Rocket-Crane Algorithm for the Feedback Arc Set Problem

David A. Bader[1*], Justin Ellis-Joyce[2,3*], Gert-Jan Both[2], Srinivas C. Turaga[2], Harinarayan Asoori Sriram[4], Srijith Chinthalapudi[4], Zhihui Du[1*]

[1]Department of Data Science, New Jersey Institute of Technology, University Heights, Newark, 07102, New Jersey, USA.
[2]HHMI Janelia Research Campus, 19700 Helix Dr, Ashburn, 10587, Virginia, USA.
[3]Department of Neuroscience, Johns Hopkins University, 1003 Wood Basic Science Building, Baltimore, 21218, Maryland, USA.
[4]Edison Academy Magnet School, 100 Technology Dr, Edison, 08837, New Jersey, USA.

*Corresponding author(s). E-mail(s): david.bader@njit.edu; ellisj1@janelia.hhmi.org; zhihui.du@njit.edu;
Contributing authors: bothg@janelia.hhmi.org; turagas@janelia.hhmi.org; harinarayansriram@gmail.com; srijith.srinivas@gmail.com;

Abstract

Understanding information flow in the brain can be facilitated by arranging neurons in the fly connectome to form a maximally "feedforward" structure. This task is naturally formulated as the Minimum Feedback Arc Set (MFAS)–a well-known NP-hard problem, especially for large-scale graphs. To address this, we propose the Rocket-Crane algorithm, an efficient two-phase method for solving MFAS. In the first phase, we develop a continuous-space optimization method that rapidly generates excellent solutions. In the second phase, we refine these solutions through advanced exploration techniques that integrate randomized and heuristic strategies to effectively escape local minima. Extensive experiments demonstrate that Rocket-Crane outperforms state-of-the-art methods in terms of solution quality, scalability, and computational efficiency. On the primary benchmark–the fly connectome–our method achieved a feedforward arc set with a total forward weight of 35,459,266 (about 85%), the highest among all competing methods. The algorithm is open-source and available on GitHub.

Keywords: Feedback Arc Set Problem, Graph Algorithms, Large Data Analysis

1 Introduction

Understanding the direction of information flow in the *Drosophila* brain, or connectome, is a fundamental goal of the Minimum Feedback Arc Set Challenge[1]. This task can be formalized as the Minimum Feedback Arc Set (MFAS) problem in graph theory, where the objective is to identify a minimal subset of edges whose removal eliminates all cycles in a directed graph, resulting in a directed acyclic graph (DAG).

The MFAS problem is well-known for its computational complexity. Lawler [1] established its NP-hardness, and Kann [2] further proved its APX-hardness, indicating that constant-factor approximations are unlikely unless P = NP. Additionally, Karp [3] demonstrated that the decision version of MFAS is NP-complete. These theoretical results collectively underscore the intrinsic difficulty of developing efficient algorithms for MFAS, particularly on large-scale graphs.

In practice, solving the MFAS problem involves three major challenges: scalability, solution quality, and computational efficiency.

Scalability: Large-scale graphs, such as those derived from neural connectomes, pose significant memory and runtime constraints. While solvers like Gurobi[2] and CPLEX[3] can, in theory, solve MFAS using Mixed-Integer Programming (MIP), their performance is often hindered by high memory consumption and prolonged execution times. Even with memory-efficient formulations such as linear arrangement (see Section 2), computing an optimal solution remains impractical within acceptable time constraints.

Existing heuristic methods also suffer from poor scalability. For example, the recent connectivity matrix approach by Borst [4] shows promise but is limited to small graphs. Divide-and-conquer techniques like Stochastic Evolution (SE) and Dynamic Clustering (DC) [5] struggle with large graphs due to memory overflow or recursion depth issues, making them unsuitable for practical use.

Solution Quality: Achieving near-optimal solutions is essential, especially given the limitations of exact solvers. Dinur et al. [6] proved that, unless P = NP, no algorithm can approximate MFAS with a factor strictly better than 1.36067, thereby setting a strong hardness barrier for approximation algorithms. Many fast heuristics [7] prioritize speed at the expense of solution quality, often removing a significantly larger fraction of the edge weight–approximately 30% compared to our 15% (see Section 4.3)–highlighting a substantial performance gap.

Computational Efficiency: Some algorithms, like simulated annealing [8], can theoretically reach optimal or near-optimal solutions but often require impractical

[1]https://codex.flywire.ai/app/mfas_challenge
[2]https://www.gurobi.com/
[3]https://www.ibm.com/products/ilog-cplex-optimization-studio

computation times. In real-world applications, high-quality results must be delivered within acceptable timeframes, making runtime efficiency a critical consideration.

Our Contribution. To overcome these challenges, we propose the **Rocket-Crane** algorithm–a novel two-phase framework that efficiently solves the MFAS problem on large graphs while maintaining high solution quality. Our main contributions are:

- We present a simple yet powerful linear arrangement formulation for MFAS, which serves as the foundation for a memory-efficient algorithm capable of scaling to large graphs.
- We develop the **Rocket-Crane** algorithm, combining a fast, gradient-based optimization phase (Rocket) with a high-quality refinement phase (Crane) that leverages randomized and heuristic strategies to escape local minima.
- We conduct extensive experiments comparing Rocket-Crane with state-of-the-art methods. Our results demonstrate superior scalability, runtime performance, and solution quality across real-world large-scale datasets. The implementation is publicly available as open-source on GitHub.

2 Problem Formulation

The Minimum Feedback Arc Set (MFAS) problem aims to identify and remove cycles in a directed graph to transform it into a directed acyclic graph (DAG). Given a directed, weighted graph $G = (V, E, W)$, where V is the vertex set, E is the edge set, and W is the weight function, let $n = |V|$ denote the number of vertices and $m = |E|$ the number of edges. For each directed edge $(u, v) \in E$, its weight is given by $w(u, v)$. The MFAS problem involves removing a minimal-weight set of edges to eliminate all cycles.

2.1 Cycle-Based Formulation

A natural approach to the MFAS problem is to explicitly identify cycles and ensure that at least one edge is removed from each. We define a binary variable $e(u, v)$ for every edge $(u, v) \in E$, where:

$$e(u,v) = \begin{cases} 1, & \text{if } (u,v) \text{ is part of the feedback arc set,} \\ 0, & \text{otherwise.} \end{cases}$$

Let C denote the set of all cycles in G. To guarantee cycle elimination, at least one edge from each cycle $c = (c_1, c_2, \dots, c_l) \in C$ must be removed:

$$\min \sum_{(u,v)\in E} e(u,v) \cdot w(u,v) \tag{1}$$

Subject to:

$$\forall c \in C, \sum_{(u,v)\in c} e(u,v) \geq 1. \tag{2}$$

For special cases, such as triangles, alternative constraints (e.g., triangle constraints) can be used. While this formulation is intuitive and straightforward, its practicality is severely limited by the exponential growth of the number of cycles.

In a dense graph with n vertices, the maximum number of cycles of length ≥ 2 is given by:

$$\sum_{k=2}^{n} \binom{n}{k} \cdot (k-1)!.$$

This results in an impractical number of constraints in Equation 2, making it infeasible for exact algorithms to handle large graphs. The key limitations of this approach include: (1) Explicit cycle enumeration is often computationally expensive and impractical for large graphs. (2) The formulation requires excessive memory for large graphs, which is often prohibitive in real-world applications.

Due to these challenges, alternative formulations are necessary to solve the MFAS problem efficiently for large graphs.

2.2 Linear Arrangement Formulation

Younger [9] proposed an alternative approach based on vertex ordering, known as the *linear arrangement* method. In this formulation, a directed graph is represented as a sequential ordering of its vertices, where an edge (u, v) is classified as a *feedforward* edge if u precedes v in the order. Otherwise, it is considered a *feedback* edge. Removing all feedback edges produces a DAG and corresponds to solving the MFAS problem.

Since every feedback arc set corresponds to a feedforward arc set in a linear arrangement, minimizing the feedback arc set is equivalent to maximizing the feedforward arc set. This equivalence allows us to reformulate the problem by optimizing the feedforward arc set over all possible linear arrangements.

Let l_v denote the position of vertex v in a permutation of the n vertices, where l_v is a unique value in $[0, n-1]$. We redefine the binary variable $e(u, v)$ as follows:

$$e(u,v) = \begin{cases} 1, & \text{if } (u,v) \text{ belongs to the feedforward arc set,} \\ 0, & \text{otherwise.} \end{cases}$$

The problem can now be formulated as:

$$\max \sum_{(u,v)\in E} e(u,v) \cdot w(u,v) \tag{3}$$

Subject to:

$$l_u + 1 \leq l_v + n \cdot (1 - e(u,v)), \quad \forall (u,v) \in E. \tag{4}$$

This formulation ensures:

- If $e(u, v) = 1$, then $l_u < l_v$ (or $l_u + 1 \leq l_v$), maintaining the feedforward property.
- If $e(u, v) = 0$, then the constraint $l_u + 1 \leq l_v + n$ always holds trivially.

The linear arrangement formulation requires only $(m + n)$ variables and m constraints, resulting in a memory complexity of $\mathcal{O}(m + n)$. This makes it significantly more scalable for large real-world graphs compared to the cycle-based approach.

However, this approach remains computationally challenging. The number of possible vertex orderings is $n!$, implying that an exact algorithm would need to evaluate all $n!$ permutations to ensure an optimal solution. To address this, numerous heuristic algorithms have been developed to find high-quality approximate solutions efficiently [4, 5, 7, 10–14].

In this work, we focus on a real-world instance of the MFAS problem and demonstrate the development of an efficient heuristic algorithm with strong performance guarantees to tackle this computational challenge.

3 Rocket-Crane Algorithm

Heuristic algorithms [15–17] operate based on a "rule of thumb," guiding the search for solutions by leveraging shortcuts, approximations, or patterns derived from prior knowledge. By reducing the problem space and focusing on the most promising candidates, heuristic methods achieve rapid solution discovery. However, their primary drawback is the tendency to become trapped in local optima, failing to reach the global optimum. The key challenge in heuristic algorithms lies in balancing efficiency and solution quality–using simple rules to improve solutions quickly while avoiding stagnation in suboptimal regions. Once a heuristic algorithm reaches a local optimum, further progress often becomes increasingly difficult, constrained by an upper bound beyond which improvements are infeasible.

Randomized algorithms [18, 19], such as Simulated Annealing [8] and Monte Carlo [20] methods, introduce stochasticity into the decision-making process to enhance solution diversity and accelerate convergence. Unlike heuristics that follow a fixed set of rules, randomized algorithms explore multiple solution paths, increasing the likelihood of escaping local optima. This capability makes them particularly effective for NP-hard problems, where deterministic approaches may struggle. The primary challenge, however, lies in efficiently managing randomness to maximize the probability of discovering better solutions.

The proposed Rocket-Crane algorithm can achieve both rapid improvement and critical refinement. Initially, a highly optimized heuristic–the *Rocket* sub-algorithm–rapidly enhances solution quality, analogous to the swift ascent of a rocket. However, further improvements become difficult once the heuristic approach reaches its limit. At this stage, the *Crane* sub-algorithm takes over, employing more sophisticated and often computationally intensive strategies to refine the solution further. This phase is akin to lifting a heavy load with a crane–requiring more effort but enabling progress beyond the limitations of heuristic techniques.

The Rocket-Crane algorithm strategically integrates heuristic optimization with randomized search to balance computational efficiency and solution quality, promoting both fast convergence and resilience against local optima. As discussed in Section 2, the linear arrangement formulation is particularly well-suited for large graphs. Accordingly, our *Rocket-Crane* algorithm is built upon this formulation.

3.1 Rocket Sub-Algorithm

The Minimum Feedback Arc Set (MFAS) problem, as formulated in Eqs. 3 and 4, takes the form of a standard integer programming problem. By relaxing the integer constraints and allowing the variables to take continuous real values, the problem can be approximated as a linear programming (LP) problem. This relaxed LP formulation can be efficiently solved using gradient-based optimization methods, which iteratively refine the solution. Although the LP solution does not guarantee an exact answer to the original integer problem, it often serves as a strong approximation. Our Rocket sub-algorithm builds upon this principle and introduces two key innovations to improve both efficiency and solution quality:

Linear Arrangement Formulation: Our linear arrangement formulation based sparse representation reduces memory requirements from $\mathcal{O}(n^2)$ in connectivity matrix approaches [4] to $\mathcal{O}(m+n)$, enabling handling of graphs with more than 10^6 edges. This eliminates memory exhaustion issues encountered by previous methods on large inputs. **Objective Alignment Mapping:** Our approach preserves the relative ordering between discrete and continuous objectives by carefully designing the loss function and the mapping between solution spaces. We observe that the Heaviside and Pauli terms introduced in [4] increase computational cost without substantially improving the quality of the resulting feedforward sorting. In contrast, our method achieves competitive MFAS performance using a simpler and more efficient loss function.

3.1.1 Continuous-Space Formulation

A solution to the MFAS problem is a permutation $\pi : V \to \{0, \dots, n-1\}$, representing a linear ordering of vertices. An edge (u, v) is considered feedforward if $\pi(u) < \pi(v)$. To enable gradient-based optimization, we relax this discrete formulation by assigning each vertex a real-valued position: $l : V \to \mathbb{R}$, while preserving the order isomorphism:

$$\pi(u) < \pi(v) \Leftrightarrow l(u) < l(v), \quad \forall u, v \in V. \tag{5}$$

The continuous objective approximates the total weight of feedforward edges using a differentiable sigmoid function:

$$\max \sum_{(u,v)\in E} \sigma_\beta(l(v) - l(u)) \cdot w(u, v), \tag{6}$$

where $\sigma_\beta(x) = \frac{1}{1+e^{-\beta x}}$ is the sigmoid function with steepness parameter $\beta > 0$. This provides a smooth approximation of the binary indicator used in Eq. 3, estimating whether an edge (u, v) is feedforward (i.e., whether $l(u) < l(v)$).

The associated loss function for optimization is defined as:

$$\mathcal{L} = -\sum_{(u,v)\in E} \sigma_\beta(l(v) - l(u)) \cdot \hat{w}(u, v), \tag{7}$$

where $\hat{w}(u, v)$ denotes the max-normalized edge weights. This formulation offers several key properties:

Smoothness: The sigmoid function σ_β is $\mathcal{C}^\infty$-smooth, making it suitable for gradient-based optimization. **Scale Invariance:** Weight normalization ensures stable gradients across graphs with varying edge weight magnitudes. **Binary Approximation:** The sigmoid function provides a smooth but accurate approximation of binary edge orientation, effectively estimating whether an edge is feedforward in a continuous setting.

3.1.2 Optimization Algorithm

Alg. 1 provides a detailed description of the proposed Rocket sub-algorithm, designed to solve the MFAS problem within a continuous space framework.

The algorithm operates as follows:

Initialization (Lines 1 to 3): The position vector P is initialized by assigning each vertex in the graph $G = (V, E, W)$ a unique random value in $\mathbb{R}$. The initial score of the current mapping, denoted as $current\_score$, is computed based on P. The variable $best\_score$ is set to $current\_score$, and the iteration counter $iter$ is initialized to 0.

Optimization Loop (Lines 4 to 16): The core optimization procedure iterates until the maximum iteration limit $Threshold$ is reached. Within each iteration: In Line 5, the position values of the source and target vertices for each edge $(u, v) \in E$ are extracted into vectors PS and PT, respectively. In Line 6, the difference $\Delta = PT - PS$ is computed, where each element $\delta(u, v) = l(v) - l(u)$ indicates whether the edge (u, v) belongs to the feedforward arc set ($\delta(u, v) > 0$) or the feedback arc set ($\delta(u, v) < 0$). In Line 7, a sigmoid transformation Sig is applied to Δ, defined as $Sig = \frac{1}{1+e^{-\beta\Delta}}$, where $\beta > 0$ is a scaling parameter controlling the steepness of the sigmoid function. In Line 8, the loss value is calculated as $Loss = -\sum_{(u,v)\in E} Sig(u, v) \times \hat{w}(u, v)$, where $\hat{w}(u, v)$ are max-normalized edge weights from $w(u, v)$. In Line 9, the Adam optimizer performs a single optimization step to update the position vector P and minimize the loss. In Line 10, the score of the updated position vector P is recalculated as $current\_score$. If $current\_score > best\_score$ (Line 11), the current mapping P is output as an improved solution (Line 12), and $best\_score$ is updated (Line 13). The iteration counter $iter$ is incremented (Line 15).

Output (Line 17): Upon completion of the loop, the algorithm returns the best position vector P found during optimization, which represents the optimal continuous-space mapping of vertices.

This approach efficiently solves large-scale MFAS problems by leveraging the smoothness of the sigmoid function and the power of gradient-based optimization, ensuring alignment with the discrete-space objective of maximizing feedforward arcs.

3.2 Crane Sub-Algorithm: A Complementary Randomized-Heuristic Approach

Heuristic methods frequently exhibit diminishing returns in solution quality due to deterministic search biases or local optima entrapment. In contrast, randomized algorithms leverage stochastic exploration to escape suboptimal regions, making them natural complements to heuristic frameworks. We propose the Crane sub-algorithm for the Minimum Feedback Arc Set (MFAS) problem, which synergistically integrates three components: (1) a topologically initialized Simulated Annealing (SA), (2) Monte

Algorithm 1 Rocket Sub-Algorithm for Gradient-based MFAS Optimization

```
    Input: Graph $G = (V, E, W)$, where $V$ is the vertex set, $E$ is the edge set, and $W$ is the edge weight set; maximum iterations $Threshold$; scaling parameter $\beta$.
    Output: Optimized position vector $P$ in continuous space.
 1: Initialize position vector $P$ with unique random values in $\mathbb{R}$ for each vertex in $V$
 2: Calculate initial mapping score $current\_score$ based on $P$
 3: $best\_score \leftarrow current\_score$, $iter \leftarrow 0$
 4: while $iter < Threshold$ do
 5:     Extract position values $PS$ and $PT$ for source and target vertices of each edge $(u, v) \in E$ from $P$
 6:     Compute $\Delta = PT - PS$, where $\Delta(u, v) = l(v) - l(u)$ for each edge
 7:     Compute sigmoid transformation $Sig = \frac{1}{1+e^{-\beta\Delta}}$
 8:     Compute loss $Loss = -\sum_{(u,v)\in E} Sig(u, v) \times \hat{w}(u, v)$
 9:     Use Adam optimizer to update $P$ and minimize $Loss$
10:     Recalculate mapping score $current\_score$ based on updated $P$
11:     if $current\_score > best\_score$ then
12:         Output current mapping $P$ as improved solution
13:         $best\_score \leftarrow current\_score$
14:     end if
15:     $iter \leftarrow iter + 1$
16: end while
17: return $P$
```

Carlo (MC) sampling, and (3) a Mixed Integer Programming (MIP) heuristic. This hybridization exploits the complementary strengths of global exploration and local intensification, achieving superior performance compared to single-method baselines.

3.2.1 Topologically Initialized Simulated Annealing

Simulated Annealing (SA) is a canonical stochastic optimization technique for navigating complex solution landscapes while avoiding premature convergence [8]. We enhance SA with a problem-specific initialization strategy:

Topological Sorting (TopoShuffle) Initialization. Given an initial permutation P, we construct a new Directed Acyclic Graph $DAG = (V, E')$, where $E' \subsetneq E$ and apply Kahn's topological sorting algorithm [21] to generate a new permutation TSP. Formally, for all edges $(u, v) \in E$, u precedes v in TSP, ensuring $\Gamma(TSP) \geq \Gamma(P)$, where $\Gamma(\cdot)$ denotes the MFAS objective score (i.e., the total weight of preserved feedforward arcs). This initialization provides diverse starting configurations for the following optimization while guaranteeing solution quality that is at least as good as, and potentially better than, the original.

3.2.2 MIP Heuristic Integration

Using the linear arrangement formulation in Section 2.2, we refine solutions via an MIP solver (e.g., Gurobi). Given an initial feasible solution S_0, the solver executes heuristic strategies (e.g., feasibility pumps, objective diving) under a fixed time budget $t_{\max}$, prioritizing feasible solutions over optimality gaps. The MIP component intensifies the search in promising regions identified by SA.

3.2.3 Monte Carlo Sampling

Monte Carlo sampling provides lightweight diversification by performing greedy stochastic swaps. For vertices $u, v \in V$, the transition probability to candidate solution S' is:

$$P(S_{t+1} = S') = \begin{cases} 1 & \text{if } \Gamma(S') > \Gamma(S_t), \\ 0 & \text{otherwise.} \end{cases}$$

This mechanism efficiently explores adjacent solutions with minimal computational overhead.

3.2.4 Hybrid Policy

Alg. 2 formalizes the Crane sub-algorithm, which alternates between SA, MIP heuristics, and MC sampling using an adaptive policy. The algorithm executes each method until a stagnation criterion is met (e.g., no improvement for k iterations), then switches to the next method. This balances: (1) SA's thermal fluctuation-driven escape from local minima, (2) MIP's problem-structure-aware intensification, (3) MC's rapid neighborhood exploration.

Algorithm 2 Crane Sub-Algorithm for MFAS Optimization

```
    Input: Graph $G$, initial permutation $P$, cooling rate $\alpha$, iteration thresholds $N_{SA}$, $N_{MC}$, MIP time budget $t_{max}$.
    Output: Optimized permutation $P_{best}$.
 1: repeat
    TopoShuffle Initialization:
 2:     Construct a $DAG$ based on $G$ from $P$
 3:     Generate $TSP$ via topological sorting on the $DAG$
 4:     Initialize $P_{best} \leftarrow TSP$, temperature $T \leftarrow T_0$, $iter \leftarrow 0$
    SA Global Exploration:
 5:     for $iter \leftarrow 1$ to $N_{SA}$ do
 6:         Randomly select $u, v \in V$; compute $\delta = \Gamma(S') - \Gamma(TSP)$
 7:         Update $T \leftarrow T \cdot \alpha$
 8:         if $\delta \geq 0$ or accept with probability $e^{-\delta/T}$ then
 9:             $TSP \leftarrow S'$
10:             if $\Gamma(TSP) > \Gamma(P_{best})$ then
11:                 $P_{best} \leftarrow TSP$
12:             end if
13:         end if
14:     end for
    MIP Intensification:
15:     $P_{best} \leftarrow$ MIP_Heuristic($P_{best}$, $t_{max}$)
    MC Local Diversification:
16:     for $iter \leftarrow 1$ to $N_{MC}$ do
17:         Randomly select $u, v \in V$; compute $\delta = \Gamma(S') - \Gamma(P_{best})$
18:         if $\delta > 0$ then
19:             $P_{best} \leftarrow S'$
20:         end if
21:     end for
22: until convergence or termination condition
23: return $P_{best}$
```

The workflow of Alg. 2 is as follows: The whole algorithm is implemented in a Repeat-Termination loop, which continues until convergence–typically when $\Gamma(P_{best})$ plateaus–or until an external stopping criterion is met (Lines 1–22). In the loop, it first performs **TopoShuffle Initialization** by constructing a directed acyclic graph (DAG) from the input permutation P and generating a new topologically sorted permutation TSP (Lines 2–4). This is followed by **SA Global Exploration**, where the algorithm iteratively perturbs TSP using Simulated Annealing (SA), guided by a geometric cooling schedule with decay rate $\alpha = 0.95$ (Lines 5–14). Next, **MIP Intensification** is applied to refine the best solution P_{best} using mixed-integer programming (MIP) heuristics within a predefined time limit t_{max} (Line 15). Finally, the algorithm uses **MC Local Diversification**, which explores neighboring solutions through greedy swaps to further improve local optima (Lines 16–21).

This hybrid strategy combines SA's global exploration, MIP's intensification, and MC's local diversification, achieving robust performance on large-scale MFAS instances (validated in Section 4.5).

4 Experimental Results

4.1 Experimental Environment and Dataset Characteristics

All experiments were conducted on a high-performance computing node within the NJIT Wulver cluster[4]. The computational node is equipped with 128 CPU cores, 512 GB of memory, and Dual NVIDIA A100 Tensor Core GPUs (80 GB HBM2 memory each), offering substantial parallel computing capabilities required for the efficient execution of large-scale algorithms.

Our JAX-based implementation of the Rocket sub-algorithm is designed to be hardware-agnostic, supporting execution on both CPU and GPU. This enables seamless hardware acceleration when GPU resources are available, significantly reducing computation time.

The dataset[5] used in our study is a large-scale directed graph representing the connectome of a fly. It consists of 136,648 vertices and 5,657,719 edges, resulting in a graph density of 0.000303, indicative of its sparsity. In this graph, vertices correspond to individual neurons, and weighted edges denote synaptic connections, where edge weights reflect the strength of these connections. The input is structured as an edge list comprising triplets of the form `(source, target, weight)`.

The graph consists of 9,626 strongly connected components (SCCs), the majority of which are relatively small. However, a single giant component dominates the structure, containing 126,840 vertices–92.8% of the entire graph. This massive component presents the primary computational challenge, as it prevents the problem from being easily decomposed into smaller, more manageable SCCs.

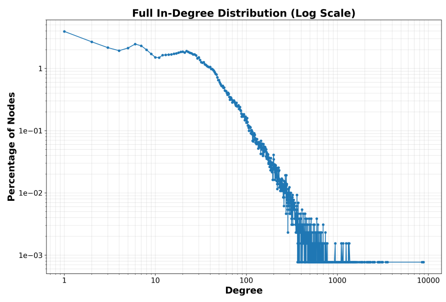

Fig. 1 In-degree distribution across all vertices. The distribution exhibits a heavy tail, with most vertices having low in-degrees and a small number exhibiting extremely high in-degrees, a feature typical of real-world networks.

Fig. 1 illustrates the in-degree distribution of the graph. The heavy-tailed nature of the distribution implies that while the majority of vertices have low in-degrees, a few serve as hubs with extremely high connectivity. Such hubs are likely to participate in multiple cycles, complicating the task of identifying a minimal set of edges for removal.

[4]https://hpc.njit.edu/clusters/wulver/
[5]https://codex.flywire.ai/app/mfas_challenge

The average in- and out-degrees are both 41.40, with maximum in- and out-degrees of 8,840 and 8,064, respectively.

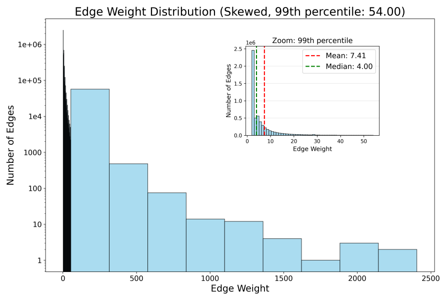

Fig. 2 Edge weight distribution. The histogram shows the skewness, with most edges concentrated at lower weights. The inset zooms into the 99th percentile (weight = 54.00), highlighting the long tail of the distribution.

Fig. 2 presents the distribution of edge weights. The distribution is highly skewed, with most weights concentrated near the lower end of the spectrum. The minimum edge weight is 2, the maximum is 2,405, the mean is 7.41, and the median is 4.00. The 99th percentile weight is 54.00, and the standard deviation is 12.77. This long-tailed distribution presents a fundamental challenge for identifying a high-weight feedforward (acyclic) arc set. While a few high-weight edges contribute disproportionately to the total edge weight, including them in the solution may introduce cycles, forcing trade-offs with many lower-weight alternatives. Conversely, although low-weight edges are individually less valuable, they are abundant and structurally dispersed, potentially forming large acyclic subgraphs if carefully selected. The irregularity and density of these small weights make it difficult to apply simple heuristics for prioritizing edge inclusion. As a result, designing an effective optimization strategy requires balancing the retention of high-weight edges with the need to maintain acyclicity over a vast, sparsely weighted graph.

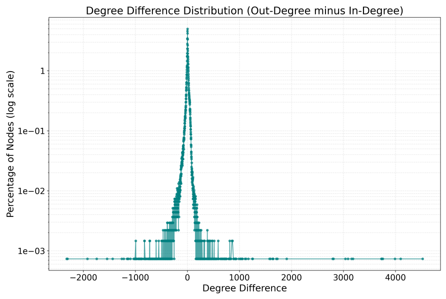

Fig. 3 Distribution of degree differences (out-degree minus in-degree). The distribution is centered around zero, indicating that most vertices have approximately balanced in- and out-degrees.

Fig. 3 shows the distribution of degree differences, computed as the out-degree minus in-degree for each vertex. A pronounced peak at zero suggests that a large number of vertices have balanced degrees. Specifically, 4,722 vertices (3.46%) exhibit a degree difference of exactly zero. The distribution is roughly symmetric, with 77,789 vertices (56.93%) having more outgoing than incoming edges, and 54,137 vertices (39.62%) having the reverse. This balance limits the effectiveness of heuristics based solely on degree differences, such as those proposed in [10].

Regarding vertex classification, the graph contains 6,651 source vertices (4.87%) with zero in-degree, 2,447 sink vertices (1.79%) with zero out-degree, and 127,550 regular vertices (93.34%) with both non-zero in- and out-degrees. The overwhelming prevalence of regular vertices presents another challenge: strategies that rely on selectively removing sources or sinks are unlikely to be sufficient for solving the problem.

In summary, the dataset's large scale, structural sparsity, heterogeneous degree distribution, and highly imbalanced edge weights collectively undermine the effectiveness of simple heuristics. These challenges highlight the need for novel methodologies capable of handling the complexity and scale inherent to real-world graphs.

4.2 Evaluation Metrics

We evaluate algorithmic performance based on the total weight of the *feedforward arc set*, either in absolute terms or as a percentage of the total edge weight in the graph. A higher feedforward weight or percentage reflects a more effective algorithm, as it indicates greater success in retaining high-weight acyclic substructures.

Performance is assessed from three complementary perspectives:

- **Time to Good (TTG)** – The time required for the algorithm to reach a solution of reasonably good quality. This metric is relevant in time-sensitive applications where approximate solutions are acceptable.
- **Time to Excellent (TTE)** – The time taken to obtain a high-quality solution that satisfies stringent quality thresholds. This is critical in contexts where only near-optimal or high-quality solutions are acceptable.
- **Time to Best (TTB)** – The time required to discover the best solution within a fixed time budget. This metric is both the most challenging to optimize and the primary focus of our evaluation, as it reflects the algorithm's peak performance under time constraints. It highlights the trade-off between solution quality and computational efficiency.

To benchmark solution quality, we adopt the *linear arrangement method* for the feedback arc set problem. A random permutation of the vertices partitions the edges into two acyclic subsets: a *feedforward edge set* and a *feedback edge set*. We select the subset with the larger total weight as the final feedforward arc set. By construction, this yields a baseline feedforward weight of at least **50**% of the total, serving as the minimum standard for acceptable performance.

Based on empirical observations from preliminary experiments, we establish the following quality thresholds:

- A solution achieving at least **75%** of the total edge weight is considered **good quality**.
- A solution achieving at least **80%** is considered **excellent quality**.

Ultimately, our objective is to **identify the highest-quality feedforward arc set within a reasonable computation time**, emphasizing algorithmic efficiency and scalability for solving NP-hard instances on large-scale, complex graphs.

4.3 Comparison of Heuristic Algorithms

To rigorously evaluate the performance of our novel Rocket-Crane algorithm, we compare it against a comprehensive selection of heuristic algorithms designed to solve the Minimum Feedback Arc Set (MFAS) problem. This evaluation provides a detailed understanding of the relative advantages and limitations of each approach. Table 1 presents the comparative results. In this table, "Score" denotes the total weight of the feedforward arc set (larger is better), "Percentage" represents the score as a fraction of the total edge weight, "Exec Time" reports the execution time (in seconds) on our experimental platform, and "Time Complexity" indicates the theoretical computational complexity. Here, n is the number of vertices, m is the number of edges, and t denotes the number of iterations (if applicable).

Table 1 Performance Comparison with Fifteen Different Heuristic Algorithms

	SE	**DC**	**Greedy**	**GreedyAbs**	**Simple**	**BergeShor**	**DFS**	**KwikSort**	**Sort**	**Sift**	**KwikSort***	**Sort***	**Sift***	**Tight-Cut***	**RASstar**	**Rocket-Crane**
Score	NA	NA	29025804	30646022	21028911	25390953	23702336	21392424	30131918	NA	22743824	31949612	NA	NA	33014221	33550730
Percentage (%)	NA	NA	69.25	73.12	50.17	60.58	56.55	51.04	71.89	NA	54.27	76.23	NA	NA	78.77	80.05
Exec Time (s)	NA	NA	49	38	3	4	33	20	4284	NA	2229	21082	NA	NA	9	4.2
Time Complexity	$\mathcal{O}(n^4)$	$\mathcal{O}(mn\log n)$	$\mathcal{O}(m+n)$	$\mathcal{O}(m+n)$	$\mathcal{O}(m+n)$	$\mathcal{O}(m+n)$	$\mathcal{O}(m+n)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(nm^4)$	$\mathcal{O}(mn)$	$\mathcal{O}(t(m+n))$

Note: "NA" indicates that the algorithm failed to produce a feasible solution within a 24-hour runtime.

4.3.1 Stochastic Evolution (SE) and Dynamic Clustering (DC) Algorithms

Saab [5] introduced two divide-and-conquer-based algorithms for the MFAS problem: Stochastic Evolution (SE) and Dynamic Clustering (DC). Unlike many heuristic methods that rely on vertex linearization, SE and DC approach the problem by recursively partitioning the graph and removing feedback arcs based on minimum bisection heuristics, a task that is itself NP-hard. Due to the absence of publicly available implementations, we developed our own Python versions, carefully following the methodological details described in the original paper.

The SE algorithm employs a stochastic evolution strategy, in which graph partitions are iteratively perturbed using randomized mutations and crossover-like operations to reduce the total feedback arc weight. This evolutionary process requires repeated evaluations over all n vertices and results in a time complexity of $\mathcal{O}(n^4)$, making it computationally expensive even for moderately sized graphs.

In contrast, the DC algorithm adopts a more structured approach by dynamically clustering vertices and recursively refining the partitions based on connectivity. Its complexity, $\mathcal{O}(mn \log n)$, is lower than SE's, as the recursive bisection depth scales with $\log n$ while still considering all edges during each partitioning step.

Empirical results in [5] show that SE and DC can outperform Greedy-based algorithms on small graphs with up to a few thousand vertices. However, when applied to our real-world connectome graph comprising 136,648 vertices and over 5.6 million edges, neither SE nor DC returned a feasible solution after more than two days of computation. The excessive computational burden, especially SE's quartic time complexity and DC's sensitivity to edge volume, renders these methods impractical for large-scale applications, despite their promising performance on smaller instances.

4.3.2 Typical Heuristic Algorithms

Simpson et al. [7] conducted an extensive evaluation of multiple heuristic algorithms for the MFAS problem, including Greedy [10], GreedyAbs, Simple [9], BergeShor [22], DFS, KwikSort [23], Sort, and Sift, as well as optimized variants: KwikSort*, Sort*, and Sift*. Originally implemented in Java for unweighted undirected graphs, we reimplemented and extended these algorithms in Python to support edge-weighted directed graphs. Multiple algorithmic and implementation-level optimizations were introduced to enhance runtime performance and output quality.

Among these, Simple, BergeShor, and DFS are distinguished by their structural heuristics, while the remaining algorithms rely primarily on linear arrangement strategies to determine vertex orderings and extract feedback arc sets.

- **Greedy**: Constructs a vertex ordering iteratively by selecting the vertex with the maximum signed difference between out-degree and in-degree. With complexity $\mathcal{O}(m+n)$, it achieved 69.25% in 49 seconds. Our implementation outperforms the benchmark code [6], which requires nearly one hour. Being deterministic, its solutions vary little across runs.
- **GreedyAbs**: A variant of Greedy that considers the absolute difference between out-degree and in-degree for vertex selection. It improves over Greedy with 73.12% in 38 seconds. Like Greedy, it is deterministic, offering minimal gains from repeated runs.
- **Simple**: Generates a random vertex permutation, derives feedforward and feedback arc sets, and selects the larger. With complexity $\mathcal{O}(m+n)$, it runs in 3 seconds and produces scores ranging from 50.17% to over 70%. Multiple runs with different seeds enhance solution quality.
- **BergeShor**: Assigns low-weight edges to the feedback arc set based on local degree comparisons in a single pass. Its linear complexity and execution time of 4 seconds yielded 60.58%. Its deterministic design precludes improvements through repetition.
- **DFS**: Identifies feedback arcs as back edges in a depth-first traversal. If the resulting feedback arc set exceeds 50% of the total weight, the complement is selected. With complexity $\mathcal{O}(m+n)$, it achieved 56.55% in 33 seconds. Varying DFS roots allows modest diversity.
- **KwikSort**: Inspired by QuickSort, partitions vertices based on their edge directions into "smaller," "equal," and "larger" subsets. Recursive partitioning leads to

[6] https://github.com/arie-matsliah/sfas/

$\mathcal{O}(n \log n)$ complexity. The base variant scored 51.04% in 20 seconds. KwikSort*, which runs 200 iterations, reached 54.27% in 2229 seconds.

- **Sort**: Starts with a random vertex order and iteratively repositions each vertex among its predecessors to minimize backward edges. With $\mathcal{O}(n^2)$ complexity, it scored 71.89% in 4284 seconds. Sort*, running 10 iterations, improved this to 76.23% in 21082 seconds.
- **Sift**: An aggressive variant of Sort that re-evaluates vertex positions across all others in each iteration. Although conceptually powerful, its high computational cost rendered it infeasible for our large graph. Sift* attempted multiple runs but similarly failed within a 24-hour window.

4.3.3 Tight-Cut* Algorithm

Hecht et al. [13] proposed the Tight-Cut* algorithm, which aims to remove feedback arcs by strategically breaking isolated cycles, thereby transforming the graph into an acyclic structure. Unlike many MFAS heuristics that rely on vertex ordering, Tight-Cut* employs a fundamentally different approach based on minimum s–t cuts. Specifically, it performs a minimum cut computation for each edge to identify and eliminate cycles, resulting in a theoretical time complexity of $\mathcal{O}(nm^4)$. This high complexity arises from the need to compute flow-based cuts across all m edges and n vertices, compounding the computational burden.

To evaluate its performance, we converted our dataset into the required format and executed the Tight-Cut* algorithm on our experimental platform. Despite running the implementation for over 48 hours, it failed to yield a feasible solution for the neuroscience graph, which consists of 136,648 vertices and over 5.6 million edges. These results confirm that, while Tight-Cut* may be effective for small graphs or specific cases–as noted in the original paper–its high computational cost makes it impractical for large-scale, real-world instances such as the one examined in our study.

4.3.4 RASstar Algorithm

Xiong et al. [14] introduced the RASstar algorithm, which integrates rule-based reductions and recursive ordering to address the MFAS problem efficiently. RASstar first applies a series of graph simplification rules to reduce structural complexity and eliminate redundant patterns. It then employs the GreedyAbs algorithm to generate an initial vertex ordering. Subsequently, the graph is recursively partitioned into two sequential subgraphs, with GreedyAbs applied at each step to further refine the ordering.

This recursive process continues until all subgraphs collapse into single vertices, producing a final topological sort from which the feedback arc set is extracted. Conceptually, RASstar bears similarity to the SE and DC approaches in its divide-and-conquer design. However, unlike those methods, RASstar leverages efficient linear-time partitioning heuristics rather than computationally expensive minimum bisections.

Implemented in C++, we adapted our dataset to RASstar's input format and executed the algorithm without modification. It achieved a solution with 78.77% feedforward weight in just 9 seconds, demonstrating both high quality and efficiency. The method's time complexity is $\mathcal{O}(mn)$, and its near-deterministic nature ensures consistent results across runs, obviating the need for repetition. These traits make RASstar a strong candidate for practical large-scale applications.

4.3.5 Rocket-Crane Algorithm

Our proposed Rocket-Crane algorithm combines high-quality solutions with exceptional runtime performance, particularly when applied to large-scale graphs. The core Rocket sub-algorithm is implemented in JAX, allowing for seamless GPU acceleration and automatic parallelization. Rocket iteratively explores vertex orderings, dynamically refining the solution using a feedforward weight maximization strategy.

The algorithm's complexity is $\mathcal{O}(t(m+n))$, where t denotes the number of iterations. In our experiments, Rocket achieved the highest overall performance, producing a feedforward arc set that retained 80.05% of the total edge weight in just 4.2 seconds. This superior efficiency, coupled with scalability, positions Rocket-Crane as a state-of-the-art method for solving the MFAS problem in large, real-world directed graphs.

A detailed analysis of Rocket-Crane's optimization mechanisms and performance characteristics will be presented in Section 4.4.

4.3.6 Clustering of Algorithms

To provide an intuitive understanding of the trade-offs between execution time and solution quality, we visualize the performance of all evaluated algorithms using a quadrant-based scatter plot (Fig. 4). The plot classifies algorithms into four categories based on their position along two axes: solution quality (feedforward arc set weight) and execution time.

- **Upper Left Quadrant (High Quality, Short Time)**: This region includes algorithms that simultaneously achieve high solution quality and low execution time. Our Rocket-Crane algorithm exemplifies this quadrant, delivering top-tier performance with exceptional efficiency.
- **Upper Right Quadrant (High Quality, Long Time)**: Algorithms here produce high-quality solutions but incur considerable computational cost. Sort* is representative, offering strong results at the expense of long runtimes.
- **Lower Left Quadrant (Low Quality, Short Time)**: These methods yield relatively weak solutions but execute quickly. Simple belongs to this group, making it a viable option for rapid approximations where precision is less critical.
- **Lower Right Quadrant (Low Quality, Long Time)**: The least desirable algorithms fall into this region, characterized by both high runtime and poor solution quality. KwikSort*, despite repeated iterations, shows only limited improvement and thus fits this profile.

This clustering framework clearly illustrates the performance landscape and supports informed algorithm selection for large-scale Minimum Feedback Arc Set (MFAS) problems, where balancing solution quality and computational feasibility is essential.

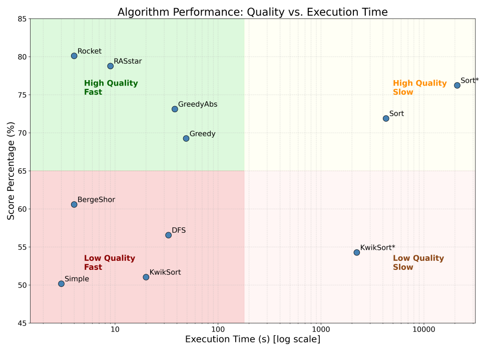

Fig. 4 Clustering of heuristic algorithms based on solution quality and execution time.

4.4 Rapid Convergence to Excellent Solutions with Rocket

The Rocket sub-algorithm can deliver high-quality solutions rapidly and continues to refine them as computation progresses. This iterative behavior allows for a controllable trade-off between speed and solution quality, making Rocket highly adaptable to various runtime constraints.

Fig. 5 presents the progression of Rocket's performance over time on both CPU and GPU platforms. The main plot captures the overall trend, while an inset zooms into the final convergence phase for finer resolution.

Key observations include:

- **Rapid Initial Improvement**: Both CPU (blue curve) and GPU (red curve) variants achieve a steep increase in solution quality within the first few seconds, with scores rising from approximately 2.2×10^7 to over 3×10^7 within the first 10^3 seconds. This indicates that Rocket is capable of producing high-quality approximations early in its execution.
- **GPU Acceleration Advantage**: Throughout the execution, the GPU consistently outperforms the CPU. For example, at 10 seconds, the GPU reaches a score of 34,522,696 (approximately 82.36%), whereas the CPU achieves 29,324,461 (about 69.97%). This demonstrates the effectiveness of JAX-based parallelization in harnessing GPU computing power.
- **Asymptotic Convergence**: As time approaches 10^5 seconds, both curves plateau near a final score of 3.4×10^7, indicating convergence. This suggests diminishing returns in solution quality beyond this point, offering a practical upper bound on useful computation time.
- **Detailed Convergence Behavior**: The inset magnifies the interval from 2×10^4 to 10^5 seconds. The CPU score improves gradually from 3.4724×10^7 to 3.4728×10^7, while the GPU score remains nearly flat between 3.4730×10^7 and 3.4732×10^7, revealing fine-grained convergence dynamics.

This analysis highlights Rocket's strengths in delivering rapid, high-quality solutions and its scalability via GPU acceleration.

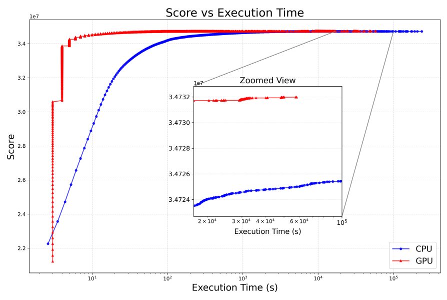

Fig. 5 Solution quality vs. execution time for the Rocket sub-algorithm on CPU (blue) and GPU (red). The inset magnifies the convergence phase.

4.5 Beyond the Excellent: Refinement with Crane

While the Rocket sub-algorithm demonstrates exceptional performance in the early and mid-stage optimization process–achieving a feedforward arc set score of 34,732,073 (approximately 82.87%)–it exhibits a characteristic plateau after several hours of execution. This behavior is typical of gradient-based methods, which quickly ascend to local optima but struggle to escape them without external intervention. Once this saturation point is reached, further runtime produces no gains, signaling the need for a more refined approach.

To address this limitation, the Crane sub-algorithm extends the solution quality beyond Rocket's plateau. It improves the best-found solution to 35,459,266 (or 84.60% in percentage terms), raising the score by 727,193 points (about 1.73%). Although the numerical improvement may appear marginal, it is highly significant in domains where near-optimal solutions are critical. This refinement comes at a cost: Crane requires approximately 20 days of computation, compared to Rocket's few-hour runtime. Nonetheless, in many real-world applications, such costs are acceptable, as even small gains in quality can provide significant value.

The Crane sub-algorithm integrates three complementary optimization strategies–Simulated Annealing (SA), Mixed-Integer Programming (MIP) heuristics, and Monte Carlo sampling–into a cohesive multi-stage procedure. It begins with SA, which is employed to escape local optima by iteratively perturbing the current best solution over $N_{\text{SA}} = 1{,}000{,}000$ iterations, thereby steering the search toward promising regions in the solution space. The solution produced by SA is then passed to an MIP-based refinement module, yielding a feasible solution with higher quality by solving a relaxed integer optimization problem using heuristic techniques within a 40-minute budget. Finally, a Monte Carlo sampling component explores the neighborhood of the MIP-refined solution over $N_{\text{MC}} = 1{,}000{,}000$ iterations, capturing subtle improvements that deterministic methods may overlook.

Our experimental results clearly demonstrate the effectiveness of the proposed complementary strategy. By leveraging Gurobi's warm-start capability, we initialize the MIP solver with a solution generated by Simulated Annealing (SA), allowing it to focus on improving high-quality heuristic solutions rather than solving for the exact optimum. We observe that once MIP identifies an improved solution, additional time rarely leads to further gains. In practice, a time budget of 20–40 minutes is typically sufficient for MIP to enhance the SA output. For example, starting from an SA solution with a score of 35,231,371, MIP improves it to 35,231,397 in under 20 minutes. However, continuing with the same approach beyond this point tends to yield diminishing returns. When SA + MIP reaches a score of 35,231,406, further improvement becomes challenging. In contrast, Monte Carlo (MC) sampling is able to push the solution to 35,435,948, creating a significantly better starting point that enables SA and MIP to rapidly refine it further to 35,448,870.

These experimental results support a key conclusion: switching to a complementary method is often more effective than persisting with the same approach after one method achieves an improvement. This reinforces the advantage of combining multiple strategies in a staged refinement pipeline.

Our experiments also highlight the importance of TopoShuffle initialization in accelerating search efficiency. Starting from a baseline score of 35,142,231 (about 84%), SA alone can improve the score to 35,147,408 in one hour. However, with TopoShuffle initialization, SA achieves 35,162,376, a 0.0426% improvement, which is substantial in the refinement phase, where gains are typically incremental. A similar trend holds for MC: from the same starting point, MC alone reaches 35,147,408, while TopoShuffle+MC reaches 35,162,376, outperforming both pure SA and pure MC.

This iterative process–alternating between SA, MIP-based heuristics, and Monte Carlo search–enables the Crane sub-algorithm to achieve refinements unattainable by any single method alone.

In summary, the Rocket sub-algorithm rapidly drives the solution quality from a baseline (e.g., 50%) to an "excellent" threshold (83%) in minimal time but tends to plateau beyond that. The Crane sub-algorithm takes over from this point, performing advanced refinements and pushing the solution beyond excellent. A hybrid approach–beginning with Rocket and transitioning to Crane–offers a powerful and balanced strategy, combining speed, robustness, and solution quality for solving large-scale Minimum Feedback Arc Set (MFAS) problems.

4.6 Discussion

Key hyperparameters–particularly the sigmoid sharpness parameter β and the number of optimization iterations–significantly influence the performance of our algorithm. To alleviate the need for manual tuning, we adopt a cyclical schedule for β[7], varying its value dynamically throughout optimization.

This adaptive adjustment allows the optimizer to alternate between smoother and sharper approximations of the binary indicator function. As a result, the algorithm gains flexibility to explore the solution space more broadly and refine promising regions, thereby reducing sensitivity to any single, fixed choice of β.

We can easily incorporate an early exit mechanism for iteration control that terminates the current optimization phase if no substantial improvement is observed within a predefined number of iterations or a time threshold. This strategy improves efficiency by avoiding wasted computation in stagnating regions of the search space and accelerates progression to subsequent optimization phases.

[7] $\beta = \left(\cos(\texttt{np.linspace}(0, 2 \cdot \pi \cdot n_{\text{cycles}}, \texttt{num\_epochs})) + 1.1\right)/2$.

4.6.1 Evaluation Based on the Proposed Metrics

To systematically compare algorithmic effectiveness, we evaluate each method according to the three performance metrics introduced earlier: **Time to Good (TTG)**, **Time to Excellent (TTE)**, and **Time to Best (TTB)**. These metrics reflect the time required to reach solution quality thresholds of approximately 75%, 80%, and 85%, respectively. Table 2 summarizes these results.

Among the evaluated algorithms, only four–Sort, Sort*, RAS*, and our proposed Rocket-Crane–are capable of achieving a Good Solution (defined as a relative score of approximately 75%). Their respective execution times are 4,284 seconds, 21,082 seconds, 9 seconds, and 3.6 seconds. Notably, Rocket-Crane reaches this threshold with the shortest execution time by a significant margin.

In terms of achieving an *Excellent Solution* (approximately 80%), Rocket-Crane is the only algorithm able to do so, requiring just 4.2 seconds. This result underscores its rapid convergence and high-quality early-stage performance.

Finally, for the *Best Solution* benchmark (approximately 85%), Rocket-Crane again stands out as the sole algorithm capable of reaching this level of performance. It achieves this score within approximately 20 days, demonstrating its capability toward near-optimal solutions at a computational cost that, while substantial, remains reasonable and acceptable given the complexity of the task.

Table 2 Evaluation of Different Algorithms Based on Performance Metrics

Algorithm	**TTG (75%) (s)**	**TTE (80%) (s)**	**TTB (85%) (days)**
SE	NV	NV	NV
DC	NV	NV	NV
Greedy	NV	NV	NV
GreedyAbs	NV	NV	NV
Simple	NV	NV	NV
BergeShor	NV	NV	NV
DFS	NV	NV	NV
KwikSort	NV	NV	NV
Sort	4284	NV	NV
Sift	NV	NV	NV
KwikSort*	NV	NV	NV
Sort*	21082	NV	NV
Sift*	NV	NV	NV
Tight-Cut*	NV	NV	NV
RASstar	9	NV	NV
Rocket-Crane	3.6	4.2	20

Note: “NV” indicates that the algorithm did not meet the specified performance criterion.

5 Related Work

The Minimum Feedback Arc Set (MFAS) problem has been extensively studied across theoretical, algorithmic, and applied domains. Foundational work by Younger [9] established the linear arrangement formulation for MFAS, which became a cornerstone for many heuristic approaches. Divieti et al. [24] transformed MFAS into the prime implicant selection problem in switching theory, revealing its structural equivalence with classical problems in Boolean logic.

Recent theoretical progress includes extremal bounds and probabilistic analysis. Fox et al. [25] derived tight upper bounds for MFAS size under forbidden subgraph constraints and introduced constructive algorithms with provable guarantees. Diamond et al. [26] analyzed MFAS behavior in random graphs generated via the Erdös-Rényi model, establishing asymptotic lower bounds using binomial-based probabilistic tools. These works contribute to a deeper theoretical understanding but are not directly applicable to large, irregular real-world graphs.

Given the NP-hardness of MFAS, specialized algorithms have been designed for restricted graph classes. For example, Kenyon-Mathieu et al. [27] developed a PTAS specifically for tournament graphs, and Eades et al. [11] proposed a heuristic for cubic graphs that guarantees a feedback arc set of size at most $|E|/4$. Hecht [28] introduced essential minors and isolated cycles to reduce problem complexity and obtain exact solutions for resolvable graphs. Similarly, Chen et al. [29] developed fixed-parameter algorithms that are effective only when the solution size is small. However, these approaches lack scalability or generalizability to dense or heterogeneous graphs, such as those seen in biological networks or software call graphs.

To address these challenges, some researchers developed preprocessing or kernelization techniques. Baweja et al. [30] proposed a semi-streaming PTAS for tournament graphs, while Bessy et al. [31] introduced a linear vertex kernel for the k-Feedback Arc Set problem, significantly reducing the problem size when k is small. On the exact algorithm front, Lempel et al. [32] formulated MFAS as a Boolean algebra problem and solved it using permanent expansions, and Baharev et al. [33] combined lazy constraint generation with integer programming to tackle sparse instances. Despite their theoretical appeal, these exact methods struggle to scale to graphs with tens or hundreds of thousands of nodes.

In practice, heuristic and approximation algorithms are commonly used. Berger et al. [22] proposed randomized algorithms capable of finding large acyclic subgraphs in polynomial time. Even et al. [34] introduced sphere-growing techniques and fractional relaxations, achieving improved theoretical guarantees supported by empirical evidence. Simpson et al. [7] conducted a comprehensive study on multiple approximation algorithms, optimizing greedy and randomized variants for large-scale graphs. Their results show that properly engineered heuristics can scale to massive graphs with billions of arcs while maintaining competitive performance.

More recent innovations leverage bio-inspired and stochastic techniques. Kudelic et al. [35] introduced the Ant-Inspired Monte Carlo Algorithm (AIMCA), which combines adaptive learning mechanisms with Monte Carlo sampling. AIMCA maintains polynomial complexity while offering probabilistic guarantees, particularly effective for multigraphs. Hecht et al. [13] proposed Tight-Cut*, a hybrid approach integrating isolated cycle detection, randomized relaxations, and minimum cut heuristics. Their method achieves near-optimal solutions on sparse graphs, validated by empirical approximation ratios close to 2.

Sorting-based heuristics have also been repurposed for MFAS. Brandenburg et al. [12] reinterpreted classical sorting strategies like insertion sort and Quicksort [23] to construct acyclic subgraphs. Their hybrid approaches demonstrate improved convergence properties by combining deterministic ordering with localized refinements.

Saab [5] developed two divide-and-conquer algorithms–Stochastic Evolution (SE) and Dynamic Clustering (DC)–that iteratively improve feedback arc sets via vertex movement and hierarchical partitioning. These algorithms perform well on small graphs but do not scale effectively to larger datasets due to their high computational complexity.

Xiong et al. [14] introduced a modern framework combining graph reduction techniques with a recursive divide-and-conquer strategy. Their method significantly improves performance on large circuit graphs by reducing redundancy and partitioning graphs for local optimization. This algorithm represents the latest in heuristic design and is included in our empirical evaluation.

Borst [4] recently proposed a novel approach for analyzing neural connectivity by mapping discrete vertex indices to continuous values and minimizing a smooth energy function using gradient-based optimization. Although this continuous-space mapping improves interpretability and reveals hidden structures, it is computationally expensive and applicable only to small networks. In contrast, our work generalizes this idea with improved mapping strategies and scalable optimization, enabling application to large, real-world datasets while preserving high solution quality.

In summary, while many existing methods provide strong theoretical insights or heuristic value, they often fall short in balancing scalability, generality, and quality on real-world graphs. Our proposed Rocket-Crane framework addresses this gap by combining rapid convergence, deep refinement, and GPU-accelerated scalability.

6 Conclusion

We have presented *Rocket-Crane*, a novel two-phase algorithm for solving the NP-hard Minimum Feedback Arc Set (MFAS) problem on large, real-world directed graphs. Our method outperforms state-of-the-art heuristics and exact solvers by delivering high-quality solutions within seconds and continuing to refine them to near-optimality. Specifically, Rocket-Crane can achieve “good” and “excellent” solutions—with approximately 75–80% of the total edge weight retained in the feedforward arc set—in under 5 seconds. Notably, Rocket-Crane is the first algorithm, to our knowledge, that achieves a “best” solution with approximately 85% feedforward arc set weight on large-scale real-world graphs. This milestone, reached within a feasible computation window of 20 days, sets a new benchmark in the field.

The strength of our approach lies in three core innovations. First, the use of a linear-arrangement-based formulation provides a memory-efficient and scalable alternative to traditional cycle-based methods, which often suffer from severe memory bottlenecks. Second, our Rocket sub-algorithm leverages continuous-space gradient-based optimization. By carefully designing a surrogate loss function that aligns well with the MFAS objective, Rocket is able to efficiently explore the solution space using highly efficient numerical optimization techniques. Third, the Crane sub-algorithm extends Rocket’s results through a synergistic integration of randomized search (simulated annealing), MIP-based heuristic, and Monte Carlo exploration. These three components are highly complementary: each explores distinct regions of the solution space, and when combined, they achieve significantly better results than any single method alone.

The effectiveness of Rocket-Crane has been demonstrated on large-scale neuroscience graphs and featured in a recent *Nature* publication [36], highlighting its applicability to real-world scientific domains. Our approach not only achieves state-of-the-art results in terms of performance and scalability but also introduces a new paradigm for solving MFAS by combining continuous optimization, heuristic scheduling, and stochastic refinement.

Our implementation is publicly available on GitHub, supporting reproducibility and future research. We believe Rocket-Crane lays the foundation for future hybrid frameworks in graph optimization, especially for problems where early-stage speed and late-stage quality are both essential.

Acknowledgment

This research was funded in part by NSF grant number CCF-2109988 and OAC-2402560.

References

[1] Lawler, E.: A comment on minimum feedback arc sets. IEEE Transactions on Circuit Theory **11**(2), 296–297 (1964)

[2] Kann, V.: On the approximability of NP-complete optimization problems. PhD thesis, Royal Institute of Technology Stockholm (1992)

[3] Karp, R.M.: Reducibility among combinatorial problems. In: 50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art, pp. 219–241. Springer, Berlin (2009)

[4] Borst, A.: Connectivity matrix seriation via relaxation. PLOS Computational Biology **20**(2), 1011904 (2024)

[5] Saab, Y.: A fast and effective algorithm for the feedback arc set problem. Journal of Heuristics (3), 235–250 (2001) https://doi.org/10.1023/A:1011315014322

[6] Dinur, I., Safra, S.: The importance of being biased. In: Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, pp. 33–42 (2002)

[7] Simpson, M., Srinivasan, V., Thomo, A.: Efficient computation of feedback arc set at web-scale. Proceedings of the VLDB Endowment **10**(3), 133–144 (2016)

[8] Rutenbar, R.A.: Simulated annealing algorithms: An overview. IEEE Circuits and Devices magazine **5**(1), 19–26 (1989)

[9] Younger, D.: Minimum feedback arc sets for a directed graph. IEEE Transactions on Circuit Theory **10**(2), 238–245 (1963)

[10] Eades, P., Lin, X., Smyth, W.F.: A fast and effective heuristic for the feedback arc set problem. Information processing letters **47**(6), 319–323 (1993)

[11] Eades, P., Lin, X.: A heuristic for the feedback arc set problem. Australas. J Comb. **12**, 15–26 (1995)

[12] Brandenburg, F.J., Hanauer, K.: Sorting heuristics for the feedback arc set problem. Department of Informatics and Mathematics, University of Passau (2011)

[13] Hecht, M., Gonciarz, K., Horvát, S.: Tight localizations of feedback sets. Journal of Experimental Algorithmics (JEA) **26**, 1–19 (2021)

[14] Xiong, Z., Zhou, Y., Xiao, M., Khoussainov, B.: Finding small feedback arc sets on large graphs. Computers & Operations Research, 106724 (2024)

[15] Romanycia, M.H., Pelletier, F.J.: What is a heuristic? Computational intelligence **1**(1), 47–58 (1985)

[16] Kokash, N.: An introduction to heuristic algorithms. Department of Informatics and Telecommunications **1**, 1–7 (2005)

[17] Maaroju, N.: Choosing the best heuristic for a NP-Problem. PhD thesis (2009)

[18] Maffioli, F.: Randomized algorithms in combinatorial optimization: A survey. Discrete Applied Mathematics **14**(2), 157–170 (1986)

[19] Buluc, A., Kolda, T.G., Wild, S.M., Anitescu, M., Degennaro, A., Jakeman, J., Kamath, C., Kannan, R., Lopes, M.E., Martinsson, P.-G., et al.: Randomized algorithms for scientific computing (rasc). arXiv preprint arXiv:2104.11079 (2021)

[20] Rubinstein, R.Y., Kroese, D.P.: Simulation and the Monte Carlo Method. John Wiley & Sons, Hoboken, NJ (2016)

[21] Kahn, A.B.: Topological sorting of large networks. Communications of the ACM **5**(11), 558–562 (1962)

[22] Berger, B., Shor, P.W.: Approximation algorithms for the maximum acyclic subgraph problem. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 236–243 (1990)

[23] Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: ranking and clustering. Journal of the ACM (JACM) **55**(5), 1–27 (2008)

[24] Divieti, L., Grasselli, A.: On the determination of minimum feedback arc and vertex sets. IEEE Transactions on Circuit Theory **15**(1), 86–89 (1968)

[25] Fox, J., Himwich, Z., Mani, N.: Extremal results on feedback arc sets in digraphs. Random Structures & Algorithms **64**(2), 287–308 (2024)

[26] Diamond, H., Kon, M., Raphael, L.: Asymptotic lower bounds for the feedback arc set problem in random graphs. arXiv preprint arXiv:2409.16443 (2024)

[27] Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors. In: Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing, pp. 95–103 (2007)

[28] Hecht, M.: Exact localisations of feedback sets. Theory of Computing Systems **62**, 1048–1084 (2018)

[29] Chen, J., Liu, Y., Lu, S., O'sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, pp. 177–186 (2008)

[30] Baweja, A., Jia, J., Woodruff, D.P.: An Efficient Semi-Streaming PTAS for Tournament Feedback ArcSet with Few Passes. arXiv preprint arXiv:2107.07141 (2021)

[31] Bessy, S., Fomin, F.V., Gaspers, S., Paul, C., Perez, A., Saurabh, S., Thomassé, S.: Kernels for feedback arc set in tournaments. Journal of Computer and System Sciences **77**(6), 1071–1078 (2011)

[32] Lempel, A., Cederbaum, I.: Minimum feedback arc and vertex sets of a directed graph. IEEE Transactions on circuit theory **13**(4), 399–403 (1966)

[33] Baharev, A., Schichl, H., Neumaier, A., Achterberg, T.: An exact method for the minimum feedback arc set problem. Journal of Experimental Algorithmics (JEA) **26**, 1–28 (2021)

[34] Even, G., Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multicuts in directed graphs. Algorithmica **20**, 151–174 (1998)

[35] Kudelić, R., Ivković, N.: Ant inspired Monte Carlo algorithm for minimum feedback arc set. Expert systems with applications **122**, 108–117 (2019)

[36] Dorkenwald, S., Matsliah, A., Sterling, A.R., Schlegel, P., Yu, S.-C., McKellar, C.E., Lin, A., Costa, M., Eichler, K., Yin, Y., *et al.*: Neuronal wiring diagram of an adult brain. Nature **634**(8032), 124–138 (2024)