An Efficient Semi-Streaming PTAS for Tournament Feedback Arc Set with Few Passes

Anubhav Baweja
Carnegie Mellon University, Pittsburgh, PA, USA

Justin Jia
Carnegie Mellon University, Pittsburgh, PA, USA

David P. Woodruff
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

We present the first semi-streaming polynomial-time approximation scheme (PTAS) for the minimum feedback arc set problem on directed tournaments in a small number of passes. Namely, we obtain a $(1+\varepsilon)$-approximation in time $O\left(\text{poly}(n)2^{\text{poly}(1/\varepsilon)}\right)$, with p passes, in $n^{1+1/p} \cdot \text{poly}\left(\frac{\log n}{\varepsilon}\right)$ space. The only previous algorithm with this pass/space trade-off gave a 3-approximation (SODA, 2020), and other polynomial-time algorithms which achieved a $(1+\varepsilon)$-approximation did so with quadratic memory or with a linear number of passes. We also present a new time/space trade-off for 1-pass algorithms that solve the tournament feedback arc set problem. This problem has several applications in machine learning such as creating linear classifiers and doing Bayesian inference. We also provide several additional algorithms and lower bounds for related streaming problems on directed graphs, which is a largely unexplored territory.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Minimum Feedback Arc Set, Tournament Graphs, Approximation Algorithms, Semi-streaming Algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2022.16

Related Version *Full Version*: https://arxiv.org/abs/2107.07141

Funding *David P. Woodruff*: partial support from NSF grant No. CCF-181584 and a Simons Investigator Award.

1 Introduction

Graph problems have historically been an area of interest because of their various applications, but as the size of these problems becomes very large it becomes essential to design algorithms suitable for models handling such graphs, such as the streaming model. In the streaming model, the input graphs are presented as a read-once tape of edges where a possibly adversarial ordering of observed edges must be accounted for, and low space complexity must be maintained. Much work studying undirected graphs is already present in the streaming literature[16]; we refer the reader to the survey by McGregor [24]. However, relatively little has been done, especially on the algorithmic side, in the streaming setting for problems involving directed graphs (digraphs). A notable exception is the the initial investigation conducted by Chakrabarti et al. [11] (see also lower bounds in [5, 6, 13, 20]), who study the Minimum Feedback Arc Set problem on tournament graphs, among other problems.

In this work, our focus is also on large directed graphs in the streaming setting, and in particular we present a new algorithm for the Minimum Feedback Arc Set problem on tournament graphs. A tournament graph is a directed graph where there exists a single edge between all pairs of vertices, and the goal is to find the minimum number of edges that need to be deleted in order to make the graph acyclic. We will also discuss applications and other problems for directed graph streams below.

1.1 Applications

The Minimum Feedback Arc Set problem on tournament graphs can be rephrased as the Ranking by Pairwise Comparison (RPC) problem. Given a finite set V and a set of pairwise preference labels (denoted by $u \prec v$ if v is preferred over u, where $u, v \in V$), the goal of the RPC problem is to find an ordering of the elements of V, from least preferred to most preferred, to minimize the number of disagreements. Finding a suitable global ranking for data, described only by pairwise preference relationships, arises in various practical applications and is commonly referred to as Kemeny-Young Rank Aggregation [22]. This has applications to machine translation [25] and ranking search engine results [19]. Whereas the bulk of the learning to rank literature involves rating standalone inputs on a predefined scale, this ranking task involves relative relationships between the objects, whether they are webpage search results or tournament competitors. An ordering of elements that satisfies this condition of minimizing regret (up to some error) and another condition of "local chaos" (described by [1]), can also be used to create a regularized large margin linear classifier. The main focus of Ailon's work [1] was to demonstrate the application's feasibility in the query-efficient setting, and is shown here to also be viable in a semi-streaming setting.

Algorithms for solving the Minimum Feedback Arc Set problem can also be used to decrease the computational cost of Bayesian inference [17, 7] by reducing the weighted loop cutset problem to a weighted blackout-feedback vertex set problem. Bayesian networks are popular among the machine learning community, as they provide an interpretable representation of data, along with varying degrees of conditional independence between attributes. The so-called updating problem in Bayesian networks can be solved using the conditioning method; however, the conditioning method runs in time exponential in the size of a loop cutset, which is potentially large, so reducing the problem and using a feedback arc set approximation would reduce the complexity of solving the updating problem.

Other applications in a similar domain for the Minimum Feedback Arc Set problem include collaborative filtering [14], where ordered recommendations need to be generated for users based on their past preferences and the product history for customers with similar interests. There are also various applications for other directed graph problems such as computing the strongly connected components of a graph, including the study of model checking in formal verification [26] and for data flow analysis in compiler optimization [8].

Before describing our results, we need to set up some notation. Let $O^*(f(n))$ denote a function bounded by $f(n)$ poly $\left(\frac{\log n}{\varepsilon}\right)$.

The Feedback Arc Set problem for tournaments is parameterized by V, the vertex set, and E, the edge set of a tournament graph. One can also use a weight matrix W to represent the edge set E such that $W(u, v) = 1$ if $(u, v) \in E$ and 0 otherwise.

▶ **Definition 1.** *The **cost of a permutation** π on vertices with respect to the vertex set V and weight matrix W is*

$$C(\pi, V, W) = \sum_{u,v \in V : \rho_\pi(u) < \rho_\pi(v)} W(v, u)$$

where $\rho_\pi(u)$ is the rank of vertex u in permutation π.

The Feedback Arc Set problem aims to find a permutation π^* of vertices for which

$$\pi^* = \text{argmin}_\pi C(\pi, V, W)$$

▶ **Definition 2.** *The **restricted cost of a permutation** π on vertices for a given vertex set V and weight matrix W, with respect to the edge set E, is*

$$C_E(\pi, V, W) = \frac{\binom{n}{2}}{|E|} \sum_{(u,v)\in E:\rho_\pi(u)<\rho_\pi(v)} W(v,u)$$

To later demonstrate lower bounds for other directed graph problems, reductions to communication games are used as in Chakrabati et al. [11], where they proved lower bounds, both in the single pass and the multi-pass setting, for a variety of digraph problems such as performing a topological sort, detecting if a graph is acyclic, as well as finding a minimum feedback arc set. A classic problem in communication complexity is the `INDEX` problem. Here Alice and Bob are given a vector $x \in \{0,1\}^n$ and an index $i \in [n]$, respectively, and the goal is for Bob to correctly determine whether x_i is a one or a zero. If only Alice can send a single message to Bob, then the minimum length of this message is $\Omega(n)$ for any randomized protocol which succeeds with probability at least $2/3$. For demonstrating a lower bound for multi-pass algorithms, another useful problem is the set chasing problem [20], which is discussed in Section 4.

1.2 Contributions and Previous Work

1.2.1 Minimum Feedback Arc Set

Although the Minimum Feedback Arc Set problem is NP-hard even for tournament graphs [3, 12], a significant amount of work has been done in order to obtain polynomial time approximations [2, 15, 4, 22] for the problem on tournament graphs. Furthermore, Chen et al. [13] gave an $n^{2-o(1)}$ space lower bound, even for $o(\sqrt{\log n})$ pass streaming algorithms for the feedback arc set problem on general graphs. The maximum acyclic subgraph problem, which is the dual of the minimum feedback arcset problem, also has a lower bound of $O(n^{1-\varepsilon^{c/p}})$ space for p-pass, $(1+\varepsilon)$-approximation polynomial time algorithms, where c is a constant [5]. This further motivates the exploration of algorithms for restricted graphs such as tournament graphs.

Our main contribution is the first algorithm that uses $O(n^{1+1/p}\ \mathrm{poly}(\log n/\varepsilon))$ space and p passes, while providing a $(1+\varepsilon)$-approximation, in polynomial time (Theorem 12). This result gives a significant improvement over [11], which achieved the same pass/space trade-off, but could only give at best a 3-approximation. Our work also significantly improves other polynomial-time algorithms which achieved a $(1+\varepsilon)$-approximation, as such algorithms either require quadratic memory or a linear number of passes. Note that the polynomial time requirement is also crucial, since an exponential time algorithm can achieve a $(1+\varepsilon)$-approximation with only 1 pass and nearly linear amount of memory with a brute force algorithm, as demonstrated by Chakrabarti et al. [11].

Note that given $\log n$ passes, this algorithm achieves $O(n\ \mathrm{poly}(\log n/\varepsilon))$ space, consistent with the original definition of the semi-streaming model. Previously, Kenyon and Schudy [22] gave a PTAS for the problem which uses $\Theta(n^2)$ space and Ailon [1] introduced an algorithm that specifically reduces the query complexity. The latter has a similar motivation to the streaming model, in that there is limited access to the input, but the model is substantially different from the streaming model. To the best of our knowledge, this is the first work to present such tradeoffs for algorithms that solve the Minimum Feedback Arc Set problem on tournaments up to a $(1+\varepsilon)$-approximation. A summary of previous work in the streaming model on this problem, as well as our results, is given in Table 1.

Table 1 Algorithms for the Feedback Arc Set problem on Tournament graphs.

Algorithm	Approx	Time	Passes	Space
Sort by wins [15]	5	Polynomial in n	1	$O^*(n)$
Kwiksort [2]	3	Polynomial in n	$O^*(1)$	$O^*(n)$
Modified Kwiksort [11]	3	Polynomial in n	p	$O^*(n^{1+1/p})$
PTAS [22]	$1+\varepsilon$	Polynomial in n	1	$O(n^2)$
Brute force solve with sketching [11]	$1+\varepsilon$	Exponential in n	1	$O^*(n)$
Sample and Rank [1]	$1+\varepsilon$	Polynomial in n	$O^*(n)$	$O^*(n)$
Our Result	$1+\varepsilon$	Polynomial in n	p	$O^*(n^{1+1/p})$

1.2.2 Additional Streaming Problems on Directed Graphs

We also continue the study of other problems on directed graphs in data streams. There has been little research in identifying strongly connected components of a directed graph (digraph) in the streaming model. Laura et al. [23] demonstrated an algorithm with $O(n \log n)$ space to find the strongly connected components in the W-stream setting, but we focus on the standard insertion-only framework while using as few passes and memory as possible. Section 3 introduces an algorithm that can be implemented with $\tilde{O}(n^{1+1/p})$ space in p passes, using a deterministic subroutine that finds a Hamiltonian path and is guaranteed to meet the space constraints, in contrast to the commonly adopted KWIKSORT algorithm [11] that only does so with high probability guarantees.

Lastly, Section 4 demonstrates lower bounds for the space complexity of three directed graph problems: finding strongly connected components, determining if the graph is acyclic, and determining whether there exists a path from a given vertex s to all other vertices. The results are presented for two kinds of space lower bounds: for single pass settings and for multiple pass settings. These strong lower bounds also motivate the shift of emphasis from solving the aforementioned directed graph problems for general inputs to special classes of digraphs, in particular tournament graphs.

1.3 Techniques and Intuition

We discuss our key techniques and intuition for our main result for the Feedback Arc Set problem on tournaments. The techniques employed for our other results are outlined in Section 3.

Previous work by Kenyon and Schudy [22] and Ailon [1] uses the idea of **single vertex moves**: given a permutation π, take a vertex u and move it to an index j such that the cost of the new permutation decreases. Ailon's work shows in order to obtain a $(1+\varepsilon)$-approximation, one does not need to strictly reach a local optimum with respect to cost-improving single vertex moves. Rather, making long moves with significant cost improvement suffices. This holds because the algorithm is recursive, so the shorter single vertex moves can be optimized away in base cases via brute force or the additive approximation algorithm given by Frieze and Kannan [18], which we can turn into a relative error approximation. Note that this latter algorithm is also used by Kenyon and Schudy [22], but we show that this algorithm can also be used in the semi-streaming setting as well.

A brief description of our algorithm is given below:

1. The algorithm first obtains an $O(1)$-approximation to the Minimum Feedback Arc Set by sorting the vertices by indegree. This achieves a 5-approximation as shown by Coppersmith et al. [15]. This is easy to compute in one pass in a stream, and already brings us close to the $(3+\epsilon)$-approximation factor of [11].

2. In order to bring the approximation factor down from 5 to $(1+\varepsilon)$, we recursively partition the vertex set into smaller vertex sets, while finding and applying cost-improving single vertex moves to the permutation:
   - The base case of the recursion is reached when the input vertex set is small enough that we can brute force through all the possible permutations, and pick the one with the least cost.
   - Another base case is reached when the cost of the vertex set on the input permutation is quadratic in the size of the vertex set, up to polynomial in ϵ factors, in which case we use the additive approximation algorithm given in [18], which we can turn into a relative error approximation. We are the first to use such an algorithm in the semi-streaming setting.
   - Otherwise, the algorithm first applies several cost-improving single vertex moves through a method called ApproxLocalImprove, partitions the vertex set into two halves, and then recursively optimizes the two subsets. Note that this fixes the relative order of all (u, v) such that $u \in U, v \in V$ in our final output permutation, where U is the left subset and V is the right subset, and the algorithm will make no further effort to optimize this cost which "crosses" between U and V. However, applying the single vertex moves before recursing on the subsets ensures that this "crossing" cost is not too large.
3. ApproxLocalImprove is our main subroutine and the method through which we find and apply cost-improving single vertex moves to the input permutation:
   - First the algorithm obtains some sample sets from the edge set E. These sample sets are denoted $E_{v,i}$, with v being the vertex being moved, and i representing the i-th sample set for v. The sets $E_{v,i}$ each have size $O(\text{poly}(\log n/\varepsilon))$ and there are also $O(\text{poly}(\log n/\varepsilon))$ such sample sets (i.e., the number of different indices i) per vertex. A more detailed description of these sample sets is given in Section 2.2. Since the algorithm can only use $O^*(n)$ space, where n is the size of the vertex set, the exact cost improvement due to a single vertex cannot be obtained since the entire vertex set E cannot be stored. Therefore, these sample sets $E_{v,i}$ help approximate the cost improvement of single vertex moves. Moreover, these $O(\text{poly}(\log n/\varepsilon))$ sample sets per vertex are independent of the permutation; therefore once some single vertex moves have been made, consulting the approximations provided by $E_{v,i}$, the sample sets $E_{v,j}$ (where $j > i$) are still independent of the resulting permutation after applying those single vertex moves.
   - Multiple single vertex moves need to be made simultaneously in order to achieve the abovementioned $O^*(1)$ sample sets per vertex. To see this, first note that even when k single vertex moves are made in parallel, the cost improvement achieved by them is similar to the sum of the cost improvements that would have been achieved if k moves were made individually. For 2 single vertex moves this is easy to see since any move affects the cost improvement achieved by another move by an additive factor of $O(1)$. Thus, the cost improvement of doing all those moves in parallel is approximated well by the sum of the individual cost improvements. This is also demonstrated with an example in Figure 1. Finally, since the cost of the feedback arc set is bounded by $O(n^2)$, if the total cost improvement achieved by the parallel moves is quadratic in n (up to $\log n$ and ε factors), then the algorithm would only need to make these parallel moves $O^*(1)$ times, justifying the number of sample sets used and showing that the number of passes required for this ApproxLocalImprove subroutine is also $O^*(1)$.

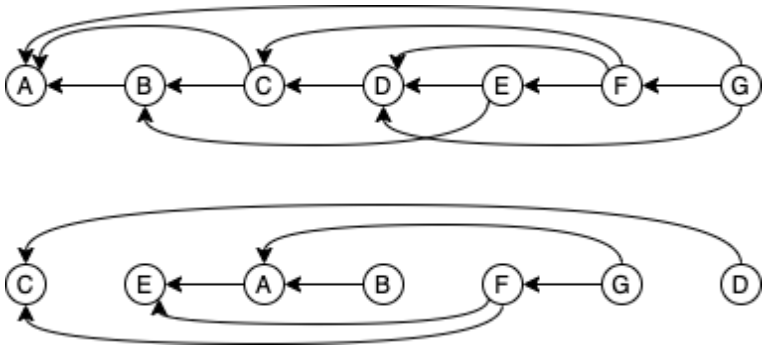

Figure 1 The image on top represents the tournament graph before the single vertex moves $(C \to 1), (E \to 2), (D \to 7)$ ($(E \to 2)$ represents moving vertex E to index 2) are made, and the image on the bottom is the graph obtained after making all the moves in parallel (the graph only displays back edges: any edges not displayed go from left to right). The 3 moves improve the cost by 2, 1, 3 respectively, and the total cost is improved by 5 after making all the moves simultaneously even though the moves overlap.

The above helps achieve $O^*(n)$ space and $\log n$ passes, but in order to obtain the desired result of $O^*(n^{1+1/p})$ space and p passes, multiple passes of the algorithm are emulated in one pass by increasing the size of the sample sets for all vertices. Specifically, the sample sets for the largest nodes in a layer L are obtained first, which are also used to approximate the cost improvement of single vertex moves in the smaller nodes of L. This way, nodes in different layers of the recursion tree can all be handled in the same pass.

2 Feedback Arc Set for Tournament Graphs

Our main algorithm departs from the methods of the earlier streaming algorithm of [11], and is instead inspired by the algorithm of [1] for the query model. However, in order to obtain a streaming algorithm, we need to make a number of crucial modifications to the algorithm of [1]. First, our algorithm executes many single vertex moves in parallel, see lines 5-26 in ApproxLocalImprove below. Second, we define a notion of **realized improvement in cost** to deal with the effects of multiple vertex moves being made in parallel. Third, we need to balance the space complexity of our algorithm by running an additive error algorithm in parallel; see the AddApproxMFAS algorithm below, which we convert to relative error in our context.

2.1 Definitions and Background

▶ **Definition 3.** *A* ***single vertex move*** $(u \to j)$ *on permutation* π *is defined as moving the vertex* u *in* π *to rank* j*, and shifting all the vertices from their original position to their new position accordingly. The resulting permutation is denoted* $\pi_{u \to j}$*. Additionally, if* M *is a set of single vertex moves, then* π_M *represents the permutation obtained after making all moves in parallel on permutation* π*.*

Note that the single vertex move described above is not a vertex swap. Additionally, for a set M of moves, if $(u_1 \to j_1) \in M$ and $(u_2 \to j_2) \in M$ then it must be true that $u_1 \neq u_2$, otherwise the moves cannot be made simultaneously. If multiple vertices were to be moved to the same position, this conflict would be resolved by preserving their original relative ordering.

▶ **Definition 4.** *TestMove*(π, V, W, u, j) *is the improvement in cost of the feedback arc set induced by a permutation* π*, when the single vertex move* $u \to j$ *is conducted to obtain the permutation* $\pi_{u\to j}$*. That is, if without loss of generality* $j \geq \rho_\pi(u)$*, then*

$$\textit{TestMove}(\pi, V, W, u, j) = \sum_{v:\rho_\pi(v)\in[\rho_\pi(u)+1,j]} \big(W(v,u) - W(u,v)\big)$$

Note that TestMove(π, V, W, u, j) can be computed easily for arbitrary inputs π, u, j if $\Theta(n^2)$ space were allowed. However, approximations become necessary to achieve less memory. The following definition provides an unbiased estimator for TestMove(π, V, W, u, j) without observing all the edges of the graph.

▶ **Definition 5.** *TestMove*$_E(\pi, V, W, u, j)$ *is an approximation to the improvement in cost of the feedback arc set from* π *to* $\pi_{u\to j}$*, obtained by only considering the edges in the subset of edges* $\tilde{E} = \{(v,u) \in E : \rho_\pi(v) \in [\rho_\pi(u)+1, j]\}$*. Assuming without loss of generality that* $j \geq \rho_\pi(u)$*, then*

$$\textit{TestMove}_E(\pi, V, W, u, j) = \frac{j - \rho_\pi(u)}{|\tilde{E}|} \sum_{v:(v,u)\in\tilde{E}} \big(W(v,u) - W(u,v)\big)$$

If $E \in \binom{V}{2}$ *is chosen uniformly at random from from all multisets of a given size, then* *TestMove*$_E(\pi, V, W, u, j)$ *acts as an empirical unbiased estimator of* *TestMove*(π, V, W, u, j) *[1].*

As will be seen later, multiple single vertex moves need to be made with respect to the same sample set to obtain the desired space bounds. Making these moves simultaneously affects the cost of other moves being made, which necessitates the notion of **realized cost improvement** for each move. In other words, there needs to be some quantification of how much worse off the cost would be if that move were not made, and this is made explicit by the next definition.

▶ **Definition 6.** *Let* M *be a set of single vertex moves.* *TestMove*$^M(\pi, V, W, u, j)$ *is the* ***realized improvement in cost*** *of the feedback arc set due to the single move* $(u \to j)$ *within* M*. That is,*

$$\textit{TestMove}^M(\pi, V, W, u, j) = C(\pi_M, V, W) - C(\pi_{M\setminus\{(u\to j)\}}, V, W)$$

Other than the above definitions which will be used in the description and analysis of our main algorithm, there is another result that will be useful:

▶ **Definition 7.** *Given a directed graph, the goal of the* ***maximum acyclic subgraph*** *(MAS) problem is to output an ordering* π *on the vertices* V *such that the number of forward edges is maximized (the forward edges making up the acyclic subgraph). This is identical to the MFAS problem since the goal of that problem is to output a permutation such that it minimizes the number of backward edges.*

▶ **Fact 8** ([4], [18])**.** *There exists a randomized algorithm polynomial-time approximation scheme for the MAS problem on tournament graphs. Given* $\beta, \eta > 0$*, the algorithm* AddApproxMAS(V, W, β, η) *outputs, in time* $O\left(n^2 + 2^{O(1/\beta^2)} \log \frac{1}{\eta}\right)$ *in [18]), an ordering* π *on* V *whose cost with respect to* W *is at least* $OPT - \beta n^2$*, with probability at least* $1 - \eta$*, where* OPT *is the size of the largest acyclic subgraph.*

Solving the maximum acyclic subgraph problem and minimum feedback arc set problem on a tournament graph produces the same ordering π. To see this let A, B be the size of the minimum feedback arc set and maximum acyclic subgraph, respectively, due to the ordering π. Then $A + B = \binom{|V|}{2}$. Clearly, minimizing A maximizes B. Therefore the above fact can be rephrased to solve the minimum feedback arc set problem on tournament graphs, that is, there exists an algorithm $\mathsf{AddApproxMFAS}(V, W, \beta, \eta)$ with all of the properties mentioned above which outputs an ordering π such that its cost is at most $OPT + \beta n^2$ with probability at least $1 - \eta$.

As will be seen later, our proposed algorithm uses the AddApproxMFAS algorithm as a subroutine with parameters $\beta = \varepsilon^3$ and $\eta = \frac{1}{n^4}$:

▶ **Theorem 9.** *Given $\beta = \varepsilon^3$ and $\eta = \frac{1}{n^4}$, $\mathsf{AddApproxMAS}(V, W, \beta, \eta)$ can output an ordering with cost at least $OPT - \varepsilon^3 n^2$, with probability at least $1 - \frac{1}{n^4}$, in time $O\left(n^2 + 2^{O(\varepsilon^{-6})} \log n\right)$, using $O^*(n)$ space and $O^*(1)$ passes.*

A brief description of the AddApproxMAS algorithm is given in the full version of the paper, as well as the proof for the above theorem.

Finally, we introduce the main result of [1], which is used as a subroutine in part of our algorithm. To describe the result we first need the following definition:

▶ **Definition 10.** *Given a set V of size n, an ordered decomposition is a list of pairwise disjoint subsets $V_1, ..., V_k \subseteq V$ such that $\bigcup_{i=1}^{k} V_i = V$. If $\Pi(V)$ is the set of all permutations on V, we say that $\pi \in \Pi(V)$ respects $V_1, ..., V_k$ if for all $u \in V_i, v \in V_j, i < j$, we have $\rho_\pi(u) < \rho_\pi(v)$ We denote the set of permutations $\pi \in \Pi(V)$ respecting the decomposition $V_1, ..., V_k$ by $\Pi(V_1, ..., V_k)$. A decomposition $V_1, ..., V_k$ is ε-**good** with respect to W if*

$$\min_{\pi \in \Pi(V_1, ..., V_k)} C(\pi, V, W) \leq (1 + \varepsilon) \min_{\pi \in \Pi(V)} C(\pi, V, W)$$

▶ **Fact 11** ([1])**.** *Given a vertex set V, a weight matrix W (describing the edges of the graph), and an error tolerance parameter $0 < \varepsilon < 1$, there exists a polynomial time algorithm which returns, with constant probability, an ε-good partition of V, querying at most $O(\varepsilon^{-6} n \log^5 n)$ locations in W in expectation. The running time of the algorithm is $O(n \ poly(\log n, \varepsilon^{-1}))$. This algorithm is referred to as* SampleAndRank*.*

Note that the above result does not output a permutation π with near-optimal cost: it outputs a partition $(V_1, ..., V_k)$ of V such that the optimal permutation that respects the partition is near-optimal globally. As it turns out, it is easy to obtain near-optimal permutations on the individual V_i because

1. They are small enough to enumerate over all permutations, or
2. Their optimal cost is high enough that AddApproxMFAS outputs a good approximation.

We now describe the algorithm in detail.

2.2 Formal Description of Algorithm

We now state our algorithms and subroutines, providing a concise English description and pseudocode for each. Afterwards, we give our formal analysis.

Algorithm 1 GetNearOptimalPermutation(V, W, ε): Top level function for computing our $(1+\varepsilon)$-approximation to the minimum feedback arc set. It obtains an $O(1)$-approximation to the minimum feedback arc set, and then calls Recurse. The latter recursively executes single vertex moves to bring down the approximation factor to $(1+\varepsilon)$.

$n \leftarrow |V|$
$\pi \leftarrow O(1)$-approximation by sorting by indegree as described by Coppersmith et al. [15]
return Recurse$(V, W, \varepsilon, n, \pi)$

Algorithm 2 Recurse$(V, W, \varepsilon, n, \pi)$: Recursive part of the algorithm that optimizes the current vertex set and then recursively optimizes partitions of the vertex set. The base case of the recursion is reached when the vertex set is small enough to enumerate over all possible $N!$ permutations. The other base case is reached when the cost of the vertex set is quadratic in N, which is when the AddApproxMFAS algorithm will be effective. If a base case is not reached, then the algorithm calls ApproxLocalImprove to perform cost-improving single vertex moves before recursing on the two subsets of V. Note that this procedure is similar to the RecurseSAR procedure given by Ailon [1], except in the base cases their algorithm returns the trivial partition $\{V\}$ since they only need to output an ε-good partition.

$N \leftarrow |V|$
if $N \leq \log n / \log\log n$ **then**
    $\pi^* \leftarrow$ Minimizer for V, W obtained by brute force
    return π^*
end if
$E \leftarrow$ random subset of $O(\varepsilon^{-4}\log n)$ elements from $\binom{V}{2}$ (with repetition)
$C \leftarrow C_E(\pi, V, W)$ (C is an additive $O(\varepsilon^2 N^2)$ approximation of $C(\pi, V, W)$)
if $C = \Omega(\varepsilon^2 N^2)$ **then**
    return AddApproxMFAS$(V, W, \varepsilon^3, 1/n^4)$
end if
$\pi_1 \leftarrow$ ApproxLocalImprove$(V, W, \varepsilon, n, \pi)$
$k \leftarrow$ uniformly random integer in the range $[N/3, 2N/3]$
$V_L \leftarrow \{v \in V : \rho_\pi(v) \leq k\}, \pi_L \leftarrow$ restrict π_1 to V_L
$V_R \leftarrow \{v \in V : \rho_\pi(v) > k\}, \pi_R \leftarrow$ restrict π_1 to V_R
return Concatenate Recurse$(V_L, W, \varepsilon, n, \pi_L)$ and Recurse$(V_R, W, \varepsilon, n, \pi_R)$

Algorithm 3 GetMoves$(V, W, \varepsilon, n, \pi, \{E_{v,i} : v \in V\}, c)$: Helper function used by ApproxLocalImprove to obtain cost-improving single vertex moves on π. The log length of the single vertex move $(\log\lceil|j - \rho_\pi(u)|\rceil)$ must lie between B and L, which are both logs of terms linear in N. Moreover, the cost improvement as approximated by the sample sets $\{E_{v,i} : v \in V\}$ must be at least $c|j - \rho_\pi(u)|\varepsilon/\log n$.

$B \leftarrow \lceil\log(\Theta(\varepsilon N/\log n))\rceil, L \leftarrow \lceil\log N\rceil$
return $\{(u \to j) : u \in V$ and $j \in [n]$ and [using $d = |j - \rho_\pi(u)|$ and $l = \lceil\log d\rceil$]
        $l \in [B, L]$ and TestMove$_{E_{u,i}}(\pi, V, W, u, j) > cd\varepsilon/\log n\ \}$

Algorithm 4 ApproxLocalImprove$(V, W, \varepsilon, n, \pi)$: Function to repeatedly apply cost-improving single vertex moves. This optimizes the permutation on V as long as there exists a single vertex move with linear cost improvement (up to $\log n$ and ε factors). No cost-improving single vertex moves need to be made if V is small enough. Otherwise, on lines 5-13 the algorithm first obtains sample sets $\{E_{v,i}\}$, where $\{E_{v,i} : v \in V\}$ and $\{E_{v,j} : v \in V\}$ are independent and identically distributed for all $i \neq j$. Once the samples are obtained, in lines 15-25 they are used to perform significant cost-improving single vertex moves until no such moves exist. Section 2.3 discusses why the algorithm succeeds in doing so with only $\text{poly}(\log n/\varepsilon)$ iterations.

1: $N \leftarrow |V|, B \leftarrow \lceil \log(\Theta(\varepsilon N/\log n)) \rceil, L \leftarrow \lceil \log N \rceil$
2: **if** $N = O(\varepsilon^{-3} \log^3 n)$ **then**
3: **return** π
4: **end if**
5: **for** $v \in V$ **do**
6: **for** $i \in [1, \Theta(\varepsilon^{-6} \log^6 n)]$ **do**
7: $E_{v,i} \leftarrow \phi$
8: **for** $m \in [1, \Theta(\varepsilon^{-5} \log^5 n)]$ **do**
9: $j \leftarrow$ uniformly random integer chosen from $[1, N]$
10: $E_{v,i} \leftarrow E_{v,i} \cup \{(v, \pi(j)\}$
11: **end for**
12: **end for**
13: **end for**
14: $i \leftarrow 1$
15: **while** $|S| > 0$ such that $S =$ GetMoves$(V, W, \varepsilon, n, \pi, \{E_{v,i}\}, 1)$ **do**
16: $M_1 \leftarrow$ GetMoves$(V, W, \varepsilon, n, \pi, \{E_{v,i}\}, \frac{1}{2})$
17: **for** $m \in [1, \Theta(\varepsilon^{-2} \log^2 n)]$ **do**
18: $M_2 \leftarrow \{(u \to j) \in M_1 : l \in [B, L]$ and $\text{TestMove}_{E_{u,i}}(\pi, V, W, u, j) > \frac{1}{2}\varepsilon d/\log n\}$
19: [using $d = |j - \rho_\pi(u)|$, $l = \lceil \log d \rceil$,
20: d is the 'length' of the single vertex move $(u \to j)$ on π]
21: $M_3 \leftarrow$ Sample $\Theta(\varepsilon^2 \log^{-2} n)$-fraction of M_2 (All u's must be unique, otherwise cannot move simultaneously)
22: $\pi \leftarrow \pi_{M_3}$
23: $i \leftarrow i + 1$ ($E_{u,i}$ is no longer independent of π, so need a fresh set of samples)
24: **end for**
25: **end while**
26: **return** π

Our main algorithm along with the required subroutines is shown above, and our corresponding theorem is formalized below.

▶ **Theorem 12.** *The algorithm* GetNearOptimalPermutation*:*

i *returns a* $(1+\varepsilon)$*-approximation to the Minimum Feedback Arc Set problem on tournaments with constant probability in time* $O\left(poly(n)2^{poly(1/\varepsilon)}\right)$,
ii *requires at most* $O^*(n)$ *space if executed in* $O(\log n)$ *passes, and*
iii *can be executed in* p *passes and* $O^*(n^{1+1/p})$ *space.*

The top level algorithm GetNearOptimalPermutation first obtains a 5-approximation by sorting by indegree, which can be achieved by storing the indegree of all vertices [15]. It then calls Recurse. The base case occurs when the vertex set becomes small enough to use brute force, or when the cost of the minimum feedback arc set becomes large enough to approximate with AddApproxMFAS (which is the source of the $O\left(2^{\text{poly}(1/\varepsilon)}\right)$ factor in the time). Given this recursion tree and the output permutation π^O, we can divide the cost of the output permutation into two sources:

1. Cost incurred at the internal nodes $\mathcal{I}$: Let $X \in \mathcal{I}$ be an internal node of the recursion tree, and let L, R be the left and right child of X. Once Recurse has divided V_X into V_L and V_R, for any $u \in V_L, v \in V_R$, it will be the case that $\rho_{\pi^O}(u) < \rho_{\pi^O}(v)$ where π is the output permutation: their relative ordering is now fixed and will never be changed. If $W(v,u) = 1$, then this is a permanent cost that will be attributed to the parent X, which we can quantify as

$$\beta_X = \sum_{u \in V_L, v \in V_R} W(v,u)$$

2. Cost incurred at the leaves $\mathcal{L}$: Let $X \in \mathcal{L}$ be a leaf of the recursion tree. The cost incurred by the leaf is given by

$$\alpha_X = \sum_{u,v \in V_X : \rho_{\pi^O}(u) < \rho_{\pi^O}(v)} W(v,u)$$

Using the above notation, it is easy to see that $C(\pi^O, V, W) = \sum_{X \in \mathcal{I}} \beta_X + \sum_{X \in \mathcal{L}} \alpha_X$.

Note that SampleAndRank [1] is similar to the GetNearOptimalPermutation: the top level function obtains an $O(1)$ approximation, and then cost-improving single vertex moves are applied recursively. The structure of recursion in SampleAndRank is identical to the one in Recurse, except SampleAndRank returns the trivial partition $\{V\}$ in the base cases, whereas Recurse optimizes the cost of the base cases using either brute force or AddApproxMFAS. However, the procedure used by the two algorithms to find cost-improving single vertex moves (ApproxLocalImprove) is very different. A detailed description of the SampleAndRank algorithm is given in the full version of the paper.

One difference that stands out in particular between the two local optimization procedures is how the sample sets $E_{v,i}$ are defined. In the lines 5-13 of ApproxLocalImprove, a sample set $E_{v,i}$ consists of $\Theta(\varepsilon^{-5} \log^5 n)$ edges that start at v and end at a uniformly random vertex in the input set V. If $i \neq j$, then $E_{v,i}$ and $E_{v,j}$ are independent and identically distributed random sets. This particular sampling scheme lets the algorithm get away with $O^*(1)$ number of passes and $O^*(1)$ space within one call of ApproxLocalImprove.

Intuitively, given the above similarities between SampleAndRank and GetNearOptimalPermutation, any statement that is proved by [1] for SampleAndRank about β_X for $X \in \mathcal{I}$, should hold for GetNearOptimalPermutation as well, as long as ApproxLocalImprove makes all single vertex moves that SampleAndRank's local optimization procedure would make. The key lemma in the query model from [1] that is useful is the following:

▶ **Lemma 13.** *[1] Let C^* be the cost of the optimal permutation, and C^*_X be the optimal permutation when V, W are restricted to the vertices in X. Then* SampleAndRank *guarantees*

$$E\left[\sum_{X \in \mathcal{I}} \beta_X\right] \leq (1+\varepsilon)C^* - E\left[\sum_{X \in \mathcal{L}} C^*_X\right]$$

where the expectation is taken over the choice of the pivots k (line 12, Algorithm 2: Recurse*).*

Again, in order to use the above lemma, an equivalence between the terminating condition of ApproxLocalImprove and the terminating condition of SampleAndRank's local optimization procedure must be shown: if ApproxLocalImprove outputs π, then there cannot exist a **strong single vertex move** on π anymore. A strong single vertex move $(u \to j)$ has the following two properties:

- It is long : $|j - \rho_\pi(u)| \geq \Theta(\varepsilon N / \log n)$, and
- It is effective : $\text{TestMove}(\pi, V, W, u, j) \geq \Theta(\varepsilon |j - \rho_\pi(u)| / \log n)$.

As long as there is no single vertex move satisfying both the above properties, the algorithm would obey the terminating condition of SampleAndRank's local optimization procedure. Proving this statement nearly establishes the first part of Theorem 12, and that is the key objective of the following section.

2.3 Proof of Correctness

Let $d = |j - \rho_\pi(u)|$ and $l = \lceil \log d \rceil$ for the single vertex move $(u \to j)$ on permutation π, and let α be an arbitrary integer in the range $[1, \Theta(\varepsilon^{-6} \log^6 n)]$. The lemmas shown below outline the structure of the justification for the first part of Theorem 12, along with their proofs. The following is a brief overview of the proof:

- When moves are made in parallel, and approximated using the samples $E_{v,i}$ (generated in lines 5-13 of Algorithm 4 : ApproxLocalImprove), we want to ensure that the realized cost of a single move is at least $\Theta(\varepsilon^2 N / \log^2 n)$. Lemmas 14, 15, 16, 17 show that the samples approximate the cost improvement of these moves well and all approximations hold with constant probability. Lemmas 18 and 19 show that strong single vertex moves have a high realized cost improvement, even if they are made in parallel and approximated using samples.
- On a vertex set of size N, the maximum feedback arc set size can be $O(N^2)$. If a move is strong, its realized cost improvement is $\Theta(\varepsilon^2 N / \log^2 n)$ as we saw. Now, it is possible that by making all these moves, we create the opportunity for a new strong single vertex move that did not exist earlier. However, lemmas 20 and 21 show that the overall cost improvement needed to create these new strong single vertex moves is $\Omega(\varepsilon^4 N^2 / \log^4 n)$.
- Given that the maximum cost of a feedback arc set is $\Theta(N^2)$, these new strong moves can only be created $O(\varepsilon^{-4} \log^4 n)$ times. Therefore, as long as the loop on line 15 in ApproxLocalImprove, there will be no strong single vertex moves left, and the local optimization can safely end. This is shown in lemmas 22 and 23.
- Finally, in the next section it is formally shown why satisfying the terminating condition of "no strong single vertex moves remaining" is sufficient to finish the proof.

We can now delve into the details of this proof.

▶ **Lemma 14** (Ailon [1]). *Let $E \subseteq \binom{V}{2}$ be a random multi-set of size m with elements $(u, v_1), \ldots, (u, v_m)$ such that for each vertex v_i, its rank $\rho_\pi(v_i)$ is between $\rho_\pi(u)$ (exclusive) and j (inclusive) and is chosen uniformly at random in this range. Then*

$$\mathbf{E}[\mathit{TestMove}_E(\pi, V, W, u, j)] = \mathit{TestMove}(\pi, V, W, u, j)$$

and additionally, for any $\delta > 0$, with probability of failure δ it holds that

$$|\mathit{TestMove}_E(\pi, V, W, u, j) - \mathit{TestMove}(\pi, V, W, u, j)| = O\left(d\sqrt{\frac{\log \delta^{-1}}{m}}\right)$$

Proof. This follows by a Hoeffding bound and the fact that $|W(u, v)| \leq 1$ for all u, v. ◀

▶ **Lemma 15.** *Given a single vertex move $(u \to j)$ on permutation π and sample set $E_{u,\alpha}$, $\mathit{GetSample}(\pi, E_{u,\alpha}, u, j)$ has at least $\Omega(\varepsilon^{-4} \log^4 n)$ elements that are drawn uniformly at random between $\rho_\pi(u)$ (exclusive) and j (inclusive), with $1 - O(n^{-6})$ probability of success.*

Proof. Follows from Chernoff bound for binomial random variables. ◀

The above two lemmas are useful for proving that these samples $E_{v,\alpha}$ can be properly utilized to approximate the cost improvement of single vertex moves, as is shown by the next lemma.

▶ **Lemma 16.** *Let $\hat{E} = GetSample(\pi, E_{u,i}, u, j)$. With probability of success $1 - O(n^{-6})$,*

$$|TestMove_{\hat{E}}(\pi, V, W, u, j) - TestMove(\pi, V, W, u, j)| \leq \frac{1}{8} d\varepsilon / \log n$$

Proof. By Lemma 15 we have $|\hat{E}| = \Omega(\varepsilon^{-4} \log^4 n)$ with at least $1 - O(n^{-6})$ probability. Setting the probability of failure $\delta = e^{-\varepsilon^{-2} \log^2 n}$, by Lemma 14 we get

$$|\text{TestMove}_{\hat{E}}(\pi, V, W, u, j) - \text{TestMove}(\pi, V, W, u, j)| = O\left(d\sqrt{\frac{\varepsilon^{-2}\log^2 n}{\varepsilon^{-4}\log^4 n}}\right) = O(d\varepsilon/\log n)$$

Note that δ can be upper bounded by $O(n^{-6})$. Therefore the claim is true with probability at least $1 - O(n^{-6})$. ◀

The high probability guarantee of a decent approximation for a single vertex move must be extended to all single vertex moves, across all nodes, with a constant probability, in order to prove that the overall algorithm works with constant probability (say 0.99).

▶ **Lemma 17.** *All sampling approximations of TestMove in Lemma 16 succeed simultaneously with constant probability.*

Proof. There are $O(n)$ many calls to ApproxLocalImprove with high probability. Since the loop on line 15 in ApproxLocalImprove can be executed at most $O(n^2)$ times and needs to correctly approximate $O(n^2 \operatorname{polylog} n/\varepsilon)$ possible moves, the total number of TestMove approximations we need is $O(n^5 \operatorname{polylog} n/\varepsilon)$. Using a union bound on Lemma 16, the approximations hold with an arbitrary constant probability (say 0.99). ◀

Being able to make one single vertex move using few samples is useful, but in order to obtain $O(\text{poly}(\log n/\varepsilon))$ many samples for each vertex, multiple single vertex moves need to be performed at a time. If two moves are made simultaneously, they affect each other's cost only by $O(1)$, which makes the following lemma possible.

▶ **Lemma 18.** *Let $TestMove^M(\pi, V, W, u, j)$ be the realized cost of the single vertex move $u \to j$ when all the moves in M are made in parallel. Then*

$$|TestMove^M(\pi, V, W, u, j) - TestMove(\pi, V, W, u, j)| < \frac{1}{8} d\varepsilon / \log n$$

Proof. Doing two single vertex moves $u \to j$ and $v \to i$ (such that $u \neq v$) simultaneously affects the cost of another move by $\Theta(1)$. Therefore doing $\Theta(\varepsilon N/\log n)$ moves simultaneously (line 22, ApproxLocalImprove) affects the cost of a single vertex move $(u \to j)$ by $\Theta(\varepsilon^2 N/\log^2 n)$. Since $l \geq B$, we get $d \geq \Theta(\varepsilon N/\log n)$, which implies that the realized cost improvement of $(u \to j)$ would be changed by $O(\varepsilon d/\log n)$. ◀

The above lemmas are critical in demonstrating that only $O(\text{poly}(\log n/\varepsilon))$ sample sets per vertex are required. In the following, Lemma 17 and Lemma 18 are used as a base case to show that every move made achieves a significant cost improvement. Since the maximum feedback arc set size is $O(N^2)$, this leads to a bound on the number of moves that need to be made.

▶ **Lemma 19.** *Each move made in* ApproxLocalImprove *has a realized cost improvement of* $\Omega(\varepsilon d/\log n)$.

Proof. Follows from Lemma 17, Lemma 18 and the triangle inequality. ◀

▶ **Lemma 20.** $\Omega(\varepsilon d/\log n)$ *single vertex moves are required to create new moves* $(u \to j)$ *such that* $\mathit{TestMove}(\pi, V, W, u, j) > \frac{7}{8}\varepsilon d/\log n$.

Proof. Any single vertex move $(u \to j)$ such that $\text{TestMove}(\pi, V, W, u, j) > \frac{5}{8}\varepsilon d/\log n$ will be included in M_1 and eventually be made unless its cost improvement goes below $\frac{5}{8}\varepsilon d/\log n$. Each single vertex move changes the cost of another move by $O(1)$. Therefore in order to create a new single vertex move $u \to j$ such that $\text{TestMove}(\pi, V, W, u, j) > \frac{7}{8}\varepsilon d/\log n$, we will have to make $\Omega(\varepsilon d/\log n)$ moves. ◀

These demonstrate that every move made contributes to a substantial cost improvement, and yet several moves need to be made in order to create a new one of significant cost improvement. Therefore, the total cost improvement that is observed before a new single vertex move of significant cost improvement is created will be quite high, as is demonstrated below.

▶ **Lemma 21.** *The total cost of the feedback arc set needs to be decreased by* $\Omega(\varepsilon^4 N^2/\log^4 n)$ *to create new strong moves* $u \to j$ *for which* $\mathit{TestMove}(\pi, V, W, u, j) > \frac{7}{8}\varepsilon d/\log n$.

Proof. By Lemma 20 and the fact that $d > \Theta(\varepsilon N/\log n)$, we need to make $\Omega(\varepsilon^2 N/\log^2 n)$ many single vertex moves to create such a new move. Since each single vertex move we make in the algorithm is such that its realized cost improvement is $\Omega(\varepsilon d/\log n)$ (by Lemma 19), and that $d > \Theta(\varepsilon N/\log n)$, the cost improvement due to each move is $\Omega(\varepsilon^2 N/\log^2 n)$. The lemma follows. ◀

Thus, the cost of the feedback arc set is reduced heavily before new strong single vertex moves are created. However, since the feedback arc set cost induced by any permutation is $O(N^2)$, the algorithm does not need to check if such a new move has been created too many times, which leads us to the next two lemmas.

▶ **Lemma 22.** *The loop on line 15 in* ApproxLocalImprove *has* $O(\varepsilon^{-4}\log^4 n)$ *iterations.*

Proof. Follows from Lemma 21 and the fact that the maximum cost of a FAS is $\binom{N}{2}$. ◀

▶ **Lemma 23.** *The counter* i *on line 14 in* ApproxLocalImprove *can increase by* $O(\varepsilon^{-6}\log^6 n)$.

Proof. In ApproxLocalImprove, the sampling on line 21 is done in such a way that every move is selected exactly once (unless the move fails the check on lines 18-20). Therefore the number of iterations required to exhaust all moves in M_1 is $O(\varepsilon^{-2}\log^2 n)$, as described on line 17. Using Lemma 22, we conclude that the total number of samples required per vertex is $O(\varepsilon^{-6}\log^6 n)$. ◀

In this algorithm, each move that is made has positive cost improvement, as shown by Lemma 19. Since only moves $(u \to j)$ such that $\text{TestMove}_{E_{u,\alpha}}(\pi, V, W, u, j) > \varepsilon d/\log n$ are executed, the terminating condition of the loop on line 15 in ApproxLocalImprove is the same as the terminating condition in SampleAndRank's version of ApproxLocalImprove. Now by the results in the query model [1], a $(1+\varepsilon)$ approximation is obtained and the proof of the first part of Theorem 12 is concluded. A more detailed justification is given in the following subsection.

2.4 Proof of main theorem

Algorithm 5 ApproxLocalImproveSAR$(V, W, \varepsilon, n, \pi)$ [1]: The local optimization procedure for SampleAndRank by Ailon [1].

1: $N \leftarrow |V|, B \leftarrow \lceil \log(\Theta(\varepsilon N/\log n)) \rceil, L \leftarrow \lceil \log N \rceil$
2: **if** $N = O(\varepsilon^{-3} \log^3 n)$ **then**
3: **return** π
4: **end if**
5: **for** $v \in V$ **do**
6: $r \leftarrow \rho_\pi(v)$
7: **for** $i \in [B, L]$ **do**
8: $E_{v,i} \leftarrow \phi$
9: **for** $m \in [1, \Theta(\varepsilon^{-2} \log^2 n)]$ **do**
10: $j \leftarrow$ integer uniformly at random chosen from $[\max\{1, r - 2^i\}, \min\{n, r + 2^i\}]$
11: $E_{v,i} \leftarrow E_{v,i} \cup \{(v, \pi(j)\}$
12: **end for**
13: **end for**
14: **end for**
15: **while** $\exists u \in V$ and $j \in [n]$ s.t. (where $l = \lceil \log |j - \rho_\pi(u)| \rceil$) : $l \in [B, L]$ and $\text{TestMove}_{E_{u,l}}(\pi, V, W, u, j) > \varepsilon |j - \rho_\pi(u)| / \log n$ **do**
16: **for** $v \in V$ and $i \in [B, L]$ **do**
17: refresh sample $E_{v,i}$ with respect to the move $(u \to j)$
18: **end for**
19: $\pi \leftarrow \pi_{u \to j}$
20: **end while**
21: **return** π

Proof of Theorem 12i. As can be seen in Algorithm 5: ApproxLocalImproveSAR, the algorithm keeps making local optimizations on π as long as there exists a single vertex move $(u \to j)$ that satisfies the following properties (where $l = \lceil \log |j - \rho_\pi(u)| \rceil$):

1. $l \in [B, L]$
2. $\text{TestMove}_{E_{u,l}}(\pi, V, W, u, j) > \varepsilon |j - \rho_\pi(u)| / \log n$

Intuitively, at the end of the optimization, there are no 'long' moves left which have a significant cost improvement (as estimated by the ensemble). Since the total number of moves that can be made by the algorithm is $O(n^2)$ and there are $O(n^2)$ many sample ensembles $E_{u,l}$, Ailon's algorithm needs the approximation used on line 15 of ApproxLocalImproveSAR to be good with probability $1 - O(n^{-4})$. Therefore, Ailon proves the following lemma:

▶ **Lemma 24** (Ailon [1]). *Any sample ensemble $\mathcal{S}$ used in* ApproxLocalImproveSAR *is a good approximation with probability $1 - O(n^{-4})$. A good approximation satisfies the following two properties for all $u \in V$ and $j \in [n]$ such that $\log |j - \rho_\pi(u)| \geq B$ (let $l = \lceil |j - \rho_\pi(u)| \rceil$):*

1. $|\{x : (u, x) \in E_{u,l}\} \cap \{x : \rho_\pi(x) \in [\rho_\pi(u), j]\}| = \Omega(\varepsilon^{-2} \log^2 n)$
2. $|\mathit{TestMove}_{E_{u,l}}(\pi, V, W, u, j) - \mathit{TestMove}(\pi, V, W, u, j)| \leq \frac{1}{2} \varepsilon |j - \rho_\pi(u)| / \log n$

Using the second property above, line 15 of ApproxLocalImproveSAR, and the triangle inequality, it can be seen that when the while loop on line 15 terminates, all 'long' moves $(u \to j)$ left have a cost improvement of $O(\varepsilon |\rho_\pi(u) - j| / \log n)$ where $u \in V$ and $j \in [n]$.

Here, 'long' refers to $l \in [B, L]$ where $l = \lceil |j - \rho_\pi(u)| \rceil$. In other words,

$$\forall u \in V, j \in [n] \text{ s.t. } \lceil |j - \rho_\pi(u)| \rceil \in [B, L] : \\ \text{TestMove}(\pi, V, W, u, j) = O(\varepsilon |\rho_\pi(u) - j| / \log n) \quad (1)$$

Now consider any recursive call of ApproxLocalImproveSAR, and let π_V^1 be the output of the call when restricted to the vertex set V. Similarly, let π_V^* be the optimal permutation when the problem is restricted to only V (using the weight submatrix W_V). Since $(u \to \pi_V^*(u))$ is a possible move that could be made for any $u \in V$, the result in the previous paragraph can be therefore weakened to

$$\text{TestMove}(\pi_V^1, V, W_V, u, \pi_V^*(u)) = O(\varepsilon |\rho_{\pi_V^1}(u) - \rho_{\pi_V^*}(u)| / \log n) \quad (2)$$

where $u \in V$ is such that $|\rho_{\pi_V^1}(u) - \rho_{\pi_V^*}(u)| = \Omega(\varepsilon N / \log n)$.

The above is precisely the post-condition that is used in ApproxLocalImproveSAR to complete the proof of Lemma 13 mentioned in the main text. Since ApproxLocalImprove also satisfies Equation 1 on termination due to Lemma 19, it also satisfies Equation 2. Therefore Lemma 13 can be used for GetNearOptimalPermutation. Let C'_X be the cost of the at a leaf X before it is optimized with AddApproxMFAS, so we get

$$\begin{aligned} E[C(\pi^O, V, W)] &= E\left[\sum_{X \in \mathcal{I}} \beta_X\right] + E\left[\sum_{X \in \mathcal{L}} \alpha_X\right] \\ &\leq (1 + O(\varepsilon))C^* - E\left[\sum_{X \in \mathcal{L}} C_X^*\right] + E\left[\sum_{X \in \mathcal{L}} \alpha_X\right] && \text{[Lemma 13]} \\ &\leq (1 + O(\varepsilon))C^* + \sum_{X \in \mathcal{L}} \varepsilon^3 |V_X|^2 && \text{[AddApproxMFAS]} \\ &\leq (1 + O(\varepsilon))C^* + \sum_{X \in \mathcal{L}} O(\varepsilon C'_X) && \text{[Line 8, Algorithm 2: Recurse]} \\ &\leq (1 + O(\varepsilon))C^* \end{aligned}$$

as desired. Finally, it is easy to see that the recursive part of the algorithm (including the single vertex moves), all can be executed in $O\left(\text{poly}(n, \varepsilon^{-1})\right)$ time. However, since we call AddApproxMFAS with an error parameter of ε^3, the base cases require $O(n^2 + 2^{\text{poly}(1/\varepsilon)} \log n)$ time (using $\beta = \varepsilon^3$ and $\eta = 1/n^4$ in Fact 8). Since there are at most n different base cases, the total time required is $O\left(\text{poly}(n) 2^{\text{poly}(1/\varepsilon)}\right)$ as desired. ◀

The next sections focus on the number of passes and space required in order to prove the second and third parts of Theorem 12.

2.5 Number of Passes And Space

▶ **Lemma 25.** *A call to* ApproxLocalImprove *requires 1 pass and uses* $O(N \cdot \text{poly}(\log n/\varepsilon))$ *space.*

Proof. Storing M_1 as defined in line 16 in ApproxLocalImprove can be in done with $O(N \log n)$ space since the algorithm only needs to store one single vertex move $(u \to j)$ for every vertex $u \in V$. The lemma follows. ◀

Recurse induces a tree where each node corresponds to a call to ApproxLocalImprove. This tree has $O(\log n)$ depth with high probability and two nodes at the same depth can be executed in the same pass. In addition, the total space used by nodes on the same layer is

$\sum_{i \in L} O(N_i \text{ poly}(\log n/\varepsilon)) = O(n \text{ poly}(\log n/\varepsilon))$ by Lemma 25 as desired. Finally, note that the algorithm mentioned on line 2 in GetNearOptimalPermutation can be executed in 1 pass and $O(n \log n)$ space, concluding the justification of the second part of Theorem 12.

The pass-space trade-off described by Theorem 12iii therefore remains to be shown. Given $O^*(n^{1+1/p})$ space, multiple passes of the original algorithm can be emulated in one pass. Partition the layers of the recursion tree into p levels, where level i contains nodes of size N such that

$$\Theta(n^{1-\frac{i}{p}}) \leq N \leq \Theta(n^{1-\frac{i-1}{p}}).$$

The key idea is that samples $E_{v,\alpha}$ for all v in nodes situated in the same level can be computed simultaneously. Once reliable samples that satisfy Lemma 15 are obtained, the proof from Lemma 16 onwards follows, and ApproxLocalImprove can be called for all nodes in the same pass.

Note that condensing the layers of the tree into p levels allows for the algorithm to use p passes, so all that needs to be shown is that a sample size of $O(n^{1/p} \text{ poly}(\log n/\varepsilon))$ per vertex is sufficient to prove Lemma 15. Proceeding in a similar fashion to the proof of Lemma 15, consider an arbitrary node X_1 in level η. Then

$$\Theta(n^{1-\frac{\eta}{p}}) \leq |X_1| \leq \Theta(n^{1-\frac{\eta-1}{p}})$$

Now consider the highest ancestor X_2 of X_1 such that $|X_2| \leq \Theta(n^{1-\frac{\eta-1}{p}})$. This is the node for which the samples $E_{u,\alpha}$, for every $u \in X_2$ to be used for the nodes in the level η, are obtained.

Let $(u \to j)$ be the single vertex move desired in X_1, and $Y = |\text{GetSample}(\pi, E_{u,\alpha}, u, j)|$. Let Y_i be an indicator for which $Y_i = 1$ if and only if the i-th sample $(u, v_i) \in E_{u,\alpha}$ is such that v_i is in the range of the single vertex move $(u \to j)$. Therefore $Y = \sum Y_i$, and

$$\mathbf{E}[Y] = \Theta\left(|X_2|^{-1}\varepsilon^{-5} d n^{1/p} \log^5 n\right) \geq \left(\frac{|X_1|}{|X_2|}\varepsilon^{-4} n^{1/p} \log^4 n\right) \geq \Theta\left(\varepsilon^{-4} \log^4 n\right)$$

Using a Chernoff bound for binomial random variables yields

$$\mathbb{P}(Y \leq \Theta(\varepsilon^{-4} \log^4 n)) \leq e^{-\Theta(\varepsilon^{-4} \log^4 n)} \leq \Theta(n^{-6})$$

as desired. Note that since all elements were picked uniformly at random with repetition in $E_{u,\alpha}$, they are still uniformly random in the required range.

2.6 Space-Time Trade-off for 1-pass Algorithms

The idea used earlier to achieve the pass-space trade-off for polynomial time algorithms was to emulate multiple passes of the algorithm in one pass by increasing the sampling rate per vertex. However, it can also be used to achieve a space-time trade-off for 1 pass algorithms: we can use our samples to emulate as many passes as possible in one pass. Once the node size becomes too small to approximate with the samples, we can solve each block by using an exponential time $(1 + \varepsilon)$ approximation algorithm similar to the one by Chakrabarti et al. [11].

▶ **Fact 26.** *There is a single pass algorithm, such that given a stream of poly(n) integer updates in the range $[-\text{poly}(n), \text{poly}(n)]$ to an underlying vector $\mathbf{x} \in \mathbb{R}^N$, uses $O(\varepsilon^{-2} \log \delta^{-1} \log n)$ bits of memory and maintains a sketch $S \cdot x$ for a matrix S of $d = O(\varepsilon^{-2} \log \delta^{-1})$ dimensions. From $S \cdot x$, one can output a $(1 \pm \varepsilon)$-approximation to $\|x\|_1$ with probability at least $1 - \delta$. [21]*

▶ **Fact 27.** *There is a single pass algorithm for the Minimum Feedback Arc Set problem on tournament graphs that uses $O(\varepsilon^{-2} n \log^2 n)$ space and returns a $(1+\varepsilon)-$approximation with probability at least $\frac{2}{3}$. [11]*

Using the above two facts, we give the following trade-off:

▶ **Theorem 28.** *There exists an algorithm which can solve the Minimum Feedback Arc Set problem on tournament graphs up to a $(1+\varepsilon)-$approximation in a single pass with $O^*(n^{2-\gamma})$ space and $O\left(2^{\Theta(n^\gamma \log n)} 2^{poly(1/\varepsilon)}\right)$ time, for any $0 < \gamma < 1$.*

Proof. We can divide the layers of the recursion tree into 2 parts: Part 1 consists of layers where all nodes X are such that $|X| \geq \Theta(n^\gamma)$ and Part 2 consists of the remaining layers. Note that by increasing the sampling rate to $\Theta(n^{1-\gamma} \operatorname{poly}(\log n/\varepsilon))$ per vertex at the root node of the recursion tree, we can successfully emulate all the passes for the layers in Part 1.

We also make another key observation here: let $V_1, ..., V_m$ be the top layer of Part 2, so we can divide the cost into two parts:

$$C(\pi, V, W) = C_{int} + C_{ext}$$

$$C_{int} = \sum_{i\in[m]} \sum_{u,v\in V_i : \rho_\pi(u) < \rho_\pi(v)} W(u,v)$$

$$C_{ext} = \sum_{i,j\in[m]: i<j} \sum_{(u,v)\in(V_i,V_j)} W(u,v)$$

where C_{int} is the internal cost within the blocks, and C_{ext} is the external cost across different blocks.

No matter what optimization we do in Part 2, C_{ext} will remain the same since the relative ordering of two vertices in two different blocks will not change. Therefore we must have that $C_{ext} \leq (1+\varepsilon)C^*$, otherwise there is no possibility that our original algorithm or Ailon's [1] algorithm would achieve a $(1+\varepsilon)$-approximation.

Therefore all that needs to be shown is that $C_{int} \leq (1+\varepsilon)C^*$. Since all vertex sets $V_1, \ldots, V_m$ are disjoint, we can non-adaptively optimize them independently of each other. This is where Fact 26 is used. Despite the fact that the ℓ_1-sketch only needs $O(\varepsilon_0^{-2} \log \delta^{-1} \log n)$ bits of space, where ε_0 is the accuracy of the sketch and δ is the probability of error, their algorithm needs to brute force over all $\Theta(n!)$ possible permutations to find the best permutation, which means that $\delta = \Theta(1/n!)$ because of the use of a union bound. However, in our algorithm we only need to union bound over $\Theta((n^\gamma!) \cdot n^{1-\gamma})$ permutations since we can separately (and non-adaptively) optimize all the $O(n^{1-\gamma})$ blocks of size $O(n^\gamma)$ each. Therefore, if C_i is the final cost obtained for block V_i, we use only $O(\varepsilon_0^{-2} n^\gamma)$ space in order to obtain a $C_i \leq C_i^* + \varepsilon_0 C^*$.

If we set $\varepsilon_0 = \frac{\varepsilon}{n^{1-\gamma}}$, we obtain the following:

- Correctness: $C_{int} = \sum_{i\in[m]} C_i \leq \sum_{i\in[m]} C_i^* + \varepsilon_0 C^* \leq C^* + m\varepsilon_0 C^* \leq (1+\varepsilon)C^*$, where C_i^* is the best possible cost for block V_i.
- Space: $O(\varepsilon^{-2} n^{2-2\gamma} n^\gamma) = O^*(n^{2-\gamma})$ space as desired. Note that this is the same as the space used by our sampling approach to solve Part 1.
- Time: Our sampling approach uses polynomial time, so the time complexity is dominated by the brute force part of our algorithm. This takes $O((n^\gamma)! \operatorname{poly}(n) 2^{\operatorname{poly}(1/\varepsilon)})$ time, which is the same as $2^{\Theta(n^\gamma \log n)} 2^{\operatorname{poly}(1/\varepsilon)}$, as desired.

This concludes the proof of Theorem 28. ◀

3 Other Streaming Problems on Tournament Graphs

An algorithm for two more directed graph problems is presented here.

1. `SCC`: the decision problem of determining whether a given digraph is strongly connected or not. A strongly connected digraph is one where there exists a directed path between any pair u and v of vertices.
2. `SCC-FIND`: the problem of finding the strongly connected components of a digraph. A strongly connected component is a subgraph which is maximally strongly connected.

Our space lower bounds for the `SCC` problem, which are shown later in Section 4, are prohibitive for general digraphs. To obtain fast streaming algorithms, we must turn to specific types of inputs, in this case motivating the study of tournament graphs. Tournament graphs have been the topic of study in computational social choice theory [10] and therefore, studying their structure, such as computing the strongly connected components, is an important goal. It turns out that one can solve the `SCC` and `SCC-FIND` problems on tournament graphs in the insertion-only streaming model, given multiple passes. The appropriate algorithm consists of two phases:

1. Finding a Hamiltonian Path in the graph.
2. Partitioning the path into segments which are strongly connected components.

Every tournament graph is known to have at least one Hamiltonian path, and Chakrabarti et al.[11] use an algorithm called KWIKSORT to find such a Hamiltonian path in p passes that uses $\tilde{O}(n^{1+1/p})$ space with high probability. Instead of emulating quicksort like KWIKSORT does, the subroutine introduced here for discovering a Hamiltonian path is inspired from mergesort, and by doing so it guarantees $\tilde{O}(n^{1+1/p})$ space in p passes.

▶ **Lemma 29.** *There exists a deterministic algorithm that can find a Hamiltonian path in a tournament graph using $\tilde{O}(n^{1+1/p})$ space and p passes over the data.*

Proof. Achieved with a divide and conquer algorithm presented and proven in the full version of the paper. ◀

▶ **Theorem 30.** *Given a Hamiltonian Path on a tournament graph, the `SCC` and `SCC-FIND` problems can be solved with an additional $O(n \log n)$ space and 1 pass.*

Proof. Once we have a Hamiltonian Path v_1 `->` v_2 `->` ... `->` v_n, the `SCC` problem can be solved as follows:

1. For each vertex v_i, we compute the minimum j such that $(v_i, v_j) \in E$. This represents the 'earliest' vertex that can be reached from v_i. This can be done with $O(n \log n)$ space.
2. Now for each v_i, we need to find the 'earliest' vertex directly connected to some vertex v_j where $j > i$. This is achieved by taking the suffix minima of the result of the first step, and let $f(i)$ be this suffix for v_i.
3. Finally, we check if there exists a j such that $f(j) = j$ and $j \neq 1$. If there does not that means that the graph is strongly connected since v_1 is reachable from v_n by continuously traversing the path needed to go from a current vertex i to $f(i)$. Otherwise there exists some j such that $f(j) = j$ and $j \neq 1$, which means that we cannot reach vertex v_k from v_j where for any $k < j$, and therefore, it is not strongly connected.

Note that the last step can be modified to solve the `SCC-FIND` problem as well: it is the number of j such that $f(j) = j$, and the membership sets follow by the segmentation of the path by these points where $f(j) = j$. ◀

Thus, with $\tilde{O}(n^{1+1/p})$ space and $p + 1$ passes, one can solve `SCC` and `SCC-FIND` for tournament graphs.

4 Lower Bounds for Directed Graph Problems

4.1 Single pass lower bounds

Borradaile et al. [9] demonstrated an $\Omega(m)$ lower bound for the space complexity of detecting digraph strong connectivity for 1-pass algorithms where m is the number of edges in the graph. However, this lower bound does not take into account the number of vertices in the graph and can therefore be improved for sparse graphs. We present tighter lower bounds that do take the number of vertices into account and obtain some interesting results:

▶ **Theorem 31.** *Given a directed graph with n vertices and m edges, any 1-pass streaming algorithm that solves* `SCC` *needs at least $\Omega(m \log \frac{n^2}{m})$ space.*

This lower bound is optimal as one can store the entire graph in this amount of memory.

Similar results also follow for other directed graph problems, namely `ACYCLIC` and `S-ALL-CONN`. The former involves detecting whether or not there are any directed cycles in the input graph, while the latter involves determining whether or not there exists a path from a fixed input vertex s to every single other vertex in the graph.

▶ **Theorem 32.** *Given a directed graph with n vertices and m edges, any 1-pass streaming algorithm that solves* `ACYCLIC` *needs at least $\Omega(m \log \frac{n^2}{m})$ space.*

▶ **Theorem 33.** *Given a directed graph with n vertices and m edges, any 1-pass streaming algorithm that solves* `S-ALL-CONN` *needs at least $\Omega(m \log \frac{n^2}{m})$ space.*

The proofs for the above theorems for the lower bounds of `SCC`, `ACYCLIC`, and `S-ALL-CONN` are reductions to `INDEX` and are given in the full version.

4.2 Multiple pass lower bounds

We first need to define the set chasing problem $\mathtt{SC}_{n,p}$ as described by Guruswami and Onak [20] to obtain multi-pass lower bounds for `SCC`:

▶ **Definition 34.** *The $\mathtt{SC}_{n,p}$ problem consists of p players, each with a function $f_i : [n] \to \mathcal{P}([n])$ where $\mathcal{P}(X)$ is the power set of X. Additionally, define $\overline{f_i} : \mathcal{P}([n]) \to \mathcal{P}([n])$ via $\overline{f_i} = \bigcup_{j \in [n]} f_i(j)$. The goal is to compute $\overline{f_1}(\overline{f_2}(\ldots \overline{f_n}(\{1\})\ldots))$, which is output by the p-th player at the end of the $(p-1)$-th round, the last round.*

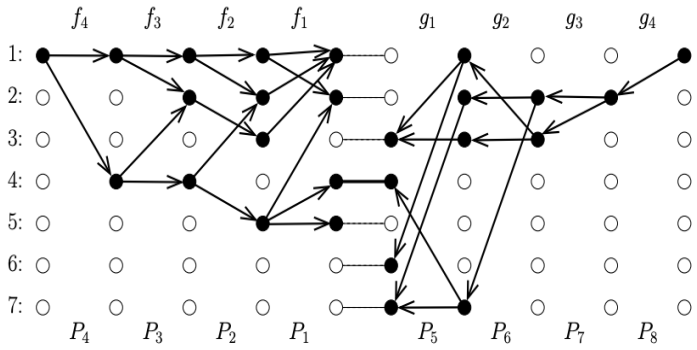

Figure 2 A true instance of $\mathtt{INTERSECT}(\mathtt{SC}_{7,4})$. Some edges have been omitted for clarity. The players play in order of their numbering. Example courtesy of [20].

The actual problem considered for the multiple pass reduction is $\texttt{INTERSECT}(\texttt{SC}_{n,p})$, which is conveniently a decision problem.

▶ **Definition 35.** *Given two instances of the $\texttt{SC}_{n,p}$, with functions $\overline{f_i}$ and $\overline{g_i}$, $\texttt{INTERSECT}(\texttt{SC}_{n,p})$ is the decision problem of checking if $\overline{f_1}(\overline{f_2}(...\overline{f_n}(\{1\})...)) \cap \overline{g_1}(\overline{g_2}(...\overline{g_n}(\{1\})...)) = \phi$.*

A diagrammatic representation of this problem is shown in Figure 2. [20] uses this problem to show a multiple pass lower bound for the directed $s-t$ connectivity problem, and a similar reduction is shown here.

▶ **Theorem 36.** *Given a directed graph with n vertices, reducing to $\texttt{INTERSECT}(\texttt{SC}_{n,p})$ demonstrates that $n^{1+\Omega(1/p)})/p^{O(1)}$ is a space lower bound for any $p-1$ pass algorithm that solves the $\texttt{SCC}$ problem (and therefore the $\texttt{SCC-FIND}$ problem).*

Proof. The reduction from $\texttt{SCC}$ to $\texttt{INTERSECT}(\texttt{SC}_{n,p})$ is constructed through the following:

1. Consider the graph similar to Figure 2, where the edges in the second half are flipped such that everything is going from left to right. This gives us a directed acyclic graph.
2. Now we add an edge from every vertex in the last layer to s, and we also add an edge from every vertex in the middle red layer to s.
3. We add an edge from t to every vertex in the graph. Also for any vertex that does not have any outgoing neighbors we add an edge from it to s.

We now consider the implications of $\texttt{INTERSECT}(\texttt{SC}_{n,p})$ on the original SCC instance.

- If $\texttt{INTERSECT}(\texttt{SC}_{n,p})$ is false then there is no path from s to t since there is no vertex in the red layer that is connected to both the left and right side that is reachable from s. Therefore the graph cannot be strongly connected.
- If $\texttt{INTERSECT}(\texttt{SC}_{n,p})$ is true then we want to show that the graph is strongly connected. First note that every node is reachable from t since we added an edge from t to every other vertex. Now if the $\texttt{INTERSECT}(\texttt{SC}_{n,p})$ is true then t is reachable from s which means that every vertex is reachable from s. Now there are two kinds of vertices:
  - Vertices that have a path to the last layer. In this case every vertex in the last layer is connected to s so s is reachable from the vertex.
  - Vertices that do not have a path to the last layer. In this case they eventually hit a vertex with no out neighbors to the last layer. But in this case we added an edge from this vertex to s so s is reachable from the vertex. ◀

References

1 Nir Ailon. An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity. *The Journal of Machine Learning Research*, 13(1):137–164, 2012.

2 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):1–27, 2008.

3 Noga Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 20(1):137–142, 2006.

4 Sanjeev Arora, Alan Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical programming*, 92(1):1–36, 2002.

5 Sepehr Assadi, Gillat Kol, Raghuvansh R Saxena, and Huacheng Yu. Multi-pass graph streaming lower bounds for cycle counting, max-cut, matching size, and other problems. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 354–364. IEEE, 2020.

6 Sepehr Assadi and Ran Raz. Near-quadratic lower bounds for two-pass graph streaming algorithms. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 342–353. IEEE, 2020.

7 Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and bayesian inference. *SIAM journal on computing*, 27(4):942–959, 1998.

8 Shuvra S Bhattacharyya, Praveen K Murthy, and Edward A Lee. Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI signal processing systems for signal, image and video technology*, 21(2):151–166, 1999.

9 Glencora Borradaile, Claire Mathieu, and Theresa Migler. Lower bounds for testing digraph connectivity with one-pass streaming algorithms. *CoRR*, abs/1404.1323, 2014. `arXiv:1404.1323`.

10 Felix Brandt, Markus Brill, and Bernhard Harrenstein. Tournament solutions, 2016.

11 Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams, 2019.

12 Pierre Charbit, Stéphan Thomassé, and Anders Yeo. The minimum feedback arc set problem is np-hard for tournaments. *Combinatorics, Probability and Computing*, 16:01–04, 2007.

13 Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh Saxena, Zhao Song, and Huacheng Yu. Almost optimal super-constant-pass streaming lower bounds for reachability. *Electron. Colloquium Comput. Complex.*, 28:27, 2021. URL: `https://eccc.weizmann.ac.il/report/2021/027`.

14 William W Cohen, Robert E Schapire, and Yoram Singer. Learning to order things. In *Advances in neural information processing systems*, pages 451–457, 1998.

15 Don Coppersmith, Lisa Fleischer, and Atri Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 776–782, 2006.

16 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

17 Paola Festa, Panos M Pardalos, and Mauricio GC Resende. Feedback set problems. In *Handbook of combinatorial optimization*, pages 209–258. Springer, 1999.

18 Alan Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

19 Xiubo Geng, Tie-Yan Liu, Tao Qin, Andrew Arnold, Hang Li, and Heung-Yeung Shum. Query dependent ranking using k-nearest neighbor. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 115–122, 2008.

20 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016.

21 Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010.

22 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 95–103, 2007.

23 Luigi Laura and Federico Santaroni. Computing strongly connected components in the streaming model. In *International Conference on Theory and Practice of Algorithms in (Computer) Systems*, pages 193–205. Springer, 2011.

24 Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

25 Antti-Veikko Rosti, Necip Fazil Ayan, Bing Xiang, Spyros Matsoukas, Richard Schwartz, and Bonnie Dorr. Combining outputs from multiple machine translation systems. In *Human*

Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference, pages 228–235, 2007.

26 Aiguo Xie and Peter A Beerel. Implicit enumeration of strongly connected components and an application to formal verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(10):1225–1230, 2000.