Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Learning-Free Ranking from Pairwise Comparisons via Feedback-Arc-Set Pruning and Add-Back

Soroush Vahidi

sv96@njit.edu

New Jersey Institute of Technology

Research Article

Keywords: Ranking, Pairwise comparisons, Feedback arc set, Directed graphs, Training-free methods

Posted Date: April 21st, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9281720/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: No competing interests reported.

Scalable and Training-Free Ranking from Pairwise Comparisons via Acyclic Graph Construction

Soroush Vahidi
New Jersey Institute of Technology, Newark, NJ, USA
sv96@njit.edu
ORCID: 0000-0003-1934-6282

Abstract

Deriving global rankings from noisy and incomplete pairwise comparisons is a fundamental problem with applications in sports analytics, preference aggregation, and evaluation. Many recent approaches rely on learning-based models, but in practice ranking systems are often required to operate under strict constraints on training cost, runtime, and reproducibility. We study a scalable, training-free alternative that constructs rankings directly from weighted directed comparison graphs.

Our approach represents pairwise comparisons as a weighted digraph and leverages the connection between ranking inconsistency and feedback-arc-set removal to build an acyclic comparison backbone. The proposed pipeline is time-bounded and consists of three stages: a local-ratio-style cycle-breaking heuristic, a stable weight-prioritized add-back procedure with up to three passes while preserving acyclicity, and an optional bounded score-refinement stage under upset-based objectives used in recent ranking benchmarks. The resulting method outputs a real-valued score vector whose induced order is consistent with the recovered acyclic structure.

Across the benchmark suite used in recent ranking-from-comparisons work, the proposed method is competitive with strong classical baselines and delivers substantial runtime advantages over training-based GNNRank configurations under fixed wall-clock budgets. These results position the method as a practical, deterministic, and scalable alternative for ranking on large pairwise-comparison graphs when training cost and deployment efficiency are important.

Keywords: Ranking; Pairwise comparisons; Feedback arc set; Directed graphs; Training-free methods

1 Introduction

Ranking from pairwise comparisons is a fundamental task in statistics, machine learning, and network analysis, with applications ranging from sports analytics to preference aggregation. Recent work in this area includes both strong classical approaches and learning-based methods, particularly graph-neural models that infer rankings directly from directed comparison graphs He et al. (2022). While learning-based methods can achieve strong performance, they also introduce training cost, hyperparameter sensitivity, and reduced reproducibility, which can be undesirable in deployment settings where runtime, determinism, and simplicity are important.

Many ranking methods produce a real-valued score vector whose induced order is evaluated by disagreement- or upset-based losses. In practice, however, directed comparison graphs often contain cycles due to noise, incompleteness, and local inconsistencies. This creates a natural tension between observed pairwise evidence and the need for a globally consistent ranking. A combinatorial way to address this issue is to remove a set of directed edges so that the remaining graph becomes acyclic; a topological ordering of the resulting directed acyclic graph (DAG) then induces a globally consistent ranking.

This viewpoint leads directly to the Minimum Weighted Feedback Arc Set (MWFAS) problem. Although the connection between ranking inconsistency and feedback arc sets is known and has been noted in prior ranking work, including spectral and synchronization-based approaches such as Fogel et al. (2014); Cucuringu (2015), it has not been developed into a practical, time-bounded, training-free ranking pipeline evaluated side-by-side with standard classical baselines and modern learning-based methods under the upset-based metrics used in recent benchmarks He et al. (2022). In this work, we use the MWFAS perspective not as an exact optimization target, but as the foundation of a deterministic ranking pipeline designed for favorable accuracy–runtime trade-offs on large comparison graphs.

To make this connection precise, we begin with formal definitions of the ranking objective and the corresponding MWFAS formulation.

Definition 1 (Ranking Problem)**.** *Let $G = (V, E)$ be a directed graph, where V is the set of items and E is the set of directed pairwise-comparison edges. Each edge $(i, j) \in E$ has weight $w_{ij} > 0$, representing the strength of the preference for i over j. A* ranking *is a total order on V, represented equivalently by a bijection $R : V \to \{1, \ldots, |V|\}$, where smaller values indicate higher rank. An edge (i, j) is* inconsistent *with R if $R(i) > R(j)$. The goal is to find a ranking that minimizes the total weight of inconsistent edges:*

$$\min_{R} \sum_{(i,j)\in E:\ R(i)>R(j)} w_{ij}. \tag{1}$$

Definition 2 (Minimum Weighted Feedback Arc Set (MWFAS))**.** *Let $G = (V, E)$ be a directed graph with edge weights $w_{ij} > 0$ for each $(i, j) \in E$. A set of edges $F \subseteq E$ is a* feedback arc set *if removing F makes the graph acyclic, i.e., $(V, E \setminus F)$ is a directed acyclic graph (DAG). The MWFAS problem is*

$$\min_{F\subseteq E} \sum_{(i,j)\in F} w_{ij} \quad s.t. \quad (V, E \setminus F) \text{ is acyclic.} \tag{2}$$

The ranking objective (1) and MWFAS (2) are two equivalent views of the same consistency-restoration problem on a directed, weighted comparison graph. Given a ranking R, the set of *backward* edges

$$\{(i, j) \in E :\ R(i) > R(j)\}$$

forms a feedback arc set, and its total weight is exactly the disagreement objective in (1). Conversely, any feedback arc set F yields a DAG $(V, E \setminus F)$ that admits a topological ordering consistent with all remaining edges. Therefore, computing a minimum-weight set of edges to delete for acyclicity is equivalent to finding an ordering that minimizes the total weight of violated comparisons.

1.1 Ranking Problem and Minimum Weighted Feedback Arc Set (MWFAS)

The equivalence above suggests a simple ranking strategy: remove a feedback arc set to obtain an acyclic backbone and then read off an ordering from the resulting DAG. This viewpoint is particularly compatible with large sparse comparison graphs, where topological ordering is linear-time in the number of edges and naturally produces a deterministic ranking once tie-breaking is fixed. However, MWFAS is computationally difficult: the (unweighted) Minimum Feedback Arc Set problem is NP-hard Karp (1972), and the weighted variant is NP-hard as well, motivating the use of approximation and heuristic algorithms.

In this work we exploit the MWFAS–ranking connection as a *ranking pipeline* rather than as an exact solver. Concretely, our method (described in §2) uses a local-ratio-style cycle-breaking heuristic inspired by Demetrescu and Finocchi (2003) to construct an acyclic backbone, followed by a stable weight-prioritized add-back stage that reinserts high-weight edges while preserving acyclicity (INS1/2/3 variants), and finally extracts a score vector whose induced order is evaluated by the same upset-based losses used in recent

ranking benchmarks He et al. (2022). Our experimental evaluation (§3) compares this learning-free MWFAS-based pipeline against standard classical ranking baselines (including BTL, SpringRank, SyncRank, SerialRank, centrality and SVD-based methods) and against learning-based GNNRank models, under a common protocol and time budgets.

1.2 Related Works

1.2.1 Previous Work on Ranking from Pairwise Comparisons

Ranking from noisy and incomplete pairwise comparisons has been studied through statistical, spectral, random-walk, and optimization-based formulations Hochberg and Rabinovitch (2000); Chen et al. (2021); Shah and Wainwright (2016). Since our experiments evaluate score-based outputs under upset-style losses, we focus here on the principal baseline families most relevant to our benchmark and to the positioning of our method as a training-free ranking pipeline.

A first major family is probabilistic and statistical ranking. BTL (Bradley–Terry) models each item with a positive latent ability parameter and defines pairwise win probabilities as $\Pr(i \succ j) = \pi_i/(\pi_i + \pi_j)$ Bradley and Terry (1952). Estimating these abilities yields real-valued scores that induce a global ranking and remains one of the standard baselines for paired-comparison data. DavidScore provides a simpler summary-based alternative that aggregates possibly unbalanced paired-comparison outcomes into a single score per item David (1987).

A second family uses random-walk and spectral constructions. RankCentrality builds a Markov chain from pairwise outcomes and uses its stationary distribution as a ranking score vector Negahban et al. (2017). SerialRank constructs a similarity/seriation matrix and then applies spectral seriation to recover an ordering Fogel et al. (2014), while SyncRank formulates ranking from incomplete and noisy comparisons as an angular synchronization problem and recovers scores through spectral or SDP-based relaxations Cucuringu (2015). Related low-rank approaches include SVD_RS and SVD_NRS, which derive one-dimensional ranking scores from pairwise measurements using singular-vector structure under different normalization choices d'Aspremont et al. (2021).

A third group consists of graph-based scoring methods that operate directly on the directed comparison network. EigenvectorCentrality assigns scores recursively through weighted endorsements from neighboring nodes Bonacich (1987), while PageRank interprets importance through the stationary distribution of a random walk with teleportation Page et al. (1999). SpringRank provides a physics-inspired alternative by embedding nodes on a real-valued axis through minimization of a quadratic spring-energy objective over directed edges, leading to a scalable linear-system-based estimator De Bacco et al. (2018).

More recently, learning-based methods have used graph neural networks to infer ranking scores from directed comparison graphs. GNNRank learns global rankings from pairwise comparisons using directed-GNN backbones and differentiable objectives tailored to ranking losses He et al. (2022). In our experiments we use the two backbones provided in that framework: DIGRAC, which is designed to capture directionality and flow imbalance in directed graphs He et al. (2021), and an inception-style directed convolutional backbone (ib) that combines multi-hop proximity channels for directed graphs Tong et al. (2020). These learning-based methods provide a useful point of comparison because they often achieve strong ranking quality but incur training cost and additional hyperparameter sensitivity.

Our work differs from the above families in emphasis rather than problem setting. Instead of learning ranking scores through statistical estimation or graph-neural training, we exploit combinatorial acyclicity structure to construct a deterministic, training-free ranking pipeline that is evaluated under the same upset-based criteria as these baselines.

1.2.2 Previous Work About Minimum Weighted Feedback Arc Set

The Minimum Weighted Feedback Arc Set (MWFAS) problem has been studied extensively from approximation, heuristic, exact, and parameterized perspectives Karp (1972); Festa et al. (2020). In general terms, MWFAS asks for a minimum-weight set of directed edges whose removal makes a graph acyclic, and it is a natural combinatorial formulation for resolving inconsistencies in directed comparison data.

Early approximation work by Even et al. (1995) was strengthened by Even et al. (2000), who developed divide-and-conquer methods based on spreading metrics and linear-programming relaxations. While these results are theoretically important, their reliance on comparatively heavy optimization machinery limits their practical appeal for large ranking graphs. For faster practical computation, Demetrescu and Finocchi (2003) proposed an $O(|V||E|)$ heuristic based on the local-ratio framework of Bar-Yehuda et al. (2004), together with an approximation guarantee parameterized by the length of the longest directed cycle. This line is particularly relevant to our work because it provides the algorithmic starting point for our cycle-breaking stage.

Exact and near-exact approaches have also been studied, primarily for smaller or sparser instances. Baharev et al. (2021) presented an exact method for the unweighted minimum feedback arc set problem on sparse directed graphs using lazy cycle enumeration, while Hecht et al. (2021) studied tight localizations of feedback sets and algorithms whose worst-case running times can become prohibitive on dense graphs. These approaches are valuable as methodological reference points, but their scalability is limited in the large benchmark settings targeted in our paper.

From a parameterized viewpoint, Raman and Saurabh (2006) investigated feedback set problems in tournaments, including weighted and unweighted feedback arc set variants, and obtained fixed-parameter algorithms with running times exponential in the solution size. Related approximation work by Hassin and Rubinstein (1994) considered the maximum acyclic subgraph problem, which is closely connected to unweighted feedback arc deletion. More recently, Shea-Blymyer and Young (2021) explored satisfiability-modulo-theories approaches to MFAS using incremental solving, again highlighting the trade-off between solution quality and scalability.

Recent heuristic work has also emphasized structural refinements of feedback-arc-set solutions. Cavallaro et al. (2024) introduced the notions of minimal and stable feedback arc sets and proposed the MS-FAS heuristic based on strongly connected component decomposition and iterative refinement. Building on this direction, Cavallaro et al. (2025) examined how different initial orderings, including centrality-based ones, affect MS-FAS behavior and reported robustness to initialization across benchmark instances.

Our contribution is not a new approximation guarantee or an exact algorithm for MWFAS. Instead, we use the MWFAS viewpoint as the foundation of a practical ranking pipeline for pairwise comparisons: we construct an acyclic backbone, selectively reinsert high-weight edges while preserving acyclicity, and then extract real-valued ranking scores that can be evaluated directly against standard ranking baselines under modern upset-based metrics.

2 A Framework for Ranking from Pairwise Comparisons via MWFAS

We adopt the local-ratio feedback-arc-set heuristic of Demetrescu and Finocchi Demetrescu and Finocchi (2003) as the combinatorial core and develop it into a deterministic, time-bounded ranking pipeline for weighted pairwise-comparison graphs. Given a directed comparison graph, the proposed pipeline (i) constructs an acyclic backbone through cycle-based residual reduction and arc removal (Phase 1), (ii) selectively reinserts high-weight arcs while preserving acyclicity (Phase 2), (iii) extracts a ranking from the resulting DAG and assigns real-valued scores consistent with that order, and (iv) optionally refines score magnitudes under bounded computation while preserving the induced ranking (Phase C). The underlying local-ratio

heuristic of Demetrescu and Finocchi (2003) comes with an approximation guarantee bounded by the length of a longest simple directed cycle in the input graph.

We do not claim any new approximation guarantee beyond that underlying result. Rather, our contribution is a practical, scalable, and training-free ranking-oriented instantiation of the MWFAS viewpoint, with deterministic tie-breaking, explicit time budgeting, robust numerical handling, multi-pass add-back variants, and an optional order-preserving refinement stage. In this sense, the method is intended not as an exact solver for MWFAS, but as a deployable alternative to learning-based ranking methods when runtime, reproducibility, and training cost are important considerations.

2.1 Problem Setting and Input Representation

Comparison graph. Let $V = \{1, \ldots, n\}$ denote the items to be ranked. We represent pairwise comparisons by a weighted directed graph $G = (V, E, w)$, encoded as a nonnegative weighted adjacency matrix $A \in \mathbb{R}^{n \times n}$, where A_{ij} denotes the aggregated weight of the directed comparison $i \to j$. The graph is fully general and is not restricted to tournaments; in particular, both $A_{ij} > 0$ and $A_{ji} > 0$ may occur for the same pair of items. In the implementation, only strictly positive weights are retained, so entries with $A_{ij} \leq 0$ are ignored in the internal representation.

Input representation. The canonical solver input is a sparse adjacency matrix. In the benchmark pipeline, datasets are loaded as sparse adjacency files and stored as `scipy.sparse.csr_matrix` objects, which are then passed directly to the proposed method. For completeness, the implementation also accepts dense inputs and converts them to CSR format before invoking the core routine.

Internal edge-list form. For computation, the sparse matrix A is converted to an edge-list representation with arrays `src`, `dst`, and `w` of length m, where each strictly positive entry of A contributes one directed edge. Thus, antiparallel comparisons are preserved as distinct arcs whenever both directions are present. Unless explicitly stated otherwise, edge weights are used in their original form and are not normalized or rescaled before the acyclicity-enforcement stages.

Outputs and score convention. The core routine returns a real-valued score vector $s \in \mathbb{R}^n$, where larger values indicate higher rank. When a discrete ordering is needed for evaluation, the benchmark derives it from the score vector in a deterministic way. The implementation can also return auxiliary metadata, including runtime and per-phase diagnostics such as the number of edges removed in Phase 1, the number reinserted in each add-back pass, and the corresponding phase timings.

Determinism and tie-breaking. A central design goal of the pipeline is deterministic behavior under fixed input and time budget. In Phases 1 and 2, cycle search traverses vertices in increasing index order and explores outgoing edges in stored edge order, while the add-back stage processes candidate edges in a stable descending-weight order. Topological ordering for add-back and score extraction is computed with Kahn's algorithm under a deterministic queue policy: initially eligible zero-indegree vertices are processed in increasing index order, and newly eligible vertices are appended as they arise. The optional score-refinement stage preserves the induced order but can introduce minor non-determinism when exact score ties are present and tie-breaking depends on non-stable sorting. Aside from such score-tie effects, the pipeline is deterministic for fixed inputs and time limits.

2.2 Phase 1: Local-Ratio Cycle Breaking (Acyclic Backbone)

This subsection describes Phase 1 of our pipeline, which produces an *acyclic backbone* by iteratively identifying directed cycles and removing a subset of arcs. The procedure is a local-ratio-style heuristic: it maintains residual arc weights and repeatedly reduces residual mass along a detected cycle, deleting arcs whose residual reaches (near) zero. We emphasize that Phase 1 is a practical heuristic component; we do not claim a new approximation guarantee beyond what is known for local-ratio-type feedback-arc-set heuristics, and our contribution is the ranking-oriented, time-bounded, deterministic instantiation used in the experimental protocol.

State and initialization. Given the input adjacency A, we first form an edge list $(\texttt{src}, \texttt{dst}, \texttt{w})$ containing one directed edge for each strictly positive entry of A (see §2.1). Phase 1 maintains: (i) a residual weight vector $r \in \mathbb{R}^m$ initialized by $r \leftarrow w$, (ii) a Boolean mask $\texttt{alive} \in \{0,1\}^m$ indicating which edges are currently kept (initialized to all `True`), and (iii) a static adjacency structure `adj_e` in which `adj_e[u]` stores the indices of all edges with source vertex u. Edges are removed by setting `alive[e]` to `False`; the adjacency lists are not rebuilt, and removed edges are skipped via the `alive` mask.

Cycle detection. At each iteration, Phase 1 calls a deterministic DFS-based cycle finder to obtain one directed cycle in the currently alive subgraph. The cycle finder performs an iterative depth-first search over vertices in increasing index order and explores outgoing edges in the stored adjacency order. It uses standard three-color marking (`WHITE/GRAY/BLACK`) together with parent pointers `parent_v` and `parent_e`. A back edge to a `GRAY` vertex identifies a directed cycle, which is reconstructed as a list of edge indices by following parent pointers and returned to the caller.

Local-ratio-style residual reduction and deletion. Given a detected cycle C (represented as a list of edge indices), we compute

$$\Delta \;=\; \min_{e \in C} r_e.$$

If $\Delta > 0$, we subtract Δ from the residual of every edge on the cycle, i.e., $r_e \leftarrow r_e - \Delta$ for all $e \in C$. We then delete all edges whose residual is numerically zero, using a tolerance `zero_tol` (default 10^{-15}): edges with $r_e \le$ `zero_tol` are removed by setting `alive[e]` to `False`. To guarantee progress in floating-point arithmetic, if residual reduction fails to delete any edge on the current cycle (e.g., due to roundoff), we forcibly delete the minimum-residual edge on the cycle by setting its residual to zero and marking it dead. This ensures that at least one edge is removed from the alive subgraph per iteration whenever a cycle is found.

Stopping conditions and time budgeting. Phase 1 repeats the above loop until either (i) no directed cycle is found in the alive subgraph (in which case the kept subgraph is acyclic), or (ii) a global wall-clock budget `time_limit_sec` is exceeded. In the latter case Phase 1 terminates early and passes the current kept set to Phase 2; therefore the output of Phase 1 is guaranteed to be acyclic only when it stops due to cycle absence (not due to timeout). In all experiments, Phase 1 is executed under the same per-run timeout policy as the rest of the non-GNN pipeline (§3.1).

Recorded diagnostics. For reproducibility and profiling, the solver records Phase 1 diagnostics in the returned metadata when enabled, including the number of Phase 1 iterations, the number of removed edges after Phase 1, the number of kept edges after Phase 1, and the Phase 1 wall-clock time. These quantities are used only for analysis and do not affect the ranking output.

2.3 Phase 2: Stable Weight-Prioritized Add-Back (Variants INS1/INS2/INS3)

Phase 2 post-processes the acyclic backbone produced by Phase 1 by attempting to reinsert previously removed arcs while preserving acyclicity. The goal of this stage is to recover high-confidence comparisons, as reflected by larger edge weights, without reintroducing directed cycles. To this end, we use a deterministic, time-bounded add-back procedure that scans candidate arcs in stable descending order of weight and accepts an arc only when it is forward with respect to a topological ordering of the current kept subgraph.

Inputs and outputs. Phase 2 is applied after Phase 1 and takes as input the vertex count n, the full edge list $(\texttt{src}, \texttt{dst}, \texttt{w})$, and an initial Boolean mask `kept_initial` indicating which arcs remain after cycle breaking. It returns a final mask `kept` describing the post-processed DAG. When enabled, the implementation also records auxiliary information such as the kept mask after each pass, the number of reinserted arcs per pass, and a termination reason.

Stable descending-weight scan. Let w_e denote the original weight of arc e. We first compute a stable ordering of all arcs in descending order of w_e, using a stable sorting procedure so that ties are handled deterministically. Within each pass, arcs are scanned in this fixed order, and only arcs that are currently absent from the kept set are considered for reinsertion. Stable ordering is important because it ensures reproducible tie-breaking among equal-weight arcs and gives priority to stronger comparisons throughout the add-back process.

Acyclicity test within each pass. At the beginning of each add-back pass, we compute a topological ordering of the current kept subgraph using Kahn's algorithm. Let $\mathrm{pos}(v)$ denote the position of vertex v in this order. An absent arc (u, v) is accepted if and only if $\mathrm{pos}(u) < \mathrm{pos}(v)$, that is, if the arc is forward with respect to the current topological order. This provides an $O(1)$ acceptance test per arc once the topological positions are known and avoids more expensive dynamic reachability checks during the scan. Importantly, the topological order is computed once at the start of a pass and is not updated after each individual insertion.

Why acyclicity is preserved. Because only forward arcs are inserted with respect to a valid topological ordering of the current kept subgraph, that same ordering remains valid after all accepted insertions within the pass. Therefore, the kept subgraph remains acyclic throughout the pass and at its conclusion. If the current kept subgraph is not acyclic, for example because an earlier stage terminated under a time limit before full resolution, Kahn's algorithm will fail to produce a complete topological order; in that case, the implementation stops Phase 2 and records the corresponding termination condition.

Multiple passes and INS variants. We consider three multi-pass variants, namely OURS-MFAS-INS1, OURS-MFAS-INS2, and OURS-MFAS-INS3, which execute $P = 1, 2, 3$ add-back passes, respectively. After each pass, a new topological order is computed on the updated kept subgraph, and the same stable descending-weight scan is repeated. Multiple passes are useful because an arc that is backward under one topological order may become forward after other insertions have altered the structure of the kept DAG. In this way, later passes can recover additional high-weight comparisons that were not admissible earlier. Phase 2 terminates early if the global time budget is exhausted, if a pass inserts no new arcs, or if a valid topological order cannot be produced.

Determinism and recorded metadata. Phase 2 is deterministic under fixed input and time limits. Determinism follows from the use of stable weight ordering together with a deterministic Kahn queue policy, in which initially eligible zero-indegree vertices are processed in increasing vertex order and newly eligible

vertices are appended as they arise. For reproducibility and profiling, the solver records metadata including the number of executed passes, the number of reinserted arcs per pass, and the reason for termination.

2.4 Ranking Extraction and Order-Preserving Score Refinement

After Phases 1–2, the solver maintains a set of kept arcs (encoded by a Boolean mask over the edge list). This subsection describes (i) how we extract a ranking-consistent score vector from the kept subgraph and (ii) two optional refinement steps that improve upset-style objectives under a fixed time budget. These steps are used uniformly in the experimental protocol (§3.1) and return a score vector whose induced ordering is evaluated by the same upset metrics as all baselines.

Score extraction from the kept DAG. Given a kept mask `kept`, we compute a topological order of the kept subgraph using Kahn's algorithm (the same routine as in Phase 2). The queue policy is deterministic: initial zero-indegree vertices are enqueued in increasing vertex id order, and newly eligible vertices are appended as they arise. Let $\mathrm{pos}(v)$ denote the position of vertex v in the returned topological order (smaller position means earlier in the order). We assign scores by

$$s_v \;=\; \max\{1,\; n - \mathrm{pos}(v)\},$$

so earlier vertices in the topological order receive larger scores (higher rank). If, due to early termination under a time limit, the kept subgraph is not acyclic and a full topological order cannot be produced, the implementation falls back to the identity order $[1, 2, \ldots, n]$ for score assignment; this case is recorded via metadata and is not typical under the benchmark time budgets.

Optional local refinement of the induced order (naive-upset swaps). Optionally, we apply a bounded local-improvement procedure that directly optimizes the *naive upset* objective while operating on the induced ordering. Starting from the current order (obtained by sorting scores), the procedure performs repeated sweeps of adjacent swaps. A swap is accepted if it strictly decreases the weighted naive-upset loss (sum of weights of edges that disagree with the current order) by at least a small numerical tolerance. The refinement is time-bounded by a local budget and also respects the global per-run time limit; it terminates early if a full sweep makes no improvement. After refinement, we reconstruct a score vector consistent with the refined order by assigning ranks $n, n-1, \ldots, 1$ along the order and clamping scores to be at least 1. This step can change the induced ranking (permutation), but it preserves the convention that larger scores indicate higher rank.

Optional order-preserving ratio-upset refinement (coordinate-wise ternary search). We also implement an order-preserving refinement of *score magnitudes* that targets the ratio-upset objective used in prior work. Given a fixed ordering induced by the current scores, we optimize score values while enforcing strict monotonicity so that the ordering does not change. Concretely, we process vertices in nondecreasing score order and update one coordinate at a time using ternary search to minimize the ratio-upset contribution, subject to box constraints

$$s_{\mathrm{prev}} + \delta \;\le\; s_v \;\le\; s_{\mathrm{next}} - \delta$$

with a small margin $\delta > 0$. A monotone repair step enforces these constraints after updates. This refinement is executed for a bounded number of passes and ternary-search iterations per coordinate, and it uses a dedicated budget `refine_time_sec` that is additionally capped by the remaining global time in `time_limit_sec`. When the remaining budget is below a minimal threshold, the refinement is skipped.

Per-pass scores and metadata. When enabled, the solver can record intermediate score vectors after each add-back pass (INS1/INS2/INS3), by computing scores from the kept mask after each Phase 2 pass and optionally applying the same refinement policy to each. The solver also records timing diagnostics, including the time spent in the refinement stage and whether ratio refinement was enabled; these quantities support profiling and reproducible benchmarking.

Determinism considerations. Phases 1–2 and score extraction are deterministic under fixed inputs and time limits (see §2.1–§2.3). The naive-swap refinement uses stable sorting for deterministic tie-handling. The ratio-upset refinement processes vertices in sorted score order using a non-stable sort; therefore, when the current score vector contains exact ties, the internal update order can vary and may lead to minor non-determinism in the refined magnitudes. This non-determinism affects only score magnitudes and is avoided when scores are strictly ordered by construction.

2.5 Complexity and Practical Considerations

We summarize the computational costs of the three stages of the proposed pipeline and relate these costs to the empirical scaling behavior observed in the benchmark. Let n denote the number of vertices, m the number of directed edges with positive weight, $P \in \{1, 2, 3\}$ the number of add-back passes in the INS variants, and I the number of Phase 1 iterations completed before termination.

Phase 1: cycle breaking. Phase 1 maintains a residual weight for each edge together with an *alive* mask indicating which edges remain active. Directed cycles are identified by depth-first search on the currently active subgraph. Since inactive edges are skipped, one cycle-detection call runs in $O(n + m_{\text{alive}})$ time, where $m_{\text{alive}} \leq m$. Each iteration reduces residual mass along a detected cycle and deletes all edges whose residual falls below the numerical threshold. Because at least one edge is removed whenever a cycle is processed, the number of iterations is at most m in the worst case. In practice, Phase 1 is also explicitly time-bounded and stops either when no directed cycle remains or when the available time budget is exhausted.

Phase 2: stable weight-prioritized add-back. Phase 2 begins with one stable sorting of all edges by descending original weight, which costs $O(m \log m)$. Each add-back pass then computes a topological ordering of the current kept subgraph in $O(n + m)$ time and scans the full edge list once. An absent edge (u, v) is accepted if it is forward with respect to the topological order of that pass, so the acceptance test is $O(1)$ once vertex positions in the topological order are known. Consequently, Phase 2 runs in $O(m \log m + P(n + m))$ time in the worst case, with early termination when no additional edges can be reinserted or when the time budget is reached. By construction, this stage preserves acyclicity throughout each pass.

Phase 3: optional score refinement. We implement two optional refinement procedures, both executed under explicit time budgets.

The first is a naive adjacent-swap refinement that improves the weighted upset objective by local swaps in the current order. One sweep examines $O(n)$ adjacent pairs, and evaluating each candidate swap requires scanning the edge set, giving worst-case cost $O(nm)$ per sweep. In practice, however, this stage is run with a small number of sweeps and a strict local time budget.

The second is an order-preserving ratio refinement that adjusts score magnitudes without changing the induced ordering. This refinement constructs arrays over unordered vertex pairs (i, j) with nonzero total weight $A_{ij} + A_{ji}$; in the worst case, this structure has size $\Theta(n^2)$. It then performs coordinate-wise updates using ternary search while enforcing strict monotonicity constraints. This stage is likewise explicitly budgeted and is capped by the remaining global time.

Family	#	n range		m range	
		min	max	min	max
Basketball (temporal + finer)	60	282	351	2,904	7,650
Football (temporal + finer)	12	20	20	107	380
Faculty (Business)	1	113	113	1,787	1,787
Faculty (Computer Science)	1	206	206	1,407	1,407
Faculty (History)	1	145	145	1,204	1,204
Animal social network	1	21	21	193	193
Head-to-head competition	2	602	602	5,010	5,010
Finance	1	1,315	1,315	1,729,225	1,729,225
Other (ERO-style)	1	within overall range		within overall range	

Table 1: Benchmark suite summary by dataset family. Each instance is a weighted directed graph of pairwise comparisons.

Memory footprint. The dominant memory cost comes from the sparse edge-list representation: endpoint arrays, weights, residuals, and masks all require $O(m)$ storage, and the adjacency lists also store $O(m)$ edge references. Phase 2 adds one sorted index array of length m. If ratio refinement is enabled, the auxiliary pairwise structures can require up to $O(n^2)$ memory in the worst case. The pipeline is otherwise designed for sparse inputs and does not require densifying the adjacency matrix.

Practical scaling and the Finance bottleneck. All stages operate under a single wall-clock budget, and the experimental driver enforces a non-GNN timeout of 1800 s (see §3.1). The main scalability limitation in the current benchmark arises on very large or dense graphs, most notably Finance with $m \approx 1.7 \times 10^6$. In such cases, the combination of repeated cycle searches in Phase 1, full-edge scans in each add-back pass, and potentially quadratic pairwise structures in the ratio-refinement stage can exhaust the available time budget. This behavior is consistent with the intended operating regime of the method: the pipeline is effective on large sparse and moderately dense comparison graphs, while extremely dense instances place simultaneous pressure on the iterative cycle-breaking stage and on the global edge- and pair-based refinement procedures.

3 Experimental Results

3.1 Experimental Setup

Datasets. We evaluate the proposed method on a benchmark suite of 80 weighted directed comparison graphs drawn from several application domains, following the benchmark setting used in recent ranking-from-comparisons work (He et al., 2022). The suite is dominated by temporal sports graphs and includes 60 basketball instances (seasons 1985–2014 together with a higher-resolution "finer" version of each season) and 12 football instances (seasons 2009–2014 together with finer versions). In addition, the suite contains three faculty hiring networks (Business, Computer Science, and History), one animal social network, two head-to-head competition graphs, one large finance graph, and one additional non-sports instance of ERO style. Across the full suite, the number of vertices ranges from $n = 20$ to $n = 1315$, and the number of directed edges ranges from $m = 107$ to approximately 1.73×10^6. Table 1 summarizes the dataset families and their scale ranges.

Methods compared. We compare four variants of the proposed pipeline: OURS-MFAS, OURS-MFAS-INS1, OURS-MFAS-INS2, and OURS-MFAS-INS3. We also compare against ten classical ranking baselines: SpringRank, SyncRank, SerialRank, BTL, DavidScore, EigenvectorCentrality, PageRank, RankCentrality, SVD-RS, and SVD-NRS. In addition, we include learning-based methods from the GNNRank framework using two directed graph neural network backbones, DIGRAC and ib. For each backbone, we report two fixed benchmark configurations: a distance-based configuration and a proximal-baseline configuration with distance-based pretraining. The main result tables report per-method results explicitly. When we additionally report compact "best-of" summaries, we treat them only as oracle-style envelopes over a fixed candidate set and not as standalone methods.

Evaluation metrics. We report three upset-based losses computed from the weighted adjacency matrix A and the predicted score vector s, where larger scores indicate higher rank. Let $M = A - A^\top$, and define score differences $T_{ij} = s_i - s_j$ over pairs with $M_{ij} \neq 0$. *Upset-naive* is the fraction of pairs for which the predicted sign disagrees with $\mathrm{sign}(M_{ij})$. *Upset-simple* averages the squared disagreement between $\mathrm{sign}(T_{ij})$ and $\mathrm{sign}(M_{ij})$. *Upset-ratio* normalizes both score differences and pairwise edge imbalance by their corresponding sums and averages the resulting squared deviation:

$$T_{ij}^{(r)} = \frac{s_i - s_j}{s_i + s_j + \varepsilon}, \qquad M_{ij}^{(r)} = \frac{A_{ij} - A_{ji}}{A_{ij} + A_{ji} + \varepsilon},$$

where $\varepsilon > 0$ is a small constant.

Protocol and compute. All methods are evaluated with `TRIALS=10` repeated runs for each dataset–method pair. For all real datasets in the suite, the dataset seed is fixed internally to `seeds=[10]`. As a result, deterministic non-GNN methods produce identical outputs across trials apart from timing variability and timeout effects, whereas GNN methods can vary across runs because of stochastic training. Experiments are executed on a SLURM cluster using a typical allocation of one task, 48 CPU cores, and 100 GB memory. Non-GNN methods, including classical baselines and the proposed variants, are run under a per-run wall-clock budget of 1800 s, while GNN methods are run under a 7200 s budget. Timed-out runs are recorded as missing values and are excluded from aggregate statistics via NaN-robust averaging.

In addition to these native budgets, we also report a compute-matched view in which all families are restricted to runs that finish within 1800 s. Coverage is reported explicitly in that view. Under the recorded protocol, the compute-matched dataset set is a strict subset of the full suite because the ERO-style instance does not complete within 1800 s.

Result aggregation and per-dataset comparisons. To ensure numerical consistency in the reported tables, we aggregate accuracy and runtime statistics primarily from saved per-trial arrays whenever they are available, using full-precision means and standard deviations across trials. If a dataset–method entry is unavailable in the array artifacts, we use the corresponding per-dataset summary extracted from the experiment logs and record this fallback in the missingness audit. Unless stated otherwise, all win/tie/loss comparisons are computed from full-precision per-dataset means using a tie tolerance of $|\Delta| \leq 10^{-6}$, where Δ denotes the difference between the losses of two methods on the same dataset. Missing entries, including timeouts and missing runtime values, are reported explicitly through coverage columns and audit tables rather than being omitted silently.

Interpreting mean±std summaries. The manuscript reports two different kinds of variability. When mean±std summarizes results across datasets, the standard deviation reflects heterogeneity across instances in the benchmark suite. When mean±std is reported for repeated runs of a fixed dataset–method pair,

Item	Value	Notes
Trials per dataset–method	10	Deterministic non-GNN methods vary only through timing and timeout effects
Seed (real datasets)	10	Fixed internally to `seeds=[10]`
Non-GNN timeout	1800 s	Applied to classical baselines and OURS variants
GNN timeout	7200 s	Applied per GNN run
Compute-matched budget	1800 s	Runs are filtered to runtime $\leq$ 1800 s and coverage is reported explicitly
GNN configurations per backbone	2	`train_with` in {`dist`, `proximal_baseline`}
GNN backbones	2	DIGRAC, IB
OURS metric coverage	77/80	Finance times out for all OURS variants under the 1800 s non-GNN budget

Table 2: Protocol and computational settings used in the benchmark suite.

the standard deviation reflects run-to-run variability. For deterministic non-GNN methods on real datasets, this variability is typically zero except for timing and timeout effects; for GNN methods, it can be nonzero because of stochastic optimization.

Unified execution and evaluation. All non-GNN methods, including the classical baselines and the proposed variants, are evaluated through the same non-neural benchmark pathway. Each method produces a score vector $s \in \mathbb{R}^n$, where larger values indicate higher rank. When a method naturally returns only an ordering, we convert that ordering to deterministic rank-based scores so that all methods can be evaluated under the same three upset losses defined in §3.1. Learning-based methods are trained and evaluated separately through the GNNRank framework, but their outputs are scored using the same loss implementations for consistency. Because the dataset seed is fixed for all real datasets (Table 2), non-GNN methods are deterministic under fixed time limits, whereas GNN runs may vary because of stochastic optimization.

Classical ranking baselines. Our classical benchmark suite includes ten standard training-free ranking methods: SpringRank, SyncRank, SerialRank, BTL, DavidScore, EigenvectorCentrality, PageRank, RankCentrality, SVD-RS, and SVD-NRS. These methods map the comparison graph directly to a score vector and span statistical, spectral, centrality-based, and physics-inspired ranking families. We use the default settings provided by the corresponding public implementations and do not tune hyperparameters on a per-dataset basis.

Proposed pipeline and variants. The proposed method is a combinatorial, training-free pipeline that first constructs an acyclic comparison backbone and then optionally densifies that backbone while preserving acyclicity. Concretely, it (i) removes arcs to eliminate directed cycles using local-ratio-style residual reduction, (ii) reinserts previously removed arcs in stable descending weight order subject to acyclicity, and (iii) extracts a score vector from the resulting DAG and optionally refines score magnitudes under the same upset-based objectives used for evaluation (see §2). We report four non-GNN variants: OURS-MFAS, OURS-MFAS-INS1, OURS-MFAS-INS2, and OURS-MFAS-INS3. These variants differ only in the number of add-back passes, with INS1, INS2, and INS3 corresponding to $P = 1, 2, 3$, respectively. Unless stated otherwise, we report OURS-MFAS-INS3 as **OURS** in the headline summaries, while INS1 and INS2 serve as lighter variants of the same pipeline.

Complexity and time-bounded operation. Let n and m denote the number of vertices and directed edges in the input graph, and let $P \in \{1, 2, 3\}$ denote the number of add-back passes. Phase 2 sorts edges once by weight, costing $O(m \log m)$, and then performs up to P passes. Each pass computes a topological ordering of the currently kept graph and scans edges in weight order with $O(1)$ acceptance checks, yielding worst-case complexity $O(m \log m + P(n + m))$. Phase 1 repeatedly detects and breaks directed cycles; each cycle-finding call runs in $O(n + m_{\text{alive}})$ over the currently active edges, and at least one edge is deleted whenever a cycle is found, so the number of iterations is at most m in the worst case. In practice, all phases operate under a fixed wall-clock budget (Table 2), so our emphasis is on empirical runtime under shared budgets rather than on introducing a new approximation guarantee.

GNNRank methods and configurations. To represent learning-based alternatives, we include two directed-GNN backbones from the GNNRank framework: DIGRAC and IB. For each backbone, we report two fixed benchmark configurations: a distance-based configuration and a proximal-baseline configuration with distance-based pretraining. This yields four GNN configurations per dataset in total. In the main tables, these configurations are reported explicitly. When we provide compact "best-of" comparisons against GNNRank, we treat them only as oracle envelopes over this fixed candidate set and not as standalone methods.

Timeout handling and missingness. The benchmark enforces per-run wall-clock limits of 1800 s for non-GNN methods and 7200 s for GNN methods (Table 2). Timed-out runs are recorded as missing values and are excluded from aggregate statistics via NaN-robust averaging. Coverage, defined as the number of datasets contributing to a method's summary statistics, is reported explicitly throughout the results section. Under the 1800 s non-GNN budget, all proposed variants time out on the Finance instance, so **OURS** contributes metrics on 77 of the 80 datasets in the full suite. Any remaining missing values, including missing runtime entries or explicit dataset-level exclusions, are reported in the missingness audit rather than omitted silently.

Aggregation and numerical precision. To reduce artifacts from log formatting or rounding, our primary table construction uses full-precision per-trial arrays saved from the benchmark runs whenever these are available. If a dataset–method entry is unavailable in the saved arrays, we fall back to the corresponding per-dataset summary extracted from experiment logs and record that fallback in the audit outputs. All tables and figures in §3.2–§3.3 are generated from the same per-method CSV artifacts, and an automated validation step is used to check internal consistency of coverage and missingness summaries.

3.2 Main Results on Benchmark Datasets

Per-method leaderboard on the full suite. Table 4 summarizes the main per-method results for the non-GNN methods on the full benchmark suite of 80 datasets. Each row reports robust aggregate performance across datasets, including the median and mean of the three upset losses together with explicit coverage. Lower values indicate better performance. We report the proposed variants and the classical baselines individually, since these methods have meaningful coverage across the suite and admit direct comparison under the shared non-GNN protocol.

For readability, we do not include long raw GNN configuration strings in the main leaderboard table. Instead, the learning-based methods are compared separately through compute-matched summaries and best-in-suite comparisons in §3.3. This keeps the main results section focused on interpretable method-level comparisons while preserving the paper's broader classical-versus-learning-based evaluation story.

Group	Method(s)	Implementation / key settings (summary)
OURS	OURS-MFAS, OURS-MFAS-INS1, OURS-MFAS-INS2, OURS-MFAS-INS3	Common implementation with Phase 1 local-ratio-style cycle breaking, Phase 2 stable weight-prioritized add-back, and optional score refinement. INS1/INS2/INS3 correspond to $P = 1/2/3$ add-back passes; INS3 is the main reported variant.
Classical baselines	SpringRank, SyncRank, SerialRank, BTL, DavidScore, EigenvectorCentrality, PageRank, RankCentrality, SVD-RS, SVD-NRS	Executed through the common non-neural evaluation pipeline using public implementations and default settings, without per-dataset hyperparameter tuning.
GNNRank	DIGRAC, ib	Evaluated under four fixed configurations per dataset: two backbones crossed with distance-based and proximal-baseline training settings; the proximal-baseline configuration uses distance-based pretraining.

Table 3: Compared method families and primary configurations used in the benchmark suite. Main tables report per-method results explicitly; any "best-of" summaries over GNN configurations are treated only as oracle envelopes for context.

Compute-matched comparison under a shared 1800 s budget. To reduce the effect of unequal compute budgets, Table 5 reports the same type of summary after restricting attention to runs that complete within the shared non-GNN wall-clock budget of 1800 s. The compute-matched set contains 79 datasets. This view is especially useful for assessing whether observed differences arise from ranking quality or from completion behavior under the fixed budget.

Within this restricted regime, the proposed variants remain competitive with the strongest classical baselines while maintaining explicit coverage accounting. As in the full-suite table, learning-based methods are handled separately through the cross-family comparisons reported later, rather than through raw configuration-heavy leaderboards in the main text.

Missingness and timeout summary. Coverage differences are an important part of the empirical picture, so Table 6 reports a compact dataset-level summary of missingness and timeouts by method family. The main limitation for the proposed method under the shared non-GNN budget is the Finance instance, which does not complete for any of the proposed variants. This explains the gap between the full suite size (80) and the OURS coverage (77) in the leaderboard tables.

Instance-level comparisons and best-of envelopes. Aggregate summaries can hide meaningful instance-level variation, so we also examine per-dataset win/tie/loss behavior using full-precision per-dataset means and a tie tolerance of $|\Delta| \leq 10^{-6}$. Our primary comparisons remain method-specific. When we additionally report compact "best-of" views, we use them only as oracle envelopes over fixed candidate sets, not as standalone algorithms. In particular, later summaries compare the proposed method against a classical oracle envelope and a GNN oracle envelope defined from the fixed candidate sets reported in the benchmark. These summaries provide context for what is achievable within each family while avoiding misleading conflation of multiple methods into a single reported algorithm.

Reproducibility note. To verify robustness after recent solver updates, we reran the proposed method on two tight-margin football instances and observed no change in upset-simple or runtime. The reported tables are generated from the same per-method CSV artifacts used throughout the analysis, and internal validation checks were used to confirm consistency of table coverage and missingness summaries.

Method	Median Upset-Simple	Mean Upset-Simple	Median Upset-Ratio	Mean Upset-Ratio	Median Upset-Naive	Mean Upset-Naive	Coverage
OURS-MFAS	0.878049	0.887880	0.505748	0.381891	0.219512	0.221970	77 / 80
OURS-MFAS-INS3	0.878049	0.887880	0.505748	0.381891	0.219512	0.221970	77 / 80
OURS-MFAS-INS2	0.878049	0.887914	0.505748	0.381874	0.219512	0.221979	77 / 80
OURS-MFAS-INS1	0.880927	0.888633	0.502815	0.381915	0.220232	0.222158	77 / 80
BTL	0.825000	0.835385	0.631922	0.575385	0.206250	0.208718	78 / 80
RankCentrality	1.000000	1.010641	0.646415	0.573846	0.250000	0.252949	78 / 80
SerialRank	1.075000	1.093333	0.307267	0.296795	0.268750	0.273333	78 / 80
SpringRank	1.675000	1.679615	0.476411	0.491154	0.418750	0.420256	78 / 80
SyncRank	1.825000	1.833462	0.476411	0.490641	0.456250	0.461795	78 / 80
PageRank	1.925000	1.934615	0.960686	0.973974	0.481250	0.488846	78 / 80
DavidScore	0.925000	0.939103	0.632356	0.578957	0.231250	0.234615	78 / 80
EigenvectorCentrality	1.825000	1.833077	0.960686	0.973623	0.456250	0.461603	78 / 80
SVD-RS	0.880927	0.890513	0.790599	0.550128	0.220232	0.222949	78 / 80
SVD-NRS	0.882925	0.898333	0.489421	0.380128	0.220732	0.224744	78 / 80

Table 4: Per-method leaderboard on the full benchmark suite for the proposed variants and classical baselines. Values are medians and means across datasets; lower is better for all losses. Coverage indicates the number of datasets contributing to each row.

3.3 Contribution and Practical Advantages

This subsection summarizes the practical contribution of the proposed method as a scalable, training-free ranking pipeline under fixed wall-clock budgets. The method constructs a ranking directly from the comparison graph by enforcing acyclicity, selectively restoring high-weight edges while preserving that acyclicity, and then extracting scores consistent with the recovered structure. Accordingly, we do *not* present the method as a new state-of-the-art algorithm for minimum weighted feedback arc set in the sense of new approximation guarantees. Rather, the contribution is a deterministic and deployable ranking pipeline grounded in the MWFAS viewpoint and evaluated under the same upset-based criteria used in recent ranking benchmarks.

Two empirical messages emerge from the benchmark. First, OURS-MFAS-INS3 (denoted **OURS**) is frequently competitive with strong classical ranking baselines on a per-dataset basis. Second, relative to training-based GNNRank configurations, **OURS** often matches or improves upset losses while providing a clear runtime advantage because it avoids training. Taken together, these findings support the paper's main practical claim: combinatorial acyclicity-based ranking can offer a favorable accuracy–runtime trade-off on large pairwise-comparison graphs.

Competitiveness relative to classical baselines. To place **OURS** relative to the strongest behavior observed among the classical methods, we compare it against a *best-in-suite* classical reference defined post hoc on each dataset as the minimum upset loss among the ten reported classical baselines. This reference is used only as context and is not treated as a standalone algorithm. The comparison is computed on the 77 datasets for which **OURS** returns a valid value under the non-GNN 1800 s budget.

Under upset-simple, **OURS** records 38 wins and 39 losses against the best-in-suite classical reference, with median gap 0.0244 and median relative gap 0.0303. It lies within 10% of the classical reference on 42/77 datasets. Under upset-ratio, **OURS** records 45 wins and 32 losses, with median relative gap -0.0781, and lies within 10% of the classical reference on 47/77 datasets. These results indicate that the proposed pipeline is often close to the strongest classical performance observed on a dataset, even though it is not tuned on a per-instance basis.

Method	Median Upset-Simple	Mean Upset-Simple	Median Upset-Ratio	Mean Upset-Ratio	Median Upset-Naive	Mean Upset-Naive	Coverage
OURS-MFAS	0.878049	0.887880	0.505748	0.381891	0.219512	0.221970	77 / 79
OURS-MFAS-INS3	0.878049	0.887880	0.505748	0.381891	0.219512	0.221970	77 / 79
OURS-MFAS-INS2	0.878049	0.887914	0.505748	0.381874	0.219512	0.221979	77 / 79
OURS-MFAS-INS1	0.880927	0.888633	0.502815	0.381915	0.220232	0.222158	77 / 79
SpringRank	0.802724	0.811757	0.569862	0.385207	0.200681	0.202939	78 / 79
BTL	0.825000	0.835385	0.631922	0.575385	0.206250	0.208718	78 / 79
SVD-RS	0.880927	0.890513	0.790599	0.550128	0.220232	0.222949	78 / 79
SVD-NRS	0.882925	0.898333	0.489421	0.380128	0.220732	0.224744	78 / 79
RankCentrality	1.000000	1.010641	0.646415	0.573846	0.250000	0.252949	78 / 79
SerialRank	1.075000	1.093333	0.307267	0.296795	0.268750	0.273333	78 / 79
DavidScore	0.925000	0.939103	0.632356	0.578957	0.231250	0.234615	78 / 79
PageRank	1.925000	1.934615	0.960686	0.973974	0.481250	0.488846	78 / 79
SyncRank	1.825000	1.833462	0.476411	0.490641	0.456250	0.461795	78 / 79

Table 5: Compute-matched per-method leaderboard under a shared 1800 s runtime budget. Coverage is computed over the 79 datasets with at least one recorded run within the budget. Lower is better for all losses.

Method family	Datasets with metrics	Datasets with runtime	Datasets with any timeout
OURS	77	77	1
classical	79	79	2
GNN	78	78	2
other	1	0	1

Table 6: Missingness and timeout audit by method family at the dataset level. Counts are computed from the same per-method artifacts used for the leaderboard summaries.

Accuracy relative to training-based GNNRank configurations. We also compare **OURS** against a *best-in-suite* GNN reference defined post hoc on each dataset as the minimum upset loss among the fixed set of reported GNN configurations. As above, this serves only as compact context and does not replace the per-configuration reporting in the main tables.

Across the same 77 datasets for which both **OURS** and the GNN reference are defined, **OURS** records 45 wins, 1 tie, and 31 losses under upset-simple, with median gap -0.1822 and median relative gap -0.1701. It is within 10% of the GNN reference on 48/77 datasets. Under upset-ratio, **OURS** records 47 wins and 30 losses, with median relative gap -0.2282, and is within 10% of the GNN reference on 47/77 datasets. Overall, these comparisons show that a training-free ranking pipeline can remain competitive even against stronger but substantially more expensive learning-based alternatives.

Runtime–accuracy trade-off and deployment advantages. The main operational advantage of **OURS** is that it avoids training and runs under the same non-GNN wall-clock budget as the classical baselines. To quantify this advantage, we compare **OURS** with the best-in-suite GNN configuration on each dataset for which both runtimes are recorded. On these 76 datasets, define the speedup

$$S = \frac{t_{\mathrm{GNN}}}{t_{\mathrm{OURS}}},$$

where t_{GNN} is the runtime of the selected GNN reference and t_{OURS} is the runtime of **OURS**.

Comparison (best-in-suite reference)	n	Metric	W/T/L	Median gap	Median rel. gap	Within 10%
OURS vs classical best-in-suite	77	upset-simple	38/0/39	0.0244	0.0303	42/77
OURS vs classical best-in-suite	77	upset-ratio	45/0/32	–	−0.0781	47/77
OURS vs GNN best-in-suite	77	upset-simple	45/1/31	−0.1822	−0.1701	48/77
OURS vs GNN best-in-suite	77	upset-ratio	47/0/30	–	−0.2282	47/77

Table 7: Summary of per-dataset comparisons using full-precision per-dataset means. "Best-in-suite" denotes a post-hoc per-dataset minimum over a fixed candidate set and is used only as context; it does not replace per-method or per-configuration reporting. Ties use $|\Delta| \leq 10^{-6}$.

Runtime summary (OURS vs GNN best-in-suite)	Value
Datasets with both runtimes	76
Speedup $S = t_{\mathrm{GNN}}/t_{\mathrm{OURS}}$: $P25$ / median / $P75$	4.72× / 10.16× / 18.18×
Speedup mean	62.69×
Datasets with $S \geq 10\times$ / $50\times$ / $100\times$	38 / 17 / 13
Pareto: better & faster / better but slower	45 / 0
Pareto: worse but faster / worse & slower	31 / 0

Table 8: Runtime–accuracy trade-off comparing **OURS** with the best observed GNN configuration per dataset over datasets with both runtimes recorded. Here the GNN runtime is taken from the best-in-suite post-hoc configuration on each dataset.

The resulting speedups are strongly favorable to **OURS**: the 25th percentile is 4.72×, the median is 10.16×, and the 75th percentile is 18.18×, with mean 62.69× due to a heavy tail of very slow GNN runs. Moreover, **OURS** achieves at least 10× speedup on 38/76 datasets, at least 50× on 17/76, and at least 100× on 13/76 datasets. Using upset-simple as the accuracy criterion, the Pareto breakdown over these 76 datasets is especially informative: **OURS** is *better and faster* on 45 datasets and *worse but faster* on 31 datasets, and it is never slower than the selected GNN configuration under the recorded protocol. This makes the deployment trade-off clear: the proposed method provides deterministic ranking with substantial runtime savings while still improving accuracy on a substantial fraction of instances.

Coverage and limitations. The method also has clear limitations. Under the 1800 s non-GNN budget, **OURS** does not complete on the largest extremely dense instances, most notably Finance, and the comparisons above are therefore reported only on datasets for which valid outputs are available. In addition, the dominant accuracy tail relative to both classical and GNN references is concentrated in the higher-resolution *finer basketball* instances. These behaviors are consistent with the design of the pipeline: Phase 1 repeatedly searches for cycles and removes arcs, while Phase 2 performs full edge scans in each pass, and both costs become more demanding as graph density increases. At the same time, finer temporal resolution can amplify local inconsistencies that are difficult to resolve using conservative acyclicity-preserving reinsertion alone. These limitations motivate future improvements such as strongly-connected-component-local refinement, reachability-aware insertion rules, and score-refinement procedures that avoid quadratic pair constructions on dense graphs.

4 Conclusion and Future Research

This paper presents a scalable, training-free ranking pipeline for weighted pairwise-comparison graphs based on the connection between ranking inconsistency and the Minimum Weighted Feedback Arc Set (MWFAS) formulation. The proposed method combines a local-ratio-style cycle-breaking stage to construct an acyclic backbone with a stable weight-prioritized add-back procedure (INS1/INS2/INS3 variants) that restores high-weight comparisons while preserving acyclicity. From the resulting graph, the pipeline produces real-valued scores whose induced order is evaluated under the same upset-based losses used in recent ranking benchmarks.

Across a benchmark suite of 80 instances under fixed wall-clock budgets, the proposed method is competitive with strong classical baselines, including BTL, spectral methods, and centrality-based methods, while also providing a consistent runtime advantage over training-based GNNRank configurations on instances where both sides are recorded. These results support the broader practical message of the paper: combinatorial acyclicity-based structure can serve as an effective, deterministic, and deployable alternative for ranking from pairwise comparisons when training cost, reproducibility, and runtime efficiency are important.

Several directions remain for future research. First, improved add-back and refinement strategies, such as reachability-aware insertion or targeted local search within strongly connected components, may reduce the remaining gap on the higher-resolution "finer" instances while preserving the time-bounded design. Second, better scalability on extremely dense graphs will likely require faster cycle detection, incremental maintenance of acyclicity constraints, and refinement procedures that avoid quadratic pair constructions. Third, it would be valuable to study principled tie-handling and uncertainty quantification when the comparison graph admits many valid topological orders or contains weakly connected components. Finally, an important direction is to design ranking objectives and calibration procedures that align more closely with downstream upset-based evaluation while retaining the computational advantages of acyclicity-based ranking.

Acknowledgments

Portions of this manuscript and the accompanying code were developed with the assistance of AI-based tools. In particular, ChatGPT (OpenAI) and Gemini were used to help draft and edit text and to support coding, and Cursor and GitHub Copilot were used to assist with software development. The author reviewed, verified, and takes full responsibility for the final manuscript and all reported results.

Statements and Declarations

Funding The author received no financial support for the research, authorship, and/or publication of this article.

Competing Interests The author declares that he has no competing interests.

Ethics Approval Not applicable. This article does not report studies involving human participants or animals.

Consent to Participate Not applicable.

Consent for Publication Not applicable.

Availability of Data and Materials The datasets used in the experiments originate from prior benchmarks cited in the manuscript (e.g., He et al. (2022)). For reproducibility, the processed versions used in this study are provided as text files containing weighted directed edge lists in the public repository linked below.

Code Availability All source code, processed data, and scripts required to reproduce the experiments are publicly available at `https://github.com/SoroushVahidi/ranking-by-feedback-arc-set`. The repository contains the implementation of the proposed method, the benchmark drivers, and the materials needed

to reproduce the reported results. Implementations of the non-proposed baselines and the GNNRank training and evaluation pipeline are based on the public GNNRank codebase of He et al. (2022), with the corresponding adaptations documented in the repository.

Author Contributions S. Vahidi is the sole author. He conceived the study, developed the algorithms, performed the experiments, and wrote the manuscript.

References

Baharev, A., Schichl, H., Neumaier, A., and Achterberg, T. (2021). An exact method for the minimum feedback arc set problem. *Journal of Experimental Algorithmics (JEA)*, 26.

Bar-Yehuda, R., Bendel, K., Freund, A., and Rawitz, D. (2004). Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463.

Bonacich, P. (1987). Power and centrality: A family of measures. *American Journal of Sociology*, 92(5):1170–1182.

Bradley, R. A. and Terry, M. E. (1952). Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Cavallaro, C., Cutello, V., and Pavone, M. (2024). Efficient heuristics to compute minimal and stable feedback arc sets. *Journal of Combinatorial Optimization*, 48(4):30.

Cavallaro, C., Cutello, V., and Pavone, M. (2025). Minimal and stable feedback arc sets and graph centrality measures. *Computers & Operations Research*, 184:107247.

Chen, P., Gao, C., and Zhang, A. Y. (2021). Optimal full ranking from pairwise comparisons.

Cucuringu, M. (2015). Sync-rank: Robust ranking, constrained ranking and rank aggregation via eigenvector and semidefinite programming synchronization.

d'Aspremont, A., Cucuringu, M., and Tyagi, H. (2021). Ranking and synchronization from pairwise measurements via SVD. *Journal of Machine Learning Research*, 22(19):1–63.

David, H. A. (1987). Ranking from unbalanced paired-comparison data. *Biometrika*, 74(2):432–436.

De Bacco, C., Larremore, D. B., and Moore, C. (2018). A physical model for efficient ranking in networks. *Science Advances*, 4(7):eaar8260.

Demetrescu, C. and Finocchi, I. (2003). Combinatorial algorithms for feedback problems in directed graphs. *Inf. Process. Lett.*, 86:129–136.

Even, G., Naor, J. S., Rao, S., and Schieber, B. (2000). Divide-and-conquer approximation algorithms via spreading metrics. *J. ACM*, 47(4):585–616.

Even, G., Naor, J. S., Schieber, B., and Sudan, M. (1995). Approximating minimum feedback sets and multicuts in directed graphs. In Balas, E. and Clausen, J., editors, *Integer Programming and Combinatorial Optimization*, pages 14–28, Berlin, Heidelberg. Springer Berlin Heidelberg.

Festa, P., Pardalos, P. M., and Resende, M. G. C. (2020). *Feedback Set Problems*, pages 1–13. Springer International Publishing, Cham.

Fogel, F., d'Aspremont, A., and Vojnovic, M. (2014). Serialrank: Spectral ranking using seriation. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Hassin, R. and Rubinstein, S. (1994). Approximations for the maximum acyclic subgraph problem. *Information Processing Letters*, 51(3):133–140.

He, Y., Gan, Q., Wipf, D., Reinert, G., Yan, J., and Cucuringu, M. (2022). Gnnrank: Learning global rankings from pairwise comparisons via directed graph neural networks.

He, Y., Reinert, G., and Cucuringu, M. (2021). Digrac: Digraph clustering based on flow imbalance.

Hecht, M., Gonciarz, K., and Horvát, S. (2021). Tight localizations of feedback sets. *ACM J. Exp. Algorithmics*, 26.

Hochberg, Y. and Rabinovitch, R. (2000). Ranking by pairwise comparisons with special reference to ordering portfolios. *American Journal of Mathematical and Management Sciences*, 20(3-4):231–253.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

Negahban, S., Oh, S., and Shah, D. (2017). Rank centrality: Ranking from pairwise comparisons. *Oper. Res.*, 65(1):266–287.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab.

Raman, V. and Saurabh, S. (2006). Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458.

Shah, N. B. and Wainwright, M. J. (2016). Simple, robust and optimal ranking from pairwise comparisons.

Shea-Blymyer, C. and Young, J. M. (2021). Inspectable incrementality in minimum feedback arc set solving. In *Proceedings of ACM Conference*, pages 1–8. Association for Computing Machinery.

Tong, Z., Liang, Y., Sun, C., Li, X., Rosenblum, D. S., and Lim, A. (2020). Digraph inception convolutional networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 17907–17918.