011.54

Algorithms for the Generalized Sorting Problem

Zhiyi Huang*
Dept. of Comp. and Inf. Science
University of Pennsylvania
Philadelphia, USA
Email: hzhiyi@cis.upenn.edu

Sampath Kannan†
Dept. of Comp. and Inf. Science
University of Pennsylvania
Philadelphia, USA
Email: kannan@cis.upenn.edu

Sanjeev Khanna‡
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA
Email: sanjeev@cis.upenn.edu

Abstract— We study the *generalized sorting* problem where we are given a set of n elements to be sorted but only a subset of all possible pairwise element comparisons is allowed. The goal is to determine the sorted order using the smallest possible number of allowed comparisons. The generalized sorting problem may be equivalently viewed as follows. Given an undirected graph $G(V, E)$ where V is the set of elements to be sorted and E defines the set of allowed comparisons, adaptively find the smallest subset $E' \subseteq E$ of edges to probe such that the directed graph induced by E' contains a Hamiltonian path.

When G is a complete graph, we get the standard sorting problem, and it is well-known that $\Theta(n \log n)$ comparisons are necessary and sufficient. An extensively studied special case of the generalized sorting problem is the nuts and bolts problem where the allowed comparison graph is a complete bipartite graph between two equal-size sets. It is known that for this special case also, there is a deterministic algorithm that sorts using $\Theta(n \log n)$ comparisons. However, when the allowed comparison graph is arbitrary, to our knowledge, no bound better than the trivial $O(n^2)$ bound is known. Our main result is a randomized algorithm that sorts any allowed comparison graph using $\widetilde{O}(n^{3/2})$ comparisons with high probability (provided the input is sortable). We also study the sorting problem in randomly generated allowed comparison graphs, and show that when the edge probability is p, $\widetilde{O}(\min\{\frac{n}{p^2}, n^{3/2}\sqrt{p}\})$ comparisons suffice on average to sort.

Keywords-algorithm; comparison-sort; sorting.

1. Introduction

Sorting is a problem of central importance both in theoretical and practical computing. While the complexity of sorting is very well understood in the standard model, recently there has been a lot of interest in models where the cost of comparisons between elements is not uniform. One can regard the elements to be sorted as nodes in a graph and pairs of comparable elements as adjacent. The problem of matching nuts and bolts [12] corresponds to the case where this graph is a complete bipartite graph. Charikar et al. [3] introduced the general problem of finding query strategies for priced information. The model here is that each probe of the data has an associated cost and our goal is to evaluate a function with minimum total cost. In the context of sorting and selection, the model is that comparisons have known costs associated with them and we want to minimize the total cost of the comparisons used to solve the problem. The goal is to design algorithms whose total cost is competitive with the cost of the cheapest *certificate* that proves the correctness of the solution. Unfortunately, it is known [3] that if comparison costs are arbitrary, then this competitive ratio can be arbitrarily bad for sorting. Subsequent papers [7], [9] consider more structured costs and achieve better competitive ratios.

In this paper we consider a very natural model that can be viewed as a special case of the non-uniform cost model of [3] but incomparable to the structured cost models of [7], [9]. Our model is also a natural generalization of the nuts and bolts model. We assume that only a subset of the comparisons are allowed and each allowed comparison has unit cost. Thus we could view this as the non-uniform model where some comparisons have cost 1 while others have cost ∞. The set of allowable comparisons is given to us as an undirected graph G. The algorithm may probe any edge and find out the direction of that edge. In other words, it may compare any two elements that are comparable. Importantly, we are promised that if we probe all edges, we would discover a total order on the nodes, i.e. a directed Hamilton path with all other edge directions implied by transitivity.

As a simple motivating scenario, consider an academic department that is deciding between candidates for a position based on letters of recommendation. Two candidates are comparable if there is a letter writer who is competent to compare them. A probe involves asking a letter writer for a comparison. Of course it is not clear if enough comparisons can be performed to lead to a total order, or even whether they will be consistent and lead to a partial order. However, scenarios like this are common in a number of settings - websites that seek to rank the hotels or restaurants in a city, movie ratings, committee reviews of proposals or papers, etc.

Prior to this paper no deterministic or randomized algorithm was known for this problem that uses a sub-quadratic number of probes. To understand the difficulty we face here it is useful to examine the techniques used in solving the related problems mentioned above and understand why they do not seem to work for our problem.

*Supported in part by ONR Award N000140710907.
†Supported in part by NSF Award CCF 1137084.
‡ Supported in part by NSF Awards CCF-0635084 and IIS-0904314.

Alon et al. [2] were the first to give a sub-quadratic deterministic algorithm for the nuts and bolts problem. Their $O(n \log^4 n)$ algorithm (using $O(n \log^3 n)$ comparisons) builds a number of expanders, performs comparisons corresponding to the edges of these expanders, and proves running time bounds using properties of these expanders. A sequence of improvements culminated in a paper by Komlos et al. [10] that showed that the problem could be solved in time $O(n \log n)$ using ideas from Ajtai-Komlos-Szemeredi sorting networks [1]. These papers exploit the fact that the graph of allowed comparisons is a complete bipartite graph and hence there exist expander subgraphs of this graph with arbitrary subsets of the vertices from the two parts of the bipartite graph. Unlike the nuts and bolts problem, since the graph here is adversarially chosen, it may not have large expander subgraphs, and even if it does, it is not clear whether the comparisons in this subgraph are useful in making progress, since an adversary could potentially ensure that any bipartite expanders have vertices of widely different ranks on the left and right sides. Without the promise that the allowed comparisons induce a total order, we can see that discovering all the order relationships will take $\Omega(n^2)$ probes, for example, when the directed graph being discovered is a dense bipartite graph with all edges going from left to right. This immediately rules out a MergeSort-like approach since, if we partition the nodes arbitrarily, the promise will typically not hold in the subproblems. Similarly, approaches like QuickSort prove difficult to emulate since no (good) pivot element might be comparable to all other elements. For a pivot x and an element y that is not directly comparable to x, it is not clear how one can discover the relationship between x and y with a few probes.

The structured cost models assume that the cost of a comparison is a monotone function of the known 'size' of the elements being compared. Unfortunately this crucial assumption is invalid in our model where the comparison graph can be arbitrary.

If all comparisons were allowed, we know that given any partial order there is a comparison that will reduce the number of possible linear extensions by a constant factor, no matter what its outcome. We will quantify this idea later and call such comparisons *balanced*. However, in our problem it is possible that at some stage of the algorithm there are no balanced comparisons and every edge has an *assumed* direction, meaning, 'most' surviving linear extensions are compatible with this direction of the edge.

In this paper, we introduce several new ideas for sorting and show the following results:

- For an arbitrary graph of comparisons with the promise of a directed Hamilton path, we can sort with $\widetilde{O}(n^{3/2})$ probes with high probability.
- On random graphs with edge probability p, we can sort with $\widetilde{O}(\min\{\frac{n}{p^2}, n^{3/2}\sqrt{p}\})$ probes with high probability, where the probability is over the input and the random choices of the algorithm. Thus the average probe complexity in the random graph model is $\widetilde{O}(n^{7/5})$ for any choice of p. In particular, when the density p is $\Omega(1)$ we get a near optimal algorithm that uses $O(n \log^3 n)$ probes.

We prove the first result by setting up several potential functions and arguing that we can decrease at least one of them at a good rate at all times. Our results for random graphs build on our ideas for the arbitrary graph case as well as utilize a QuickSort-like pivoting scheme when the random graph is sufficiently dense. The fact that random graphs may be easier than arbitrary graphs suggests that careful combinatorial constructions may be needed to improve the lower bound beyond $\Omega(n \log n)$ probes. Although we are primarily interested in the number of probes, all of our algorithms above run in polynomial time.

Organization: Section 2 formally defines the problem and introduces some key concepts used in our algorithms. We present in Section 3 our main result, namely, a randomized algorithm to sort any allowed comparison graph using $\widetilde{O}(n^{3/2})$ comparisons. Section 4 presents our algorithms for random graphs. We conclude with some remarks in Section 5.

2. Preliminaries

The input to the generalized sorting problem is an undirected graph $G(V, E)$ where an edge $(u, v) \in E$ indicates that the element u can be compared to the element v. By probing (u, v), we reveal a directed edge (u, v) or (v, u), depending on whether $u < v$ or $v < u$. Let $\vec{G}(V, \vec{E})$ denote the directed graph where $(u, v) \in \vec{E}$ iff $(u, v) \in E$, and $u < v$. Given the promise that $\vec{G}(V, \vec{E})$ is an acyclic graph that contains a directed Hamiltonian path, the problem is to find this path by adaptively probing the smallest number of edges in E.

When G is the *complete graph*, then this becomes the regular sorting problem, and $\Theta(n \log n)$ probes are necessary and sufficient to reveal the underlying Hamiltonian path. If G is the *complete bipartite graph* with $n/2$ vertices on each side, then this becomes the problem of *matching nuts and bolts*. Again, $\Theta(n \log n)$ probes are necessary and sufficient as mentioned earlier.

2.1. Linear Extensions and Average Rank

We let Π denote the set of bijections $\pi : V \mapsto [n]$. Each bijection $\pi \in \Pi$ specifies a total order $(V, <_\pi)$: For every pair of vertices u and v, $u <_\pi v$ if $\pi(u) < \pi(v)$ and vice versa. For every bijection $\pi \in \Pi$ and every vertex $v \in V$, we refer to $\pi(v)$ as the *rank* of v with respect to π.

We will let $\widetilde{E}$ denote the set of directed probed edges. The edges in $\widetilde{E}$ define a partial order $(V, <_{\widetilde{E}})$: For every pair of vertices u and v, $u <_{\widetilde{E}} v$ if and only if there exists a directed path from u to v in $\widetilde{G} = (V, \widetilde{E})$. A *linear extension*

of $\widetilde{E}$ is a bijection $\pi \in \Pi$ such that $(V, <_\pi)$ is consistent with the partial order $(V, <_{\widetilde{E}})$. We let $\Pi_{\widetilde{E}}$ denote the set of linear extensions of $\widetilde{E}$. Note that this definition includes linear extensions some of whose "successor edges" may not exist in G. For every vertex v, we let $\pi_{\widetilde{E}}(v)$ denote the *average rank* of v among all linear extensions of $\widetilde{E}$, that is, $\pi_{\widetilde{E}}(v) \stackrel{\text{def}}{=} \sum_{\pi \in \Pi_{\widetilde{E}}} \pi(v) / \left|\Pi_{\widetilde{E}}\right|$.

For every pair of vertices u and v, we will let $\Pi_{\widetilde{E}}(u < v)$ denote the set of linear extensions π of $\widetilde{E}$ such that $u <_\pi v$. It is easy to see that $\Pi_{\widetilde{E}}(u < v) = \Pi_{\widetilde{E} \cup (u,v)}$.

2.2. Balanced Edges and Assumed Directions

We will use the lemma below that states that if two vertices u and v have similar average rank, then a non-trivial fraction of the linear extensions will reverse the order indicated by the average rank. This is a natural generalization of the result in [8] and can be proved by the same technique. The proof is deferred to Appendix A.

Lemma 2.1: If $\pi_{\widetilde{E}}(v) - \pi_{\widetilde{E}}(u) \leq c \geq 0$, then $\frac{\left|\Pi_{\widetilde{E}}(v<u)\right|}{\left|\Pi_{\widetilde{E}}\right|} > \frac{1}{e^{c+1}}$.

Balanced edges: An unprobed edge is *balanced* if probing the edge would reduce the number of linear extensions by at least a $\left(1 - \frac{1}{e\sqrt{n}}\right)$ factor, regardless of the underlying direction of the edge. The following result is a simple corollary of Lemma 2.1.

Corollary 2.2: If $\left|\pi_{\widetilde{E}}(v) - \pi_{\widetilde{E}}(u)\right| \leq \ln n/2$ for some unprobed edge $\{u, v\} \in E$, then $\{u, v\}$ is a balanced edge.

Suppose there always exists a balanced edge. Then, since the total number of linear extensions is initially $n!$ and probing a balanced edge reduces the number of linear extensions by at least a $\left(1 - \frac{1}{e\sqrt{n}}\right)$ factor, we only need to probe $O(n^{3/2} \log n)$ edges to reduce the number of linear extensions to one, and hence sort the n elements.

Assumed directions: Suppose at some point we reach a stage in which for every edge $\{u, v\} \in E$ with unknown direction, either $\pi_{\widetilde{E}}(v) < \pi_{\widetilde{E}}(u) - \ln n/2$ or $\pi_{\widetilde{E}}(u) < \pi_{\widetilde{E}}(v) - \ln n/2$. We will assign an assumed direction to each unprobed edge (u, v): in the first case (when the average rank of v is much smaller than that of u), the assumed direction is (v, u), and it is (u, v) otherwise. We let $\widehat{G} = (V, \widehat{E})$ denote the directed graph in which each edge with unknown direction is replaced by an edge with the assumed direction. (Directions of edges with known directions are preserved in $\widehat{G}$.)

Inversions: If the true direction of an unprobed edge (u, v) contradicts the assumed direction, then we call such edge an *inversion*. By letting $c = 0$ in Lemma 2.1, we get that finding an inversion reduces the number of linear extensions by at least a $\frac{1}{e}$ factor.

3. Generalized Sorting in $\widetilde{O}(n^{3/2})$ Comparisons

At a high-level, the algorithm proceeds by maintaining at all times an assumed direction graph $\widehat{G}$ that indicates for each edge $(u, v) \in E$, the likely relationship between u and v. Then at each step, the algorithm either identifies an edge $(u, v) \in \widehat{E}$ to probe, or it identifies a set of $O(\sqrt{n})$ vertices for which the direction of all unprobed incident edges can be confirmed at a total cost of $O(n \log n)$. Clearly, the total cost incurred over all steps in the latter case is $O(n^{3/2} \log n)$. In the former case, if probing of the edge (u, v) reveals an inversion, then once again the total cost incurred over all such steps can be bounded by $O(n^{3/2} \log n)$. However, not every probed edge may reveal an inversion. We handle this by showing that at each step, the edge (u, v) can be chosen such that over all iterations, the number of probed edges that do not reveal an inversion is bounded by $O(n^{3/2} \log n)$.

We now describe the algorithm in detail. We will consider a constant number of budgets of $O(n^{3/2} \log n)$ probes each. Then, every probe in our approach will be charged to one of these budgets. Hence, the total number of probes is at most $O(n^{3/2} \log n)$.

3.1. Potentials and Budgets

We will consider the following potentials:

- Φ_V: Number of vertices with at least one incident edge with unknown direction.
  We say a vertex *exhausted* if all its incident edges have known directions and is *live* otherwise.
- Φ_{LE}: Logarithm of number of feasible linear extensions.

We will consider the following budgets, each of which is of size $O(n^{3/2} \log n)$.

- Static Budgets:
  - B_{init} : Budget for sampling at most $O(n^{3/2} \log n)$ edges initially to estimate the in-degree of each vertex in $\vec{G}(V, \vec{E})$ to within an $\sqrt{n}$ additive error.
  - B_{in} : Budgets for probing at most $O(\sqrt{n})$ incoming edges for each vertex.
- Dynamic Budgets:
  - B_V : Budget for decreasing potential Φ_V at a cost of $O(\sqrt{n} \log n)$ per unit decrease.
  - B_{LE} : Budget for decreasing Φ_{LE} at a cost of $O(\sqrt{n})$ per unit of decrease.

Note that $\Phi_V = n$ at the beginning, and $\Phi_V = 0$ at the end, the total charge to this budget does not exceed $O(n^{3/2} \log n)$. Similarly, $\Phi_{LE} = O(n \log n)$ at the beginning (all $n!$ total orders are feasible), and $\Phi_{LE} = 0$ at the end, the total charge to this budget does not exceed $O(n^{3/2} \log n)$.

3.2. Key Subroutines

We will first introduce several key subroutines that will serve as the building blocks for our algorithm.

3.2.1. Estimating the Average Ranks: The first subroutine aims to compute the approximate average ranks of the vertices efficiently. Recall that an edge has an assumed direction only when the expected rank of the two vertices are well-separated. In this case, the well-approximated average ranks suffice to imply the assumed direction. Hence, we can indeed compute the assumed directions of the edges.

We now describe a subroutine to compute average ranks efficiently.

1: Sample $N = 2(n+1)^2 \ln n$ points $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N$ in the order polytope independently and uniformly at random (see [4] for a poly-time implementation of this sampling step.).
2: Compute the average

$$\bar{\boldsymbol{x}} = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}^i \ .$$

3: Let $(n+1)\bar{x}_v$ be the approximate average rank of vertex v for every $v \in V$.

Figure 1. Algorithm for computing average ranks

Lemma 3.1: With probability at least $1 - \frac{1}{n^2}$, $(n+1)\bar{x}_v$ in the procedure in Figure 1 approximates the average rank of each vertex v up to an additive error of 1 for all $v \in V$.

Proof: For each $1 \leq i \leq N$ and $v \in V$, x_v^i is a random variable with value range $[0,1]$ and expectation $\pi_{\widetilde{E}}(v)$. So by Chernoff-Höeffding bound (see [11], for instance) we get that $\Pr\left[\left|\sum_{i=1}^N x_v^i - N\pi_{\widetilde{E}}(v)\right| \geq \frac{N}{(n+1)}\right] \leq 2\exp\left(-2\frac{N}{(n+1)^2}\right) = 2\exp\left(-4\ln n\right) < \frac{1}{n^3}$.

Note that if $\left|\sum_{i=1}^N x_v^i - N\pi_{\widetilde{E}}(v)\right| \leq \frac{N}{(n+1)}$, then $\left|(n+1)\bar{x}_v - \pi_{\widetilde{E}}(v)\right| \leq 1$ By union bound, with probability at least $1 - \frac{1}{n^2}$, this sampling error bound holds for all $v \in V$. ∎

We highlight below a useful property of the vector $\bar{\boldsymbol{x}}$.

Lemma 3.2: With probability at least $1 - \frac{1}{n^2}$, the estimated average ranks $(n+1)\bar{x}_v$ have the property that for each pair of vertices u and v, if $\bar{x}_v \geq \bar{x}_u$ then $\frac{\left|\Pi_{\widetilde{E}}(v>u)\right|}{\left|\Pi_{\widetilde{E}}\right|} \geq \frac{1}{e^3}$.

Proof: By Lemma 3.1, with probability at least $1 - \frac{1}{n^2}$, we have that $\left|(n+1)\bar{x}_v - \pi_{\widetilde{E}}(v)\right| \leq 1$ for all $v \in V$. In this case, $\bar{x}_v \geq \bar{x}_u$ implies that $\pi_{\widetilde{E}}(v) \geq \pi_{\widetilde{E}}(u) - 2$. Lemma 2.1 then implies the corollary. ∎

3.2.2. Initial Sampling: Estimating the In-Degrees: We say a vertex v becomes *active* if the number of unverified in-edges to v is at most $4\sqrt{n}\log n$. We now give a sampling scheme that uses $O(n^{3/2}\log n)$ probes to estimate the in-degree of each vertex in $\vec{G}(V,\vec{E})$ to within an additive error of $3\sqrt{n}\log n$. This will help determine the set of active vertices. After a vertex v becomes active, we can safely probe any incoming-edge incident to v in $\widehat{G}$ because we are in a win-win situation: Either the we find an inversion, in which case we decrease Φ_{LE} at a good rate, or we verify one of the at most $4\sqrt{n}\log n$ unverified in-edges of the vertex and charge that probe to B_{in}. (The algorithm below is applied when there are no balanced edges since as long as there are balanced edges we can probe them and reduce Φ_{LE} at a good rate.)

1: **while** there is a vertex $v \in V$ whose in-degree is not estimated **do**
2: Estimate the average ranks and compute the assumed direction graph $\widehat{G}$.
3: Sample the edges incident to v independently with probability $\frac{1}{\sqrt{n}}$.
4: **if** there are no inversions among the sampled edges **then**
5: Let v's estimated in-degree $\bar{d}_v^-$ be its in-degree in $\widehat{G}$.
6: **end if**
7: **end while**

Figure 2. Algorithm INIT-SAMPLE

Lemma 3.3: With high probability, we can get estimates, $\bar{d}_v^-$, for the in-degrees such that $\left|\bar{d}_v^- - \vec{d}_v^-\right| \leq 3\sqrt{n}\log n$ for every $v \in V$ in $O(n^{3/2}\log n)$ probes.

Proof: We will show that if the estimated expected ranks are within the desired additive error every time, which holds with high probability, then we will obtain the estimated in-degree with the desired accuracy and within $O(n^{3/2}\log n)$ probes with high probability.

Inside each while loop, we claim that if there are at least $3\sqrt{n}\log n$ inversions among the edges incident to v, then the sampling will catch at least one inversion with probability at least $1 - \frac{1}{n^3}$. This is because the probability that none of the $3\sqrt{n}\log n$ inversions is sampled is $\left(1 - \frac{1}{\sqrt{n}}\right)^{3\sqrt{n}\log n} < \frac{1}{n^3}$.

We call a while loop iteration *bad* if there are at least $3\sqrt{n}\log n$ inversions and yet the sampling does not catch any of them. By union bound, with probability at least $1 - \frac{1}{n}$ there are no bad iterations among the first $O(n\log n)$ iterations of the while loop.

Finally, we argue that if there are no bad iterations within the first $O(n\log n)$ iterations of the while loop, then the algorithm will provide estimated in-degrees that satisfy the desired accuracy requirement. Inside each while loop, two things may happen. First, we may find an inversion with at most $\sqrt{n}$ probes. In this case we will charge these probes to B_{LE}. Since finding an inversion will decrease Φ_{LE} by a constant, it cannot happen more than $O(n\log n)$ times. Second, we do not find any inversion. By our assumption that this is not a bad loop, we conclude that there are at

most $3\sqrt{n}\log n$ inversions among the in/out-edges. In this case, using the in-degree in $\widehat{G}$ as our estimated in-degree d_v^- satisfies the error bound stated in the lemma. ■

Remark 1: The sampling algorithm above works for any DAG (not necessarily sortable), and implies the following result, which may be of independent interest. Suppose we are given a graph where the edge directions are not revealed, and we want to estimate the in-degrees of the vertices in the underlying directed graph to within an additive error of $\sqrt{n}$. If the directed graph is arbitrary, then arguably the best we can do is uniform sampling which requires sampling at rate $\Omega(1)$, and hence uses $\Omega(n^2)$ probes. However, if the underlying graph is a DAG, then this algorithm illustrates how to sample more efficiently with $\widetilde{O}(n^{3/2})$ probes.

Remark 2: If the maximum degree is $d_{\max}$, then we may sample edges with probability $\frac{4}{\sqrt{d_{\max}}}$ to obtain the estimated in-degree of every vertex within an additive error of $\sqrt{d_{\max}}\log n$ using $O(n\sqrt{d_{\max}}\log n)$ probes.

3.2.3. Decreasing Φ_V via Binary Search: Finally, we use a subroutine that decreases Φ_V at a good rate provided $\widehat{G}$ has a large number of live vertices with a known total order.

Input: A set $S = \{v_1 < \cdots < v_k\}$ of $k \geq \sqrt{n}$ live vertices with known total order.

1: **for** $v \in V \setminus S$ **do**
2: Let $v_{i_1}, \ldots, v_{i\ell}$ be its neighbors in S, $i_1 \leq \cdots \leq i_\ell$.
3: Let $L = 1$ and $H = \ell$.
4: **while** $H \geq L$ **do**
5: Let i_j be the median of $i_L, \ldots, i_H$. Probe the edge (v, v_{i_j}).
6: If $v < v_{i_j}$ then let $H = j - 1$; otherwise, let $L = j + 1$.
7: **end while**
8: **end for**

Figure 3. Algorithm for decreasing Φ_V via binary search

Lemma 3.4: The total number of probes in the binary search algorithm in Figure 3 is at most $O(n \log n)$, and the potential Φ_V decreases by at least $k \geq \sqrt{n}$.

3.3. The Algorithm

Now we are ready to describe our algorithm.

Recall that a vertex is *active* if the estimated number of unprobed in-edges (i.e. $\bar{d}_v^-$ minus the number of probed in-edges) is at most $4\sqrt{n}\log n$. We say that an unprobed edge (u, v) is *free* if $(u, v) \in \widehat{E}$ and v is active. The lemma below asserts that there are always some active vertices.

Lemma 3.5: The $\sqrt{n}$ lowest rank live vertices in the underlying graph $\vec{G}$ are all active.

Proof: Let S be the set of $\sqrt{n}$ lowest rank live vertices. Fix any vertex $v \in S$. Then any unprobed incoming edge to v must necessarily come from a vertex in S since every other vertex of rank smaller than v is already exhausted. Thus the unprobed in-degree of v is bounded by $\sqrt{n}$. Since we know the in-degree of v to within an additive error of $3\sqrt{n}\log n$, it follows that v must be active. ■

1: **Init:** Estimate the in-degrees $\bar{d}_v^-$, $\forall v \in V$, by the sampling subroutine.
2: **while** the vertices are not completely sorted **do**
3: **while** there is a balanced edge e **do**
4: Probe e
5: **end while**
6: Estimate average ranks and compute the assumed direction graph $\widehat{G}$.
7: **Case 1:** There exists a free edge (u, v). In this case, simply probe the edge.
8: **Case 2:** There exists a set S of at least $\sqrt{n}$ live vertices with known total order. In this case, use the binary search subroutine.
9: **end while**

Figure 4. Algorithm for sorting with restricted comparisons

The next lemma argues that the two cases we consider in the algorithm cover all possibilities.

Lemma 3.6: If there are no free edges, then there exists a set of at least $\sqrt{n}$ live vertices with known total order.

Proof: Consider the set S of $\sqrt{n}$ lowest rank, live vertices, say, $v_1, \ldots, v_{\sqrt{n}}$, in the underlying graph $\vec{G}$. By Lemma 3.5, these vertices are active. If there are no free edges, we argue that the total order among vertices in $S = \{v_1, \ldots, v_{\sqrt{n}}\}$ is known.

For each $1 \leq i < \sqrt{n}$, there are two cases. In the first case, suppose $\pi(v_{i+1}) = \pi(v_i) + 1$, that is, v_{i+1} and v_i are two adjacent vertices on the underlying Hamiltonian path, then we have $(v_i, v_{i+1}) \in E$. Moreover, this edge is an in-edge of v_{i+1} in $\widehat{G}$. Since v_i and v_{i+1} are both active and there are no "free" edges, this edge must have already been probed, and hence $v_i < v_{i+1}$ is known. Now we turn to the case $\pi(v_{i+1}) > \pi(v_i) + 1$. By our choice of S, all vertices with rank between $\pi(v_i)$ and $\pi(v_{i+1})$ must be exhausted and hence all their incident edges have known directions. So we have a verified path that confirms the order $v_i < v_{i+1}$. ■

Finally, we will show that the algorithm uses $O(n^{3/2}\log n)$ probes.

Lemma 3.7: The total number of probes used by the algorithm in Figure 4 is $O(n^{3/2}\log n)$.

Proof: The number of probes used in the initial sampling step is at most $O(n^{3/2}\log n)$ according to Lemma 3.3. We will charge these probes to B_{init} and B_{LE} as we discussed in Lemma 3.3. Probes in line 4 are charged to B_{LE}. Each probe used in case 1 is charged to B_{in} if the edge is not inverted and to B_{LE} otherwise. By our definition

of "free" edges and inversions, the total number of probes of this type does not exceed $O(n^{3/2}\log n)$. The probes used in case 2 are charged to B_V. By Lemma 3.4, there are at most $O(n^{3/2}\log n)$ such probes. In sum, the total number of probes used by the algorithm is at most $O(n^{3/2}\log n)$. ∎

Theorem 3.8: There is a poly-time algorithm that reveals with high probability the underlying Hamiltonian path for any graph G within $O(n^{3/2}\log n)$ probes.

4. Sorting in Random Comparison Graphs

In this section, we will consider a random graph model that is similar to the Erdös-Rényi model: Given a set of n vertices, first pick a random order of the vertices $v_1 < \cdots < v_n$; let $(v_i, v_{i+1}) \in \vec{E}$ for $1 \le i \le n-1$; for every other edge $e = (v_i, v_j)$, $i < j-1$, let $e \in \vec{E}$ with probability p.

We will refer to the probability p in this model as the *density* of the random graph. We will consider the probe complexity of algorithms in the fashion of average case analysis in the random graph model. The goal is to design good algorithms whose average performance, i.e. expected number of probes over random realizations of $\vec{G}$, is non-trivially better than $\widetilde{O}(n^{3/2})$.

Summary of results: Our first result is a QuickSort like algorithm, called Sort-Dense-RG, that performs well when edge density is large. Its average probe complexity is $\widetilde{O}(n/p^2)$ when the density is $p > n^{-1/3}$. Thus when $p = \Omega(1)$, the algorithm sorts uses $\widetilde{O}(n)$ probes on average. Our second result is a complementary algorithm, Sort-Sparse-RG that performs well in the sparse edge density regime. Its average probe complexity is $\widetilde{O}(n^{3/2}\sqrt{p})$. Combined together, these results show that the average probe complexity in the random graph model is $\widetilde{O}(n^{7/5})$ for any choice of p.

4.1. Algorithm Sort-Dense-RG

The algorithm is modeled after QuickSort. However, non-trivial changes are needed both to find the relationship of all elements with the pivot and to handle the base case of the recursion. The algorithm starts by picking a random vertex v as the pivot. Since the comparison graph is not complete, it is non-trivial to partition all elements with respect to the pivot. Nevertheless, we show that an adaptive sampling approach combined with special handling when only $O(\log n/p^3)$ elements remain to be partitioned, produces a complete partition. Unfortunately, these remaining elements may not be of contiguous rank and so some care is needed to ensure that the probes reveal the order between them and the pivot. We then recursively apply QuickSort to the two sets produced by this partition. Again, the base case is when the block of elements to be sorted has size $O(\log n/p^3)$. But in this case we are assured that this is a contiguous block and hence can simply apply the algorithm for general graphs to this base case, using $\widetilde{O}(p^{-9/2})$ probes.

We first introduce some notation and a lemma that summarizes some properties that are useful for our algorithm and its analysis. The proof of the lemma is a straightforward applications of the Chernoff-Höeffding bound and union bound, and hence omitted.

For any three vertices u, v, and w, we say w is *sandwiched between* u and v if $(u,w),(w,v) \in \vec{E}$.

Lemma 4.1: With high probability, for every pair of vertices u and v such that $\pi(v) - \pi(u) \ge 64\log n/p^3$, we have the following properties:

(a) There are at least $p^2(\pi(v)-\pi(u))/2$ vertices and at most $2p^2(\pi(v)-\pi(u))$ vertices sandwiched between u and v. Moreover, at least one quarter of the vertices sandwiched between u and v have rank in the range $[\frac{3}{4}\pi(u)+\frac{1}{4}\pi(v), \frac{1}{4}\pi(u)+\frac{3}{4}\pi(v)]$.

(b) Every vertex w has at least $p^3(\pi(v)-\pi(u))/2$ neighbors that are sandwiched between u and v; thus a random vertex sandwiched between u and v has a probability at least $p/4$ of being a neighbor of w.

Suppose we randomly choose $16\log n/p$ vertices that are sandwiched between u and v and probe all their incident edges. As a simple corollary of the lemma we get the following facts with high probability. For all w with rank lower than u, we have verified the order $w < v$ via one of the sandwiched vertices. Similarly, for all w with rank greater than v, we have verified the order $w > u$.

We are now ready to describe the partitioning step and the base case handling for our quicksort-like algorithm.

Partitioning Step: The heart of the algorithm is an efficient way of partitioning around a randomly chosen pivot element v whenever the instance contains at least $(256\log n)/p^3$ elements. We describe how elements smaller than v can be identified efficiently; a symmetric procedure will verify elements greater than v. The algorithm will maintain a lower marker $v^- < v$. This marker will progressively shift closer to v in rank through the stages of the algorithm as long as $\pi(v) - \pi(v^-) \ge 64\log n/p^3$. The invariant we will maintain is that all vertices lower in rank than v^- have been verified to be less than v.

Initially v^- is a virtual vertex that loses directly to all other vertices. It is vacuously true that vertices lower in rank than v^- have been verified to be less than v. We first probe every edge incident to v. Let V^- be the set of vertices sandwiched between v^- and v. Initially this is the set of all vertices that have directed edges to v. Randomly pick a set W of $16\log n/p$ vertices in V^- and probe all their incident edges. By Lemma 4.1, we have verified $u < v$ for every vertex u whose rank is less than the rank of v^- with high probability. (For the initial step where v^- is the virtual lowest-rank vertex this step will not reveal any new vertices u of lower rank than v^- that are verified to be less than v, because there are none. But this step is necessary in subsequent rounds.) Let $W' \subseteq W$ denote the set of vertices $w \in W$ such that there are at least $16\log n/p$ vertices that

are sandwiched between w and v. If $|W'| < \log n/p$, we conclude that $\pi(v) - \pi(v^-) < 64 \log n/p^3$. By Lemma 4.1 if v^- and v are sufficiently far in rank, with high probability at least one quarter of the vertices in W are $16 \log n/p^3$ from v in rank and hence will be in W'. Thus when $|W'| < \log n/p$ we can conclude that v^- and v were close in rank. Otherwise, pick a random vertex in W' and let it be the new lower marker v^-. What is the probe complexity of this partition step? It uses $O(n \log n)$ probes in each iteration because it probes the incident edges of $O(\log n/p)$ vertices and with high probability no vertex has more than $3np$ incident edges. By Lemma 4.1, the rank difference between v and the lower marker v^- decreases by at least a $\frac{3}{4}$ factor with probability at least $\frac{1}{4}$. So the expected number of rounds is $O(\log n)$ and the probe complexity of this phase is $O(n \log^2 n)$.

We now describe how to finish off pivoting around the vertex v. Using the above algorithm we can verify the order $u < v$ for every vertex u that is much smaller than v. Similarly, we can verify the order $w > v$ for every vertex w that is much larger than v.

We let v^- denote the lower marker in the last iteration of the above algorithm and symmetrically let v^+ denote the upper marker. By Lemma 4.1 and the definition of our algorithm, we have $\pi(v) - \pi(v^-) < 64 \log n/p^3$ and $\pi(v^+) - \pi(v) < 64 \log n/p^3$. We will also verify the relation $u < v^-$ for every vertex u with rank $\pi(u) < \pi(v^-) - 64 \log n/p^3$ using the same recursive procedure we used to do this for v. Similarly we will do this for v^+. We observe that it suffices to probe edges between vertices that are unverified to be smaller than v^- or larger than v^+. This is because all unresolved vertices are between v^- and v^+ and the vertices that are not between v^- and v^+ will not help resolve them. By our discussion, there are only $O(\log n/p^3)$ candidate vertices which could potentially be between v^- and v^+. So we can probe all edges both of whose end points lie in this set. Taking into account that the density is p gives a probe complexity of $O(\log^2 n/p^5)$ for this clean-up step. Thus the overall probe complexity of partitioning with respect to one pivot is $O((n+p^{-5}) \log^2 n)$.

We have described how to partition around one pivot. We now mimic QuickSort until we get down to subproblems of size $O(\log n/p^3)$. Using a recursion tree approach to analyze the recurrence relation for probe complexity of this part and noting that the number of leaves in the tree is $O(np^3/\log n)$ we find that the probe complexity of all partition steps is $O(n \log^3 n + n \log n/p^2)$.

Base Case: Once the problem size becomes $O(\log n/p^3)$, Lemma 4.1 does not apply any more. At this stage, we will simply sort these using the $O(n^{3/2} \log n)$ probe-algorithm. Since each base case requires $O(p^{-9/2}(\log n)^{5/2})$ and there are $np^3/\log n$ base cases, the probe complexity of the base case is $O(n(\log n/p)^{3/2})$.

SORT-SPARSE-RG

1: *Choose markers:* Pick a subset $M \subseteq V$ of markers s.t. each vertex is chosen independently w.p. $\sqrt{p/n} \log^{-1} n$. Let K be the number of markers.
2: *Initial sampling:* Estimate the in-degree of every vertex up to an $\sqrt{np} \log n$ additive error via INIT-SAMPLE with sampling rate $\frac{4}{\sqrt{np}}$.
3: **for** i **from** 1 **to** $K + 1$ **do**
4: *Probe "free" edges:* While there is a free edge, probe the edge and update the assumed direction graph.
5: *Find long path:* Find the directed path P_i consisting of all active vertices that are not verified to be larger than any marker. Remove these vertices.
6: **end for**

Figure 5. The algorithm SORT-SPARSE-RG which sort vertices using $\widetilde{O}(n^{3/2}\sqrt{p})$ probes with high probability in the random graph model.

Putting it all together the overall probe complexity of SORT-DENSE-RG is $O(n \log^3 n + n \log n/p^2) = \widetilde{O}(n/p^2)$. (The base case probe complexity is dominated by one of the terms in the complexity of recursive partitioning no matter whether p is above or below the critical value of $1/\log n$.)

4.2. Algorithm SORT-SPARSE-RG

In this section, we will introduce the algorithm SORT-SPARSE-RG that sorts using $\widetilde{O}(n^{3/2}\sqrt{p})$ probes in the random graph model. Assume that $p > 4 \log n/n$ since otherwise exhaustive probing works.

With high probability, the degree of each vertex v is $O(np)$. By using the subroutine INIT-SAMPLE with sampling rate $4/\sqrt{np}$, we can estimate the in-degree of every vertex up to an additive error of $\sqrt{np} \log n$ using $O(n^{3/2}\sqrt{p} \log n)$ probes. We will modify the criterion of activating a vertex as a function of the density: A vertex $v \in V$ is marked as *active* if the initial sampling and the probed edges indicate that v has at most $6\sqrt{np} \log^2 n$ unprobed incoming edges. Similar to the case for the general graph, we will have a static budget B_{in} of $O(n^{3/2}\sqrt{p} \log^2 n)$ probes for probing the incoming edges of active vertices. We will refer to such edges as "free" edges.

Using the next lemma we can prove that w.h.p the $\sqrt{n/p}$ lowest rank vertices will be active. Note that the lowest rank vertices can have in-edges only from other lowest rank vertices at rate p.

Lemma 4.2: For each subset of vertices $U \subseteq V$ such that $|U| \geq 4 \log n/p$, with probability at least $1 - O(\frac{1}{n^3})$, we have that for every vertex $v \in V$, $E(v, U)$, the number of edges between v and the vertices in U is at least 1 and at most $2p\,|U|$.

If we can recursively find and remove a verified directed path of length $k \geq \sqrt{n/p}$ consisting of the k lowest rank vertices, then the total number of probes would be $\widetilde{O}(n^{3/2}\sqrt{p})$ as desired.

There may be other paths in the graph, but removing the vertices in one of the other paths may lead to a sub-problem that does not contains a Hamiltonian path. In order to identify the path consisting of the lowest rank vertices, we introduce the idea of markers. For each vertex v independently, we pick it as a marker with probability $\sqrt{p/n}\log^{-1} n$. We will let $m_1 < \cdots < m_K$ denote the markers. We note that the order of the markers is unknown to the algorithm and needs to be revealed as the algorithm proceeds. These markers naturally divide the vertices into $K+1$ blocks, each of which consists of the set of vertices between two consecutive markers (w.r.t. the underlying Hamiltonian path). For notational convenience, we let m_0 and m_{K+1} denote two virtual vertices of rank 0 and $n+1$. Let B_i denote the set of vertices between m_{i-1} and m_i (w.r.t. the underlying Hamiltonian path), that is, $B_i \stackrel{\text{def}}{=} \{v \in V : m_{i-1} < v \leq m_i\}$.

By the Chernoff-Höeffding bound, we easily get that with high probability the number of markers is $O(\sqrt{np}\log^{-1} n)$. Furthermore, with high probability we have that for every $1 \leq i \leq K+1$, $\frac{4\log n}{p} \leq |B_i| \leq 4\sqrt{n/p}\log^2 n$. The upper bound holds due to Chernoff-Höeffding bound. The lower bound follows from an argument similar to the birthday paradox. In the following discussion, we will assume these claims indeed hold.

The size of each B_i is large enough that Lemma 4.2 holds with high probability, and is small enough that the vertices in B_i would become active if the algorithm has removed all vertices in $B_1, \ldots, B_{i-1}$. Our strategy is to recursively identify and remove B_i A careful analysis (details are deferred to the full proof) shows that every other vertex is either inactive or verified to be larger than the lowest remaining marker. Hence, all other paths will be ruled out.

The formal description of the algorithm is presented in Figure 5.

Theorem 4.3: The algorithm SORT-SPARSE-RG sorts the vertices in $O\left(n^{3/2}\sqrt{p}\log^2 n\right)$ probes with high probability (over random realizations of graph $\vec{G}$ and random coin flips used by the algorithm).

Proof: It is easy to see that the probe complexity is $O(n^{3/2}\sqrt{p}\log^2 n)$. Now we prove the correctness (with high probability) of the algorithm. We can show using the Chernoff-Höeffding and union bounds that none of the bad events we have identified such as too many markers or vertices of too high degree, occur. In particular, Lemma 4.2 holds for all B_i's. Next, we will show that the algorithm sorts the vertices as desired when these claims hold. Here is the key lemma that we need.

Lemma 4.4: In each iteration of the for-loop, we have $P_i = B_i$.

Proof of Lemma 4.4: We will prove it by induction on i. Let us first consider the base case: $i = 1$. The proof consists of two parts. First, we will show the algorithm has probed all edges along the path consists of the vertices in B_1. Second, we will prove that every other vertex is either inactive or verified to be larger than m_1.

The first part is true for the following reason. By the upper bound on the size of B_1 (Lemma 4.2), and the error bound of initial sampling, all vertices in B_1 are active. So algorithm has probed all edges with both endpoints in B_1 in Step 4. In particular, the algorithm has probed the edges along a subpath of the Hamiltonian path consisting of the vertices in B_1.

Now let us proceed to the second part. We will rephrase the statement as follows: For every active vertex u that is not in B_1, the algorithm has probed a directed path that certifies the fact that u is larger than m_1. Let us first consider the vertices in B_2. By the upper bound on the size of B_1 and B_2, and the error bound of initial sampling, we conclude that the vertices in B_2 are active. Similar to the argument in the previous paragraph, the algorithm has revealed the underlying subpath of the Hamiltonian path consisting of the vertices in B_2 and m_i. So the vertices in B_2 are verified to be larger than m_1. Next, consider an active vertex $u \in V \setminus (B_1 \cup B_2)$. We have that $|E(u, B_2)| > 0$ (Lemma 4.2). Let $w \in B_2$ be a vertex such that $(u, w) \in E$. The edge (u, w) is an incoming edge of either u or w. Since both u and w are active, the algorithm has probed this edge. Hence, w is verified to wins against m_1.

Now suppose the lemma holds for $1, \ldots, i-1$. Then, by the recursive structure of the algorithm, we have removed all vertices in $B_1, \ldots, B_{i-1}$. So the case is now identical to the case $i = 1$. In sum, Lemma 4.4 holds for all i. ■

Suppose the w.h.p. events indeed hold. Then, by Lemma 4.4, the algorithm recursively reveals and removes a subpath of the underlying Hamiltonian path consisting of a subset of lowest rank vertices, and have removed all vertices by the end. So the algorithm sorts the vertices correctly. ■

5. CONCLUSIONS

We give the first non-trivial algorithms for the generalized sorting problem where only a subset of all pairwise comparisons are allowed. When the graph of allowed comparisons is adversarially chosen, we give an algorithm that sorts with high probability using $\widetilde{O}(n^{3/2})$ comparisons. On the other hand, if the allowed comparison graph is randomly generated, then we show sorting can be done using $\widetilde{O}(\min\{\frac{n}{p^2}, n^{3/2}\sqrt{p}\}) = \widetilde{O}(n^{7/5})$ comparisons. In particular, when the density p is $\Omega(1)$ we use a near-optimal $O(n\log^3 n)$ probes.

Algorithms for the classical sorting problem implicitly utilize the existence of balanced comparisons at every stage of the algorithm. This plays a crucial role in getting the

$O(n \log n)$ comparison bound for the classical sorting problem. In contrast, when the set of allowed comparisons is restricted, balanced edges may not exist in intermediate stages of the algorithm. Sorting in this general setting thus becomes a considerably more challenging problem. Nonetheless, the best known lower bound even in our restricted comparison setting remains the classical $\Omega(n \log n)$ information-theoretic lower bound on sorting. We note that for some non-trivial generalizations of the sorting problem, the information-theoretic lower bound is known to be essentially tight. In particular, Fredman [5] showed that when the sorted sequence is known to belong to a subset Γ of all possible permutations, then $\log |\Gamma| + O(n)$ comparisons suffice to sort. Thus the information-theoretic lower bound of $\log |\Gamma|$ is tight to within an additive $O(n)$ term in Fredman's generalization of the sorting problem. However, this upper bound result requires that all possible comparisons are allowed. A natural question is if in our restricted comparison model, the lower bound for sorting can be strengthened to $\Omega(n^{1+\epsilon})$ for some constant $\epsilon > 0$.

REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi, "An $O(n \log n)$ sorting network," in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*. ACM, 1983, pp. 1–9.

[2] N. Alon, M. Blum, A. Fiat, S. Kannan, M. Naor, and R. Ostrovsky, "Matching nuts and bolts," in *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1994, pp. 690–696.

[3] M. Charikar, R. Fagin, V. Guruswami, J. Kleinberg, P. Raghavan, and A. Sahai, "Query Strategies for Priced Information," *Journal of Computer and System Sciences*, vol. 64, no. 4, pp. 785–819, 2002.

[4] M. Dyer, A. Frieze, and R. Kannan, "A random polynomial-time algorithm for approximating the volume of convex bodies," *Journal of the ACM (JACM)*, vol. 38, no. 1, p. 17, 1991.

[5] M. Fredman, "How good is the information theory bound in sorting?" *Theoretical Computer Science*, vol. 1, no. 4, pp. 355–361, 1976.

[6] B. Grünbaum, "Partitions of mass-distributions and of convex bodies by hyperplanes." *Pacific Journal of Mathematics*, vol. 10, no. 4, pp. 1257–1261, 1960.

[7] A. Gupta and A. Kumar, "Sorting and Selection with Structured Costs," in *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 2001, p. 416.

[8] J. Kahn and N. Linial, "Balancing extensions via Brunn-Minkowski," *Combinatorica*, vol. 11, no. 4, pp. 363–368, 1991.

[9] S. Kannan and S. Khanna, "Selection with monotone comparison costs," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2003, pp. 10–17.

[10] J. Komlós, Y. Ma, and E. Szemerédi, "Matching nuts and bolts in $O(n \log n)$ time," in *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1996, pp. 232–241.

[11] R. Motwani and P. Raghavan, *Randomized algorithms*. Cambridge Univ Press, 1995.

[12] G. Rawlins, *Compared to what?: An introduction to the analysis of algorithms*. Computer Science Press, 1992.

APPENDIX

Lemma A.1 (Lemma 2.1 restated): If $\pi_{\widetilde{E}}(v) - \pi_{\widetilde{E}}(u) = c \geq 0$, then $\frac{\left|\Pi_{\widetilde{E}}(v<u)\right|}{\left|\Pi_{\widetilde{E}}\right|} > \frac{1}{e^{c+1}}$.

We will use the Brunn-Minkowski Theorem (Theorem A.2) and Lemma A.3, the proof of which also follows from the Brunn-Minkowski Theorem, to show Lemma A.1.

Theorem A.2: Let $n \geq 1$. For any compact convex set $K, L \subseteq \mathbb{R}^n$, the n^{th} root of the Euclidean volume $\text{Vol}(\cdot)$ is concave with respect to Minkowski combination:

$$\forall \lambda \in [0,1]: \quad \text{Vol}\left((1-\lambda)K + \lambda L\right)^{1/n} \geq (1-\lambda)\text{Vol}(K)^{1/n} + \lambda \text{Vol}(L)^{1/n} \ .$$

Lemma A.3 (e.g. [6]): Let K be a n-dimensional convex body in $\mathbb{R}^n$ with centroid $\boldsymbol{c}$. Let $H = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \cdot \boldsymbol{v} = \boldsymbol{c} \cdot \boldsymbol{v}\}$ be a hyperplane through the centroid of K and $H^+ = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \cdot \boldsymbol{v} \geq \boldsymbol{c} \cdot \boldsymbol{v}\}$. Then, $\text{Vol}(K \cap H^+) \geq \frac{1}{e}\text{Vol}(K)$.

Order polytope. [8]: The order polytope $P_{\widetilde{E}}$ is the set of mappings $f : V \mapsto \mathbb{R}$ such that:

$$\forall v \in V: \quad 0 \leq f(v) \leq 1$$
$$\forall (u,v) \in \widetilde{E}: \quad f(u) \leq f(v)$$

With each linear extension $\pi \in \Pi_{\widetilde{E}}$, we let Σ_π denote the following simplex:

$$\Sigma_\pi = \{f \in P_{\widetilde{E}} : 0 \leq f(\pi^{-1}(1)) \leq \cdots \leq f(\pi^{-1}(n)) \leq 1\} \ .$$

We have the following facts.

Fact A.4: 1) The simplices Σ_π triangulate $P_{\widetilde{E}}$;
2) $\text{Vol}(P_{\widetilde{E}}) = \left|\Pi_{\widetilde{E}}\right| / n!$
3) The centroid of $P_{\widetilde{E}}$ is $\frac{1}{n+1}\pi_{\widetilde{E}}$.

Proof: The first two facts follow from symmetricity. Now we prove the third fact. The $n+1$ vertices of the simplex Σ_π are

$$\begin{aligned}
f^0: &\quad f^0(\pi^{-1}(1)) = \cdots = f^0(\pi^{-1}(n)) = 0\,; \\
f^1: &\quad f^1(\pi^{-1}(1)) = \cdots = f^1(\pi^{-1}(n-1)) = 0\,, \\
&\quad f^1(\pi^{-1}(n)) = 1\,; \\
f^2: &\quad f^2(\pi^{-1}(1)) = \cdots = f^2(\pi^{-1}(n-2)) = 0\,, \\
&\quad f^2(\pi^{-1}(n-1)) = f^1(\pi^{-1}(n)) = 1\,; \\
&\quad \cdots \quad \cdots \\
f^n: &\quad f^n(\pi^{-1}(1)) = \cdots = f^n(\pi^{-1}(n)) = 1\,.
\end{aligned}$$

Hence, the centroid of Σ_π is $\frac{1}{n+1}\left(f^0 + f^1 + \cdots + f^n\right) = \frac{1}{n+1}\pi$. We get that the centroid of $P_{\widetilde{E}}$ is $\frac{1}{\left|P_{\widetilde{E}}\right|}\sum_{\pi \in P_{\widetilde{E}}} \frac{1}{n+1}\pi = \frac{1}{n+1}\pi_{\widetilde{E}}$. ∎

Proof of Lemma A.1: By Fact A.4, we have that $\mathrm{Vol}(P_{\widetilde{E}}) = \left|\Pi_{\widetilde{E}}\right|/n!$. Also, we have $\mathrm{Vol}\left(P_{\widetilde{E}\cup(v,u)}\right) = \left|\Pi_{\widetilde{E}\cup(v,u)}\right|/n! = \left|\Pi_{\widetilde{E}}(v<u)\right|/n!$. Hence, it suffices to show that $\mathrm{Vol}\left(P_{\widetilde{E}\cup(v,u)}\right) > \frac{1}{e^{c+1}}\mathrm{Vol}(P_{\widetilde{E}})$.

For every $-1 \leq \lambda \leq 1$, we let $H^{\lambda}_{(v,u)}$ denote the hyperplane $\{f \in \mathbb{R}^V : f(v)+\lambda = f(u)\}$. And let $H^{\geq\lambda}_{(v,u)} = \bigcup_{\mu\geq\lambda} H^{\mu}_{(v,u)} = \{f \in \mathbb{R}^V : f(v)+\lambda \leq f(u)\}$. Then, $P_{\widetilde{E}\cup(v,u)} = P_{\widetilde{E}} \cap H^{\geq 0}_{(v,u)}$ and $H^{-\frac{c}{n+1}}_{(v,u)}$ is a hyperplane through the centroid of $P_{\widetilde{E}}$. Without loss of generality, let us assume that $\mathbf{0} \in H^1 \cap P_{\widetilde{E}}$.

By Lemma A.3, we get that

$$\mathrm{Vol}\left(P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}\right) \geq \frac{1}{e}\,\mathrm{Vol}\left(P_{\widetilde{E}}\right) \quad . \tag{1}$$

Let $K = P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}$, $L = \{\mathbf{0}\}$, and $\lambda = \frac{c}{n+1}/\left(1+\frac{c}{n+1}\right)$ in Theorem A.2. We get that

$$\mathrm{Vol}\left(\frac{1}{1+\frac{c}{n+1}}P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}\right)^{1/n} \geq \frac{1}{1+\frac{c}{n+1}}\mathrm{Vol}\left(P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}\right)^{1/n} \tag{2}$$

Note that

$$\frac{1}{1+\frac{c}{n+1}}P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)} \subseteq P_{\widetilde{E}} \cap H^{\geq 0}_{(v,u)} \quad . \tag{3}$$

Hence, we have that

$$\begin{aligned}
&\mathrm{Vol}\left(P_{\widetilde{E}} \cap H^{\geq 0}_{(v,u)}\right)\\
\geq&\mathrm{Vol}\left(\frac{1}{1+\frac{c}{n+1}}P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}\right) && \text{(by (3))}\\
\geq&\left(\frac{1}{1+\frac{c}{n+1}}\right)^{n}\mathrm{Vol}\left(P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}\right) && \text{(by (2))}\\
\geq&\left(\frac{1}{1+\frac{c}{n+1}}\right)^{n+1}\mathrm{Vol}\left(P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}\right)\\
>&\frac{1}{e^{c}}\,\mathrm{Vol}\left(P_{\widetilde{E}} \cap H^{\geq -\frac{c}{n+1}}_{(v,u)}\right)\\
&(1+x<e^{x} \text{ for any } x>0)\\
>&\frac{1}{e^{c+1}}\,\mathrm{Vol}\left(P_{\widetilde{E}}\right) && \text{(by (1))}
\end{aligned}$$

This finishes the proof of Lemma A.1. ■