Algorithms for the Generalized Poset Sorting Problem

Shaofeng H.-C. Jiang
School of Computer Science, Peking University, Beijing, China

Wenqian Wang
School of Electronic, Information and Electrical Engineering, Shanghai Jiao Tong University, China

Yubo Zhang
School of Computer Science, Peking University, Beijing, China

Yuhao Zhang
John Hopcroft Center for Computer Science, Shanghai Jiao Tong University, China

Abstract

We consider a generalized poset sorting problem (GPS), in which we are given a query graph $G = (V, E)$ and an *unknown poset* $\mathcal{P}(V, \prec)$ that is defined on the same vertex set V, and the goal is to make as few queries as possible to edges in G in order to fully recover $\mathcal{P}$, where each query (u, v) returns the relation between u, v, i.e., $u \prec v$, $v \prec u$ or $u \not\sim v$. This generalizes both the poset sorting problem [Faigle et al., SICOMP 88] and the generalized sorting problem [Huang et al., FOCS 11].

We give algorithms with $\tilde{O}(n \operatorname{poly}(k))$ query complexity when G is a complete bipartite graph or G is stochastic under the Erdős-Rényi model, where k is the *width* of the poset, and these generalize [Daskalakis et al., SICOMP 11] which only studies complete graph G. Both results are based on a unified framework that reduces the poset sorting to partitioning the vertices with respect to a given pivot element, which may be of independent interest. Moreover, we also propose novel algorithms to implement this partition oracle. Notably, we suggest a randomized BFS with vertex skipping for the stochastic G, and it yields a nearly-tight bound even for the special case of generalized sorting (for stochastic G) which is comparable to the main result of a recent work [Kuszmaul et al., FOCS 21] but is conceptually different and simplified.

Our study of GPS also leads to a new $\tilde{O}(n^{1-1/(2W)})$ competitive ratio for the so-called weighted generalized sorting problem where W is the number of distinct weights in the query graph. This problem was considered as an open question in [Charikar et al., JCSS 02], and our result makes important progress as it yields the first nontrivial sublinear ratio for general weighted query graphs (for any bounded W). We obtain this via an $\tilde{O}(nk + n^{1.5})$ query complexity algorithm for the case where every edge in G is guaranteed to be comparable in the poset, which generalizes a $\tilde{O}(n^{1.5})$ bound for generalized sorting [Huang et al., FOCS 11].

2012 ACM Subject Classification Theory of computation → Sorting and searching; Theory of computation → Online algorithms

Keywords and phrases sorting, poset sorting, generalized sorting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.92

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2304.01623` [19]

Funding *Shaofeng H.-C. Jiang*: Research partially supported by a national key R&D program of China No. 2021YFA1000900 and a startup fund from Peking University.
Yuhao Zhang: Yuhao Zhang is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62102251.

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 92; pp. 92:1–92:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We consider a generalized poset sorting problem and obtain various new algorithmic results. In the *generalized poset sorting* problem (GPS), we are given an undirected *query graph* $G = (V, E)$ and an unknown *poset* $\mathcal{P} = (V, \prec)$. The goal is to fully recover the poset $\mathcal{P}$, that is, to figure out the relation between all $x, y \in V$, through the smallest number of queries to the edges in G. Here, when the algorithm makes a query $(u, v) \in E$, the relation of u and v in the poset, i.e., $u \prec v$, $v \prec u$ or $u \not\sim v$ (which stands for u and v are not comparable), is returned. [1]

When the comparison graph G is a complete graph, GPS reduces to a special case called the *poset sorting* problem which was suggested by [11]. This poset sorting is a fundamental problem since it captures the presence of incomparable elements in a partially ordered set which does not have a linear ordering. For this problem, an algorithm with optimal $\Theta(nk + n \log n)$ query complexity was given in [10] where k is the *width* of the poset (the width of a poset is defined as the size of its largest antichain).[2] However, this $\tilde{O}(nk)$ bound heavily relies on the fact that G is complete, and does not work for our general case where the query graph G can have missing edges (u, v) which forbid the query of the relation between u and v (we shall provide a more detailed technical discussion later).

In fact, the missing edges in the query graph already introduce significant challenges even when the poset $\mathcal{P}$ is a total order (where every two elements are comparable). This special case (general graph G and total order) is called *generalized sorting* whose study was initiated by [18]. The state-of-the-art algorithm for this generalized sorting needs to use $\tilde{O}(\sqrt{mn})$ queries [22] for general graphs G, far from matching the $\Theta(n \log n)$ bound for the classic sorting. On the other hand, a parallel research theme aims to explore whether $\tilde{O}(n)$ query complexity can be obtained for generalized sorting on special graph families. Notably, such algorithms were obtained for complete bipartite graphs [1, 7, 2, 21] and Erdős-Rényi stochastic graphs [18, 22].

Our focus. Thus, a fundamental question is to figure out which families of query graphs G admit algorithms with the optimal $\tilde{O}(nk)$ queries (to match that for the complete graphs [10]) for GPS, where k is the width of the poset. An ideal goal is to achieve this $\tilde{O}(nk)$ bound for general query graphs, but as we mentioned, even for the total order case it is already difficult to improve over $\sqrt{mn}$. Therefore, we instead focus on complete bipartite and Erdős-Rényi stochastic query graphs, which are fundamental cases and were very well studied in the special case of generalized sorting. Moreover, we also study how GPS connects to other settings, especially its implications for variants of generalized sorting. This connection is plausible since a natural way for sorting is to build a partially sorted solution and then solve the remaining sub-problem, and this sub-problem may often be modeled as a poset sorting problem.

Technical challenges. However, designing algorithms for GPS turns out to be nontrivial and requires new approaches. Below, we briefly discuss why the existing techniques from tightly related problems, including poset sorting and generalized sorting problems, cannot be readily applied.

[1] To make sure the problem is well-defined, we need to define the correspondence between $\mathcal{P}$ and query results of G, see Section 3 for more details.

[2] Throughout, $\tilde{O}(f) := O(f \operatorname{poly}\log f)$.

- **Techniques from poset sorting.** The missing edges in G can increase the query complexity of existing algorithms for poset sorting [11, 10]. In these algorithms, the overall framework is to incrementally add elements to the current sorting, and when an element x is to be added, a binary search is used to figure out the relation between x and every other element added so far. A crucial step to bound the query complexity is that there always exists a path cover of size k (which is the width) in the induced subgraph of the added elements, and this ensures only k binary search suffices. While this is true for complete graphs, the width k can no longer upper bound the size of the path cover only by using edges in an induced subgraph of a general G.
- **Techniques from generalized sorting.** Incomparable edges $(u, v) \in E$ $(u \not\sim v)$ reveal very little information about ordering, hence algorithms for generalized sorting should avoid querying these edges. However, existing algorithms [18, 23, 22] for generalized sorting (designed for general query graph G) relies on a rough estimation of the relation between elements, and "useful" edges may be wrongly classified as incomparable edges. If this happens, then it is very difficult to detect the "useful" edge without querying a lot of incomparable edges, making existing algorithms less efficient.

1.1 Our Results

We give efficient algorithms for GPS that make $\tilde{O}(n \operatorname{poly}(k))$ queries for Erdős-Rényi stochastic query graphs (Theorem 1.1) and complete bipartite query graphs (Theorem 1.2), where k is the width of the poset throughout (see Section 3 for formal definitions of these query graph models). These are the first results for GPS parameterized by the width of the poset k, and the query complexity bound is nearly-optimal in n (and up to a factor of k). We obtain our results via a unified framework and it may be of independent interest (will be discussed in Section 4). These results are our main technical contributions.

▶ **Theorem 1.1** (Erdős-Rényi Stochastic Graphs). *There exists an algorithm that solves GPS on Erdős-Rényi stochastic query graphs and width-k posets using $\tilde{O}(nk^2)$ queries with high probability. This holds regardless of the probability parameter $0 < p \leq 1$ in Erdős-Rényi $G(n, p)$.*

Roughly, the Erdős-Rényi stochastic query graph is a union of an adversarily choosen base graph and an Erdős-Rényi graph $G(n, p)$. Although our bound for Erdős-Rényi stochastic query graph does not depend on p, it still relies on the structural property of the Erdős-Rényi graph where edges are i.i.d. generated. This case of Erdős-Rényi query graph has been well studied in (total order) generalized sorting (i.e., $k = 1$), where [18] and [22] are milestones. Compared with [18], our result is significantly better than their $\min\{np^{-2}, n^{1.5}\sqrt{p}\}$, especially that our algorithm is $\tilde{O}(n)$ regardless of p and theirs can obtain near-linear query complexity only for a very limited range of p. On the other hand, compared to the more recent work [22] whose bound is $O(n \log(np))$, our result is worse by a $\operatorname{poly}\log n$ factor. However, our slightly worse bound generalizes to poset sorting and is also technically different. See Section 4 for a more detailed discussion. Finally, we remark that a unique feature of [22] is that when p is very small, say $p = 1/n$, then the complexity of sorting can even be better than $O(n \log n)$ which is the well-known lower bound for classic sorting. We leave it as an open question to figure out if one can achieve a similar bound for GPS with Erdős-Rényi query graphs.

▶ **Theorem 1.2** (Complete Bipartite Graphs). *There exists an algorithm that solves GPS on complete bipartite query graphs and width-k posets using $\tilde{O}(nk)$ queries with high probability.*

This result for complete bipartite graphs is tight up to poly$(\log n)$ factors, since the $\Omega(nk)$ lower bound for complete query graph in [10] still holds in the complete bipartite case. Our result is also a generalization of a series work of nuts-and-bolts problems [1, 7, 2, 21], and our bound for $k = 1$ nearly matches the state-of-the-art bound for this problem (up to poly log factors, compared to the best $O(n \log n)$ result in (total order) generalized sorting problem on complete bipartite graphs by [21]). Note that there is a difference between the nuts-and-bolts problem and the generalized sorting problem on complete bipartite graphs, where the nuts-and-bolts problem has an additional assumption that every node is assigned an edge with the query result "equal". With the help of the "equal" edge, a natural randomized quicksort-like algorithm achieves the query complexity $O(n \log n)$, by an $O(n)$ partition algorithm to partition nodes based on a randomly selected pivot. However, if the "equal" edge is not provided, it is already a non-trivial task to design a partition algorithm. This is also noted by [21], and they resolve the missing "equal" edges and design an $O(n \log n)$ sorting algorithm with some other indirect methods. However, this indirect method does not yield a partition algorithm, and we find it hard to generalize these indirect methods to the setting of poset. In our result, we devised a partition algorithm for complete bipartite graphs without "equal" edges, and this type of partiton algorithm was unknown even for the total order setting. This partition step turns out to be useful and naturally generalizable to posets.

Weighted generalized sorting. Apart from the significance in its own right, another important implication of GPS is that it can be used as an intermediate step for other (total order) sorting problems. We showcase this idea by presenting new results for the weighted generalized sorting problem.

In the weighted (total order) generalized sorting problem, the query graph is weighted (with weight function $w : E \to \mathbb{R}_{\geq 0}$), and each query $(u, v) \in E$ incurs a weighted cost $w(u, v)$ instead of a unit cost. Since the objective is weighted, we measure the performance of the algorithm using the *competitive ratio* [16], defined as the total cost incurred by the algorithm divided by $\sum_i w(v_i, v_{i+1})$ ($v_1 \prec \ldots \prec v_n$ is the total order), which is the cost of cheapest proof, i.e. comparing every consecutive elements in the sorted order. We obtain the following result for weighted generalized sorting.

▶ **Theorem 1.3.** *There exists an algorithm that solves the weighted (total order) generalized sorting problem with competitive ratio $\tilde{O}(n^{1-1/(2W)})$, where W is the number of distinct weights in the graph.*

Indeed, obtaining nontrivial bounds for this weighted generalized sorting has been suggested as an *open question* by [9], and it received significant attention in various subsequent works [16, 3, 4, 13, 14]. However, these existing works mostly focus on understanding certain special cases of weights, such as bounded number of distinct weights [4, 13, 14] or structured/random weights [16, 17, 3]. For the general case, we are only aware of an $O(n)$ ratio [16], which is trivial in the unweighted case since one can query all edges, but is already nontrivial in the weighted setting.

Our result makes progress on the weighted generalized sorting problem, and our bound implies a strictly sublinear ratio when the number of distinct weights is bounded (and the weights can take any non-negative values). This improves over the known $O(n)$ ratio in [16], and our ratio also matches ratios for several notable special cases. When $W = 1$ which reduces to (total order) generalized sorting, our bound matches the $\tilde{O}(n^{1.5})$ query complexity in [18] which is the state-of-the-art for dense graphs (for sparse graphs a $\sqrt{mn}$ bound was obtained in [22], where m is the number of edges in the query graph). Moreover, we give an improved analysis of our algorithm for the case when the weights are well-separated, and this result matches an $\tilde{O}(n^{3/4})$ ratio, obtained in a recent work [13, 14], for the case when the weights are picked from $\{0, 1, n, \infty\}$ (where ∞ weight can be interpreted as "missing edge" in the query graph in our case).

▶ Remark 1.4. In several previous works [16, 18, 22] it has been mentioned that the ratio for finding a maximum element in a weighted query graph is $\Omega(n)$, and this seems to suggest that $\Omega(n)$ is also a lower bound for sorting (since sorting implies finding the maximum). However, this is not true, for two reasons. One is that the ratio in the maximum-finding problem is defined with respect to the cost of a minimum-weight certificate for the maximum, which can be much smaller than the $\sum_i w(v_i, v_{i+1})$ cost for identifying total order in sorting. Secondly, the hard instance in the lower bound of maximum-finding [16] only uses three types of weights, and by our upper bound, this type of instance cannot be hard for sorting. A similar discussion of this gap was also made in [13, 14]. In fact, a further implication of our result is that, in order to prove $\Omega(n)$ lower bound for weighted generalized sorting, one must use at least $\Omega(\log n)$ distinct weights in the hard instance.

Auxiliary problem: GPS with comparable edges (GPSC). As we mentioned, GPS is used as an important intermediate step for obtaining Theorem 1.3. In particular, we consider a special case of GPS whose query graph consists of *comparable* edges only, i.e., $(u, v) \in E$ only if $u \prec v$ or $v \prec u$, and we call this special case the GPS with comparable edges (GPSC). Due to the fact that GPSC is in between GPS and (total order) generalized sorting, and that it may be useful for other sorting problems, the result of this problem, stated below, may be of independent interest.

▶ **Theorem 1.5** (GPSC). *There exists an algorithm that solves GPSC on general query graphs and width-k posets using $\tilde{O}(nk + n^{1.5})$ queries with high probability.*[3]

As mentioned, Theorem 1.5 is a crucial subroutine for Theorem 1.3, but to obtain a sublinear ratio for weighted generalized sorting (provided that the number of distinct weights is bounded), any $n^{2-\epsilon}$ query bound for GPSC suffices, although it may lead to a worse constant in the exponent of n in the ratio (i.e., worse than $O(n^{1-1/(2W)})$ but still $o(n)$). We give a detailed overview on how this can be used to obtain Theorem 1.3 in Section 4.

▶ Remark 1.6 (Comparison to [13, 14]). A recent work [13] independently studies similar problems that are relevant to our results.[4] The meta problem suggested in their paper is a weighted version of the GPSC problem, which they call universal sorting. This weighted GPSC model is more general than the unweighted GPSC (which is the model of our Theorem 1.5), but is incomparable with GPS which we focus on. However, due to a lower bound of $\Omega(n)$ for the universal sorting model, their concrete results/upper bounds only deal with special cases. We compare with each of their main results in the following; in short, all their results concern problems that are either special cases or immediate extensions of ours, but the bounds may not be comparable (i.e., they may obtain better bounds for special cases or use a different measure of complexity).

[3] This algorithm may run in exponential time.
[4] [14] is another version of [13] and is to appear in ITCS'24. We focus on [13] since [14] only contains a subset of results of [13].

- **Weighted generalized sorting with special weights** [13, Theorem 1]. They study a special case of weighted generalized sorting, where all weights are among $\{0, 1, F\}$ ($F > n^{3/4}$), and they achieve a $\tilde{O}(n^{3/4})$ competitive ratio. Our algorithm for general weights achieves the same competitive ratio for this special case.
- **Bichromatic sorting** [13, Theorem 2]. They study a special case of weighted generalized sorting, where the comparison costs are among $\{1, \alpha, \beta\}$ and are assigned to query graph in a structured way. They achieve a nearly optimal ratio of $O(\log^3 n)$ for this special case, and our general algorithm for weighted generalized sorting (our Theorem 1.3) can also be applied which yields a ratio of $O(n^{5/6})$.
- **GPSC with weight-0 edges** [13, Theorem 6]. They propose an algorithm for GPSC that additionally allows weight-0 edges. This algorithm is used as a subroutine in their sorting algorithm. Our algorithm for GPSC (our Theorem 1.5), even though not explicitly stated to be able to handle the weight-0 edges, can still be applied by treating all weight-0 edges as weight-1. Hence, we still obtain the same bound of $\tilde{O}(nk + n^{1.5})$ as in no weight-0 edges, and this is actually better than their $O(n^{1.5} k \log n)$ bound.
- **GPSC on complete bipartite graphs** [13, Theorem 22]. They study GPSC problem on complete bipartite graphs. They measure the performance by instance optimality instead of query complexity, and their algorithm is $O(\log^3 n)$ instance optimal. Since GPSC is a special case of GPS problem, our algorithm for GPS problem on complete bipartite graphs (our Theorem 1.2) can be applied which yields a query complexity of $\tilde{O}(nk)$. This is also a near optimal result because optimal solution requires $\Omega(nk)$ queries in the worst case.

1.2 Technical Contribution

We give a highlight of technical challenges and our technical contributions. A more detailed overview of the proof can be found in Section 4.

A general framework for GPS. Previous algorithms for generalized sorting [18, 23, 22] use an incremental method to iteratively discover the (nearly-)minimum element, but this does not work directly in GPS due to incomparable edges and non-unique minimal elements. We develop a general framework for GPS, which reduces GPS to finding a linear extension, and we further show this linear extension can be found by a quicksort-like algorithm proposed by [18, 10]. The quicksort-like algorithm relies on a partition algorithm that receives as input a subgraph and a pivot, and it partitions the subgraph into three parts: less than, larger than, and incomparable to the pivot. We require the partition algorithm to have a refined query complexity that only depends on the width of the part of the input subgraph (to the partition algorithm) that is comparable to the pivot. This refined property of partition algorithms enables us to obtain $\tilde{O}(nk)$ query complexity for the quicksort-like algorithm. Hence, this framework is capable of obtaining (nearly) tight bounds when combined with carefully designed partition algorithms (for instance, for Erdős-Rényi query graphs and complete bipartite graphs), which may be of independent interest.

Novel partition algorithms for Erdős-Rényi graphs based on stochastic BFS. Our partition algorithm for Erdős-Rényi graphs is based on a stochastic BFS, where the key idea is to skip a vertex from the BFS queue if that vertex has been visited by sufficiently many other vertices. This still guarantees the correctness with high probability due to the property of Erdős-Rényi graph. To make sure we trim most vertices in a few iterations, we also run the BFS in a random order of vertices. Previously, algorithms for Erdős-Rényi graphs were only known for generalized sorting (without considering a poset), and the techniques are not readily applicable. In particular, the framework of [18] requires an algorithm with a subquadratic query complexity for general query graph, which is not available in GPS. Another recent work [22] uses a very different approach, but the efficiency of one of its subroutines relies on the uniqueness of the minimal element. Hence it is highly nontrivial to generalize to the poset setting while achieving subquadratic complexity.

Weighted generalized sorting via new algorithms for GPSC. To achieve the $\tilde{O}(n^{1-\frac{1}{2W}})$ competitive ratio, we partition the edge set into cheap and expensive edges according to a threshold, and the two cases are balanced and solved by one of the following two algorithms: a) a new sorting algorithm that may receive a partially sorted graph (i.e. a partial order) as extra input and the competitive ratio depends on the width of the input partial order; and b) a new GPSC algorithm as in Theorem 1.5. In our new GPSC algorithm, we employ the framework of sorting with predictions [18, 23, 22] (which was proposed for generalized sorting), where we construct a prediction graph that "guesses" the direction of the edges, and make decisions and refine the prediction in an iterative manner. To achieve better query complexity, we devise a stronger predictor that has an "everywhere" guarantee for each vertex, as opposed to having a collective bound on the total number of wrongly predicted edges, as considered in [23, 22].

1.3 Related Work

Parameterization other than the width of the poset which we use was also considered in the literature, and they are generally not comparable to our results. In [5, 6], the GPS is parameterized by the number of missing edges $q = \binom{n}{2} - m$ (where m is the number of edges in the query graph) while there is no restriction on the poset, and nearly-tight bounds were obtained with respect to q. In a recent work [24], the query graph can be general but the poset is assumed to be a tree and is parameterized by the maximum degree d, and they also obtained nearly tight query complexity bounds.

In addition to generalized sorting problems, other related problems were also considered. Examples include noisy sorting/selection [12, 8, 15] and generalized/weighted selection [16, 20, 3, 10].

2 Preliminaries

Throughout, we use $\mathcal{P} = (V, \prec)$ to denote a poset, and we let $k_{\mathcal{P}}$ denote the width of $\mathcal{P}$. By Dilworth's Theorem, a poset of width k can be decomposed to k chains, say $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, where the elements are comparable to each other on each chain. Suppose $\mathcal{P} = (V, \prec)$ is the underlying poset of GPS problem, we directly use k to denote $k_{\mathcal{P}}$. For every $X \subseteq V$, let k_X denote the width of poset $(X, \prec)$. For every $X \subseteq V, v \in X$, let $X_{\prec v}, X_{\nsim v}, X_{\succ v}$ denote the elements of X that are smaller than v, incomparable with v and larger than v respectively. Recall that $x \nsim y$ denotes "x is incomparable with y". For some set X, denote the set of permutations of X by $\mathrm{perm}(X)$. For every set X with a total order $(X, \prec)$ and every $x \in X$, define $\mathrm{rank}_X(x) = |\{y \in X \mid y \prec x\}| + 1$ as the rank of x. Similarly, let $\mathrm{rank}_{\mathbf{p}}(x)$ be the rank of x in permutation $\mathbf{p}$.

For a graph $G = (V, E)$ and a vertex subset $S \subseteq V$, denote $G[S]$ as the induced subgraph of G on S, whose vertex set is S and the edge set is $\{(u, v) \in E : u, v \in S\}$. Given a directed acyclic graph (DAG) $\vec{G}(V, \vec{E})$, let $\mathcal{P}(\vec{G})$, which stands for induced poset of $\vec{G}$, be a poset $\mathcal{P}'(V, \prec)$, such that $\forall u, v \in V$, $u \prec v$ if and only if there is a directed path from u to v in

$\vec{G}$. This implies that $u \not\sim v$ if and only if u cannot reach v and v cannot reach u in $\vec{G}$. By Dilworth's Theorem, a DAG can be covered by k paths, and these paths form a path cover of $\vec{G}$, where k is the width of $\mathcal{P}(\vec{G})$, i.e., every vertex is contained in at least one path (and may be contained in multiple paths).

3 Models

Generalized poset sorting (GPS). Formally, in the GPS problem, we are given an n-element underlying (unknown) poset $\mathcal{P} = (V, \prec)$ and a graph $G = (V, E)$. An oracle receives queries of the form $(u, v) \in E$, and returns the relation of u, v in $\mathcal{P}$. The goal of GPS is to use the minimum number of queries to fully recover $\mathcal{P}$, i.e., $\forall u, v \in V$, correctly determine the relation of u, v in $\mathcal{P}$.

Model of query graphs in GPS. To make sure the problem is well-defined, e.g., G has sufficient edges to recover $\mathcal{P}$, we need to add some further constraints on the query graph G. Specifically, we enforce the following: let $\vec{E} := \{(u, v) \in E : u \prec v\}$ and $\vec{G} = (V, \vec{E})$ (noting that $\vec{G}$ is defined with respect to both G and $\mathcal{P}$) then

$$\mathcal{P} = \mathcal{P}(\vec{G}). \tag{1}$$

This is well-defined if G is deterministic (for instance G is a complete bipartite graph), but for stochastic case enforcing this directly may cause randomness issues. Hence, we discuss how we define the Erdős-Rényi stochastic query graph in more detail in the following.

Model of query graphs in GPS: Erdős-Rényi stochastic case. Let $G(n, p)$ denote the Erdős-Rényi random graph with n vertices and probability parameter $0 \le p \le 1$. Specifically, this $G(n, p)$ is generated by independently adding an undirected edge (u, v) with probability p for every vertex pair $u \neq v$. Clearly, this random graph is unlikely to be able to uniquely identify $\mathcal{P}$. Hence, we still wish to enforce the property stated in (1). Specifically, we need to add to the Erdős-Rényi graph a *minimal* DAG G_{base} which is a "base graph". Here, we say a DAG $\vec{G}$ is minimal if there is no redundant edge in $\vec{G}$, where we call an edge $u \to v$ redundant if we have $\exists x \notin \{u, v\}$, $u \to x$ and $x \to v$. Formally, we have the following definition, and it indeed satisfies (1) (stated in Fact 3.2).

▶ **Definition 3.1.** *Fix some minimal DAG $\vec{G}_{\text{base}}$ such that $\mathcal{P} = \mathcal{P}(\vec{G}_{\text{base}})$, denoting its underlying undirected graph as G_{base}, the Erdős-Rényi stochastic query graph G is a union of G_{base} and $G(n, p)$.*

▶ **Fact 3.2.** *The random query graph $G = (V, E)$ defined in Definition 3.1 satisfies* (1) *with probability* 1*, namely,* $\Pr[\mathcal{P} = \mathcal{P}(\vec{G})] = 1$ *where $\vec{G} := \{(u, v) \in E \mid u \prec v\}$.*

Indeed, this definition can be viewed as a generalization of the stochastic setting in the (total order) generalized sorting [18, 22], here they use a directed Hamiltonian Path as a base graph (which is the only minimal choice in the total order setting).

Generalized poset sorting with comparable edges (GPSC). In this model, all edges in G are comparable edges. Specifically, when an edge in E is queried, the answer will only be $v \prec u$ or $u \prec v$, corresponding to $\mathcal{P}$. We remark that when $\mathcal{P}$ is a total order set, then the model draws back to the generalized sorting model, so GPSC is already a generalization of generalized sorting.

Weighted generalized (total order) sorting. In this model, the poset is total order, and the query graph is weighted by a weight function $w : E \to \mathbb{R}_{\geq 0}$. We aim to minimize the sum of costs we pay to solve the GPS under this setting. We evaluate our algorithms by the competitive ratio, which is the maximum ratio taken over all possible inputs, measured by the cost of the algorithm, denoted as ALG, divided by OPT which is the sum of costs $\sum_i w(v_i, v_{i+1})$ where $(v_1, \ldots, v_n)$ is the total order defined by $\mathcal{P}$.

4 Proof Overview

Due to space limit, we give a sketch for the proofs of main results in this section, and the full proof, except for a more detailed presentation of our main framework (in Section 5), can be found in the full version.

A general framework for generalized poset sorting. As mentioned, we obtain algorithms for GPS via a new unified framework. In this framework, we first reduce the GPS to finding a *linear extension* (Lemma 5.1). A linear extension for the poset $\mathcal{P} = (V, \prec)$ is a total order such that $\forall x, y \in V$, if $x \prec y$ then x appears before y in the total order. Finding a linear extension is an interesting problem in its own right, and it has also been studied in [18, 10]. However, previous studies did not establish the connection between GPS and linear extension, which we do in our framework.

To find the linear extension, we employ a quicksort-like algorithm to randomly select a pivot element $v \in V$ and partition the elements into three parts, elements smaller than v, elements incomparable with v and elements larger than v. Given this partition, one can compute the linear extension of these three parts recursively and combine them in the order of smaller-incomparable-larger to obtain the linear extension of $\mathcal{P}$.

This quicksort-like algorithm was also used in [10] to find a linear extension for complete query graphs. While their analysis may be adapted to the general query graph case, it only leads to sub-optimal bounds with respect to k. We give new analysis to this quicksort-like algorithm, and we are able to obtain an improved dependence in k, provided that the partition algorithm have a refined query complexity that only depends on the width of the part of the input subgraph (to the partition algorithm) that is comparable to the pivot (see Lemmas 5.3 and 5.4 for more details). We manage to design partition algorithms with refined query complexity for both Erdős-Rényi and complete bipartite query graphs.

Now we explain our new steps in the analysis to the quicksort-like algorithm. In [10], it is observed that the depth of the recursion tree is $O(k + \log n)$. This is good enough for complete query graphs, since the partition step can be done in $O(n)$, and this, combined with the depth of the recursion tree, translates to an $O(nk + n \log n)$ bound. However, when G is not a complete graph, the partition problem often requires $\Omega(n)$ queries, say $O(nk^c)$ queries, then the analysis in [10] leads to an $\tilde{O}(nk^{c+1})$ bound, which introduces an additional k factor. In order to avoid this additional k factor, we require partition algorithms to use $O(nk_v^c)$ queries that depend on k_v, which denotes the width of elements comparable with pivot vertex v. A crucial observation is that, if k_v is small, then the partition algorithm uses few queries, and if k_v is large, then the next pivot v' (in the incomparable part) is likely to have a small $k_{v'}$.

Partition algorithms. For the partition step, if it were the complete graph case, we could directly query the relations between the pivot and every other element using $n - 1$ queries. However, this simple but efficient bound is no longer easily obtainable when the query graph is not complete. Nonetheless, we introduce novel ideas for this partition step, and we manage to obtain algorithms that use $\tilde{O}(nk^2)$ queries for Erdős-Rényi query graphs and $\tilde{O}(nk)$ queries for complete bipartite query graphs.

Partition algorithms: Erdős-Rényi graphs. It is helpful to interpret the problem as a graph problem. We define a directed graph $\vec{G}$ from G, by defining the direction of every edge $(u, v) \in E$ according to the relation between u, v, i.e. the direction is $u \to v$ if and only if $u \prec v$. Then for every vertex u, u is smaller than the pivot vertex if and only if there exists a path from u to pivot in $\vec{G}$. Hence, the partition problem reduces to finding all vertices that can be reached from the pivot vertex. This graph problem may be solved using BFS, but a vanilla BFS needs to query all edges, which is too costly. To resolve this issue, we design a variant of BFS that can make use of the structure of Erdős-Rényi graphs, called Skip-BFS.

We start by giving the overall intuition by assuming we are given a chain decomposition of the poset (which is of size k, guaranteed by Dilworth's Theorem). An important property of Erdős-Rényi $G(n, p)$ is that, if we select $O(p^{-1} \log n)$ arbitrary vertices, then every vertex is adjacent to at least one selected vertex[5]. Hence, if we select the $O(p^{-1} \log n)$-largest vertices from each chain in the chain decomposition of the poset then every vertex has outgoing edges to at least one selected vertex. Exploring (the neighbors of) these selected vertices only takes $\tilde{O}(kp^{-1} \cdot np) = \tilde{O}(nk)$ queries, and this finishes the partition.

However, the chain decomposition is not known to our algorithm a priori. Thus, we need a method to gauge whether a vertex is worth exploring, i.e., it is sufficiently large in its chain. To this end, Skip-BFS maintains a counter $c[v]$ for each vertex v, which is initialized as some parameter $R = \Theta(\log n)$. Then, if some vertex v becomes the current vertex for which we start to explore its neighbors, we decrease the counter $c[u]$ by 1 for every v's neighbor u such that u is smaller than v. When the counter of some vertex u is decreased to 0, we skip this vertex u by removing it from the BFS queue. Such u can be safely skipped since Skip-BFS has already explored $R \geq \Omega(\log n)$ vertices that are larger than u, and these vertices are likely to cover all incoming vertices of u. To see this, since u's counter is decreased $R \geq \Omega(\log n)$ times, we already visited $O(p^{-1} \log n)$ vertices that are larger than u, and that each such vertex connects to p fraction of vertices smaller than u. Hence, these already-visited $O(p^{-1} \log n)$ vertices connect to/cover all vertices that are smaller than u. Finally, to guarantee the efficiency of this process, we need to explore vertices in a random order for each BFS step (in step i we explore all vertices with distance i from the starting vertex), in order to trim most u's in only a few steps. This eventually leads to an $\tilde{O}(nk^2)$ time partition algorithm for Erdős-Rényi query graphs.

Compared with the approach in [22] who gave an algorithm for the total order case that uses $O(n \log(np))$ queries which is tight, our bound is comparable, but our approach is conceptually different. In fact, it is unclear if their approach can be efficiently generalized to the poset case. In their algorithm, they repeatedly find the minimum vertex of the current graph and remove it. To find the minimum vertex, they identify a set of candidate vertices and then trim the wrong ones by testing if there is an incoming edge, which requires querying the edges between the candidates and other vertices. However, in GPS, there are multiple minimal vertices, and it is nontrivial to bound the number of candidates and the adjacent edges to query since one cannot stop before one is certain that the surviving candidates are minimal. Hence, it is nontrivial to generalize their approach to GPS using even subquadratic queries.

[5] This does not always happen and only with high probability, but in the following discussions we ignore this and talk about the typical behavior.

Partition algorithm: complete bipartite graphs. Suppose the two parts of the bipartite graph are A and B, and suppose the pivot is $b \in B$. Let $A_{\succ b}$ and $B_{\succ b}$ be the elements in A and B that are greater than b, respectively. We focus the discussion on finding the elements that are larger than b, i.e., $A_{\succ b} \cup B_{\succ b}$. Since the graph is complete bipartite, it is easy to obtain $A_{\succ b}$, but it is nontrivial to obtain $B_{\succ b}$ since one cannot directly compare any other point in B with the pivot b. A natural idea to deal with this is to find the minimal elements in $A_{\succ b}$ so that one can figure out $B_{\succ b}$ from these elements. However, finding the minimal element is technically nontrivial even in the total order setting (whose minimal is unique), let alone there may be multiple minimal elements in posets. Indeed, the existing algorithm for the total order setting does not seem to make progress on this simple and fundamental task, and they solved the problem via other indirect methods [1, 7, 2, 21]. We provide a completely new algorithm to find the minimal elements for poset in bipartite complete graphs, which is a technical contribution to the study of sorting and selection for bipartite graphs.

We first devise a FindMin procedure that finds a "local" minimal element in $A_{\succ b}$ (the "local" is due to the fact that we may make iterative calls and only run the procedure on an induced subgraph). Then, we apply this FindMin iteratively to both find the minimal elements of $A_{\succ b}$ and construct $B_{\succ b}$. In particular, every time we run FindMin to obtain a vertex a, we try to find $B_{\succ a}$ and expand the currently found $B_{\succ b}$, and remove a from A to continue. Each iteration takes $\tilde{O}(n)$ queries. Then, using the property and the randomness of FindMin, the new element a' that we find must be smaller than a if they are comparable, and this also shrinks the distance from a to the pivot by a constant factor with good probability. Finally, if one takes one chain in a chain decomposition, this entire process would typically run on a vertex from this chain for $O(\log n)$ iterations. Summing over k chains, the total query time is $\tilde{O}(nk)$ in the typical case.

GPSC. Recall that there are no incomparable edges in the query graph (which means every edge $(u, v) \in E$ satisfies either $u \prec v$ or $v \prec u$) of the GPSC problem. This conceptually simplifies the problem since this avoids the issue of gaining essentially no information from querying an incomparable edge. Technically, this allows us to apply techniques/frameworks developed for generalized sorting problems, which crucially relies on the information gained from querying an edge. Specifically, we use an idea proposed in [18] and further developed in [22], where one first constructs a prediction graph which "guesses" the direction/relation of all edges in the query graph. Then, an incremental algorithm that iteratively adds a currently "minimal" element, i.e., an element u with a small number of "incoming edges" (which are the edges (v, u) such that $v \prec u$) in the prediction, is employed to generate the sorting.

To apply this framework to GPSC, especially to achieve a linear dependence in k, we cannot use [18, 22] in a black-box way, since we need a stronger predictor such that it has a bounded number of wrongly predicted edges *everywhere*: $\forall v$, there are $\tilde{O}(\sqrt{n})$ wrongly predicted edges among all adjacent edges to v. This is stronger than the previously designed predictors [18, 22], since [18] only guarantees an $\tilde{O}(\sqrt{n})$ absolute error for the in-degree of every vertex (instead of the edge predictions), and [22] only guarantees an overall number of wrong edges (instead of our "everywhere" guarantee). We manage to obtain such a stronger predictor.

Then, with this predictor, we iteratively maintain a current set A of sorted vertices, and we show it is possible to identify a key vertex $v \in A$, whose incoming vertices (with respect to the prediction) can be partitioned into $X_v \subseteq A$ and $Y_v \subseteq V \backslash A$, such that the poset induced by X_v still has width k, and that $|Y_v| = O(\sqrt{n})$ (by the stronger guarantee of the predictor).

This, together with the fact that X_v can be decomposed into k chains, implies that one can verify/discover all incoming edges (from the predictor) to v using $O(k \log n + \sqrt{n})$ queries. In total, this entire iterative process of updating A happens $O(n)$ times, which leads to our final query bound $\tilde{O}(nk + n^{1.5})$.

Weighted generalized sorting. We start with designing an $O(k \operatorname{poly} \log n)$-competitive algorithm $\mathcal{A}$, whose input consists of a chain decomposition (of the total order) of size k in addition to the weighted query graph (and the underlying total order), for weighted generalized sorting. Notice that one can always feed a trivial chain decomposition of size n to $\mathcal{A}$ and obtain $\tilde{O}(n)$ competitive ratio, which is already nontrivial as we mention in Section 1.1. Although the algorithm by [16] can also achieve an $O(n)$ ratio for weighted generalized sorting, it only works for the case when all chains are single nodes (i.e., $k = n$), hence it is not useful for obtaining a sublinear ratio.

Next, we employ a threshold algorithm to "combine" $\mathcal{A}$ with Theorem 1.5 to obtain a sublinear ratio when the number of distinct weights is bounded. Suppose the weights are $w_1 < \ldots < w_W$. We use a threshold parameter $1 \le \tau \le W$, and define G_τ as the subgraph of the query graph with edge weights *at most* w_τ. We also consider the poset $\mathcal{P}_\tau$ induced by G_τ, and let k_τ denote its width. We start with running Theorem 1.5 on $(G_\tau, \mathcal{P}_\tau)$ and ignoring the weight, which takes $\tilde{O}(nk_\tau)$ queries (assuming $k_\tau \ge \sqrt{n}$ in this discussion), and it generates a chain decomposition of $\mathcal{P}_\tau$. Notice that this chain decomposition of $\mathcal{P}_\tau$ is also a chain decomposition of $\mathcal{P}$ since they are supported on the same element set. Then, we feed this chain decomposition to $\mathcal{A}$, and use the output of $\mathcal{A}$ as the result. The entire algorithm achieves an $\tilde{O}(\frac{nk_\tau \cdot w_\tau}{\mathrm{OPT}} + k_\tau)$ ratio, and we can further show this ratio is at most $O(\frac{nw_\tau}{w_{\tau+1}})$ (assuming that k_τ is not the dominating factor), which depends on the "gap" between two adjacent weights. The final result can be achieved by fine-tuning of τ to minimize this ratio: if all weights are of a small gap, then one can view it as the unweighted case and run Theorem 1.5 directly, and otherwise, we have a significant gap which still allows a sublinear ratio.

5 A General Framework for Generalized Poset Sorting

In this section, we present our framework for GPS. As mentioned, this framework consists of two steps: it first reduces GPS to finding a linear extension, and eventually reducing the task of finding a linear extension to constructing a partition oracle. We start with formally defining the mentioned linear extension problem and the partition problem. We establish two sets of technical lemmas that relate GPS with linear extension Lemma 5.1 and the partition problem Lemmas 5.3 and 5.4, respectively.

Linear extension problem. In linear extension problem, there is an underlying poset $\mathcal{P} = (V, \prec)$ and query graph $G = (V, E)$. The algorithm receives G as input and has access to an oracle, which accepts queries u, v such that $(u, v) \in E$ and answers the relation of u, v in $\mathcal{P}$. The algorithm needs to compute a linear extension of $\mathcal{P}$ by making as few queries as possible. We call $p_1, p_2, \ldots, p_n \in \operatorname{perm}(V)$ a linear extension of $\mathcal{P}$ if and only if for every $1 \le i < j \le n$, $p_i \not\succ p_j$ (i.e. either $p_i \prec p_j$ or $p_i \nsim p_j$).

▶ **Lemma 5.1** (Linear Extension to Poset). *There exists an algorithm that given a linear extension of the underlying width-k poset $\mathcal{P}$ solves GPS in $\tilde{O}(nk)$ queries.*

Lemma 5.1 shows that given any linear extension of $\mathcal{P}$, we can solve $\mathcal{P}$ using $\tilde{O}(nk)$ queries. Specifically, our algorithm maintains a vertex set X such that all directions of edges in $G[X]$ is determined. Initially, X is empty. The vertices are added to X by their order in the linear extension. When vertex v is added to X, our algorithm needs to determine all directions of edges between X and v. In the proof of Lemma 5.1, we show that X can always be decomposed into at most k paths. For every vertex v and every path $P = p_1 \prec p_2 \prec \ldots \prec p_s$, there exists a, b such that

- For every $1 \leq i \leq a$, $p_i \prec v$.
- For every $a < i < b$, $p_i \nsim v$.
- For every $b \leq i \leq s$, $p_i \succ v$.

a, b can be found by binary search. Hence the directions of edges between X and v can be determined in $O(k \log n)$ queries. Our algorithm is shown as Algorithm 1.

Algorithm 1 Constructing GPS using Linear Extension.

```
1: procedure GPS($p$)
2:     input: $p$, a linear extension of $\mathcal{P}$
3:     for $i \leftarrow 1, 2, \ldots, n-1$ do
4:         let $X_i$ be $\{p_1, p_2, \ldots, p_i\}$
5:         compute a path cover of induced subgraph $\vec{G}[X_i]$, denoted by $P_1, P_2, \ldots, P_{w_i}$ ($w_i$
   denotes the width of poset $\mathcal{P}(\vec{G}[X_i])$)
6:         determine the directions of edges between $X_i$ and $p_{i+1}$ by applying binary search
   between $P_j$ and $p_{i+1}$ for every $1 \leq j \leq w_i$
7:     end for
8: end procedure
```

▶ **Lemma 5.2.** *For every linear extension $p_1, p_2, \ldots, p_n$ of $\mathcal{P}$ and every $1 \leq i \leq n$, we have $\mathcal{P}(\vec{G}[X_i]) = (X_i, \prec)$ where $X_i = \{p_1, p_2, \ldots, p_i\}$.*

Proof. To prove $\mathcal{P}(\vec{G}[X_i]) = (X_i, \prec)$, we show that for every $u, v \in X_i$, $u \prec v$ if and only if u can reach v in $\vec{G}[X_i]$.

- If u can reach v in $\vec{G}[X_i]$, then u can also reach v in $\vec{G}$, which implies $u \prec v$.
- If $u \prec v$, then there exists a path $u \to q_1 \to q_2 \to \ldots \to q_\ell \to v$ in $\vec{G}$. For every $1 \leq j \leq \ell$, suppose $q_j \notin X_i$, let $q_j = p_a, v = p_b$, we have $p_a \prec p_b$ and $b \leq j < a$, which contradicts the definition of linear extension. Hence we have $q_1, q_2, \ldots, q_\ell \in X_i$, which implies u can reach v in $\vec{G}[X_i]$. ◀

Proof of Lemma 5.1. By Lemma 5.2, for every $1 \leq i < n$, $\mathcal{P}(\vec{G}[X_i]) = (X_i, \prec)$. The width of poset $(X_i, \prec)$ is no more than k since $X_i \subseteq V$. Hence, for every i, all directions of edges between $p_1, p_2, \ldots, p_i$ and p_{i+1} can be determined by applying at most k binary searches on the path cover of $\vec{G}[X_i]$. Clearly, this entire process takes $O(nk \log n)$ queries in total, which finishes the proof. ◀

Partition problem. The partition algorithm is defined on an underlying DAG $\vec{G} = (V, \vec{E})$ and a query graph $G = (V, E)$. Notice that partition algorithm is a pure graph problem (there is no poset in the problem definition). The algorithm receives G and vertex $p \in V$ as input and has access to an oracle, which accepts queries u, v such that $(u, v) \in E$ and answers the relation of u, v in $\vec{G}$. There are three possible relations, u can reach v, v can reach u, neither u nor v can reach each other. The algorithm needs to compute $V_{\to p}, V_{\not\leftrightarrow p}, V_{\leftarrow p}$ by making as few queries as possible, where

- $V_{\to p} = \{u \in V \mid u \neq p, u \text{ can reach } p\}$
- $V_{\leftarrow p} = \{u \in V \mid u \neq p, p \text{ can reach } u\}$
- $V_{\not\leftrightarrow p} = \{u \in V \mid u \neq p, \text{neither } u \text{ nor } p \text{ can reach each other}\}$

The following two lemmas reduce the problem of finding linear extensions to finding a partition of the elements with respect to a given pivot. These two versions of lemmas are essentially the same, except that one allows the partition oracle to make mistakes and with a randomized query complexity (but needs to succeed with high probability), and the other requires the correctness (with probability 1) and a good query complexity in expectation. We need these two since we find it is not trivial to convert one to the other, and our downstream algorithms may need both of them.

▶ **Lemma 5.3.** *If for every $X \subseteq V$ and $p \in X$, Partition$(G[X], p)$ correctly outputs $X_{\to p}, X_{\not\leftrightarrow p}, X_{\leftarrow p}$ within $O(|X|(k_{X_{\prec v}} + k_{X_{\succ v}})f(n,k))$ queries with probability $1-\varepsilon$ (f is some function of n,k), then Part-to-LE(V) outputs a linear extension of $\mathcal{P}$ in $O(nkf(n,k)\log^2 n)$ queries with probability of at least $1 - n\varepsilon - 2n^{-6}$.*

▶ **Lemma 5.4.** *If for every $X \subseteq V$ and $p \in X$, Partition$(G[X], p)$ always correctly outputs $X_{\to p}, X_{\not\leftrightarrow p}, X_{\leftarrow p}$ and uses $O(|X|(k_{X_{\prec v}} + k_{X_{\succ v}})f(n,k))$ queries in expectation, then Part-to-LE(V) outputs a linear extension of $\mathcal{P}$ in $O(nkf(n,k)\log^2 n)$ queries in expectation.*

References

1 Noga Alon, Manuel Blum, Amos Fiat, Sampath Kannan, Moni Naor, and Rafail Ostrovsky. Matching nuts and bolts. In *SODA*, pages 690–696. ACM/SIAM, 1994.

2 Noga Alon, Phillip G. Bradford, and Rudolf Fleischer. Matching nuts and bolts faster. *Inf. Process. Lett.*, 59(3):123–127, 1996.

3 Stanislav Angelov, Keshav Kunal, and Andrew McGregor. Sorting and selection with random costs. In *LATIN*, volume 4957 of *Lecture Notes in Computer Science*, pages 48–59. Springer, 2008.

4 Indranil Banerjee and Dana Richards. Sorting under 1-∞ cost model. *CoRR*, abs/1508.03698, 2015.

5 Indranil Banerjee and Dana S. Richards. Sorting under forbidden comparisons. In *SWAT*, volume 53 of *LIPIcs*, pages 22:1–22:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

6 Arindam Biswas, Varunkumar Jayapaul, and Venkatesh Raman. Improved bounds for poset sorting in the forbidden-comparison regime. In *CALDAM*, volume 10156 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2017.

7 Phillip G Bradford. Matching nuts and bolts optimally. Technical report, Max-Planck-Institut für Informatik, 1995.

8 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *SODA*, pages 268–276. SIAM, 2008.

9 Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon M. Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. *J. Comput. Syst. Sci.*, 64(4):785–819, 2002.

10 Constantinos Daskalakis, Richard M. Karp, Elchanan Mossel, Samantha J. Riesenfeld, and Elad Verbin. Sorting and selection in posets. *SIAM J. Comput.*, 40(3):597–622, 2011.

11 Ulrich Faigle and György Turán. Sorting and recognition problems for ordered sets. *SIAM J. Comput.*, 17(1):100–113, 1988.

12 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994.

13 Mayank Goswami and Riko Jacob. Sorting with priced comparisons: The general, the bichromatic, and the universal. *CoRR*, abs/2211.04601v2, 2023. A subset of results in this paper has been accepted to ITCS 2024, as [14]. `arXiv:2211.04601`.

14 Mayank Goswami and Riko Jacob. An algorithm for bichromatic sorting with polylog competitive ratio. In *ITCS*, 2024. See also `arXiv:2311.05773`. A prelimary version [13] has appeared in arXiv.

15 Yuzhou Gu and Yinzhan Xu. Optimal bounds for noisy sorting. In *STOC*. ACM, 2023.

16 Anupam Gupta and Amit Kumar. Sorting and selection with structured costs. In *FOCS*, pages 416–425. IEEE Computer Society, 2001.

17 Anupam Gupta and Amit Kumar. Where's the winner? max-finding and sorting with metric costs. In *APPROX-RANDOM*, volume 3624 of *Lecture Notes in Computer Science*, pages 74–85. Springer, 2005.

18 Zhiyi Huang, Sampath Kannan, and Sanjeev Khanna. Algorithms for the generalized sorting problem. In *FOCS*, pages 738–747. IEEE Computer Society, 2011.

19 Shaofeng H. C. Jiang, Wenqian Wang, Yubo Zhang, and Yuhao Zhang. Algorithms for the generalized poset sorting problem, 2023. `arXiv:2304.01623`.

20 Sampath Kannan and Sanjeev Khanna. Selection with monotone comparison cost. In *SODA*, pages 10–17. ACM/SIAM, 2003.

21 János Komlós, Yuan Ma, and Endre Szemerédi. Matching nuts and bolts in o(n log n) time. *SIAM J. Discret. Math.*, 11(3):347–372, 1998.

22 William Kuszmaul and Shyam Narayanan. Stochastic and worst-case generalized sorting revisited. In *FOCS*, pages 1056–1067. IEEE, 2021.

23 Pinyan Lu, Xuandi Ren, Enze Sun, and Yubo Zhang. Generalized sorting with predictions. In *SOSA*, pages 111–117. SIAM, 2021.

24 Jishnu Roychoudhury and Jatin Yadav. Efficient algorithms for sorting in trees. *CoRR*, abs/2205.15912, 2022. `arXiv:2205.15912`.