algorithms

MDPI

Article

A Recursive Solution to the Global Maximum Minimum Cut Problem with a Fixed Sink

Xiaoyao Huang [1], Shuo Quan [1] and Jie Wu [1,2,*]

[1] China Telecom Cloud Computing Research Institute, Beijing 100083, China; huangxy32@chinatelecom.cn (X.H.); quansh@chinatelecom.cn (S.Q.)
[2] Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
* Correspondence: jiewu@temple.edu

Abstract

In graph theory and network design, the minimum cut is a fundamental measure of system connectivity and communication capacity. While prior research has largely focused on computing the minimum cut for a fixed source–sink pair, practical scenarios such as data center communication often demand a different objective: identifying the source node whose minimum cut to a designated sink is maximized. This task, which we term the Global Maximum Minimum Cut with Fixed Sink (GMMC-FS) problem, captures the goal of locating a high-capacity source relative to a shared sink node that aggregates multiple servers. The problem is of significant engineering importance, yet it is computationally challenging as it involves a nested max–min optimization. In this paper, we present a recursive reduction (RR) algorithm for solving the GMMC-FS problem. The key idea is to iteratively select pivot nodes, compute their minimum cuts with respect to the sink, and prune dominated candidates whose cut values cannot exceed that of the pivot. By recursively applying this elimination process, RR dramatically reduces the number of max-flow computations required while preserving exact correctness. Compared with classical contraction-based and Gomory–Hu tree approaches that rely on global cut enumeration, the proposed RR framework offers a more direct and scalable mechanism for identifying the source that maximizes the minimum cut to a fixed sink. Its novelty lies in exploiting the structural properties of the sink side of suboptimal cuts, which leads to both theoretical efficiency and empirical robustness across large-scale networks. We provide a rigorous theoretical analysis establishing both correctness and complexity bounds, and we validate the approach through extensive experiments. Results demonstrate that RR consistently achieves optimal solutions while significantly outperforming baseline methods in runtime, particularly on large and dense networks.

Keywords: complexity analysis; data center networks; minimum cut

check for updates

Academic Editor: Massimiliano Caramia

Received: 31 August 2025
Revised: 11 October 2025
Accepted: 14 October 2025
Published: 20 October 2025

Citation: Huang, X.; Quan, S.; Wu, J. A Recursive Solution to the Global Maximum Minimum Cut Problem with a Fixed Sink. *Algorithms* **2025**, *18*, 665. https://doi.org/10.3390/a18100665

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In graph theory and network optimization, the concept of a *minimum cut* plays a fundamental role in quantifying network reliability [1], communication bottlenecks [2], and resource allocation constraints [3]. Given a network represented as a capacitated undirected graph, the classic minimum cut problem focuses on identifying the smallest set of edges whose removal disconnects a designated pair of nodes—typically a source and a sink. This problem has been extensively studied and forms the basis of many important results, including the celebrated Max-Flow Min-Cut Theorem [3].

While traditional algorithms have concentrated on computing the minimum cut between a single source–sink pair [4–7], many emerging applications require a global view of network capacity. In particular, it is often important to identify the source node that can sustain the highest possible minimum cut to a fixed sink [8,9]. This leads to the *Global Maximum Minimum Cut with Fixed Sink (GMMC-FS)* problem: given a fixed sink node in a capacitated graph, find the source node whose minimum s–t cut is the largest among all possible sources. Applications of this problem arise in a variety of real-world network scenarios. For example, consider a communication network in which multiple base stations transmit data to a central controller. Selecting the base station whose connection to the controller has the largest minimum cut ensures the most reliable data delivery path, even under potential link or node failures. This idea naturally extends to other domains. In server provisioning and data center selection [10], the formulation assists in determining which server can sustain the highest aggregate throughput toward a fixed sink, thereby improving overall network utilization. In resilient data dissemination [11], the approach helps identify a source node that maintains service continuity under multiple link disruptions. Finally, in load-aware routing, the model facilitates balanced traffic distribution by selecting sources that provide the strongest connectivity margins to the sink.

The GMMC-FS problem can be naturally expressed as a nested max–min optimization [12], making it both conceptually and computationally more difficult than classical minimum cut computations. A naive approach would require solving a minimum cut problem from every other node to the sink, resulting in an excessive computational cost—especially in large or dense graphs. Furthermore, unlike the global minimum cut [13], the GMMC-FS problem lacks known deterministic polynomial-time algorithms with guaranteed optimality.

In this paper, we propose a **recursive reduction (RR)** algorithm for solving the GMMC-FS problem with provable correctness and practical efficiency. The key idea is to iteratively select a pivot source, compute its minimum cut with respect to the fixed sink, and then prune all vertices that cannot possibly exceed the pivot in cut value. By recursively applying this elimination process, the algorithm progressively reduces the candidate set of sources until only the maximizer remains. This design transforms an otherwise exhaustive enumeration of all source nodes into a streamlined recursive search, often requiring exponentially fewer max-flow computations in practice.

Our main contributions are summarized as follows:

- We formally define the Global Maximum Minimum Cut with Fixed Sink (GMMC-FS) problem, capturing a natural max–min connectivity formulation relevant to diverse applications.
- We design RR, a recursive reduction algorithm that leverages pivot-based elimination to efficiently solve the GMMC-FS problem. RR guarantees correctness while drastically reducing the number of max-flow computations compared to naive enumeration.
- We provide a rigorous theoretical analysis of RR, proving that it always identifies the optimal source and establishing both worst-case and expected complexity bounds.
- We conduct extensive empirical evaluations across diverse graph topologies. Results demonstrate that RR consistently achieves optimal solutions with substantially reduced runtime relative to baseline methods, particularly on large and dense graphs.

The remainder of the paper is organized as follows. Section 2 formally defines the GMMC-FS problem and presents the necessary preliminaries. Section 3 introduces our RRA algorithm. Section 4 provides theoretical analysis, including correctness and complexity bounds. Section 5 presents experimental results with the baselines and varying scenarios. Section 6 introduces the related work and Section 7 concludes the paper.

2. Problem Definition and Preliminaries

We consider a capacitated undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set, with the number of nodes $n = |V|$ and edges $m = |E|$. Each edge $e \in E$ has a non-negative capacity $c_e \geq 0$. Let $t \in V$ be a fixed terminal node, referred to as the sink. For any node $s \in V \setminus \{t\}$, define

$$f(s) \;=\; \min\{\mathrm{cap}(\delta(X)) : s \in X,\; t \notin X\},$$

where for any edge set $F \subseteq E$, $\mathrm{cap}(F) := \sum_{e \in F} c_e$, and for any vertex subset $X \subseteq V$, $\delta(X) := \{\, \{u, v\} \in E : u \in X,\; v \notin X \,\}$. That is, $f(s)$ represents the minimum capacity of any cut that separates node s from the fixed sink t. The objective of the problem is to identify the source node

$$s^\star \;=\; \arg\max_{s \in V \setminus \{t\}} f(s).$$

2.1. Relationship to Classical Min–Cut and Max–Flow

The GMMC-FS problem differs from the classical minimum cut problem, which seeks the smallest cut separating a single pair of nodes. In contrast, GMMC-FS is a max–min problem: it aims to identify the source node that has the highest possible minimum cut to a fixed sink. This makes the problem inherently more complex, as it involves evaluating multiple source candidates rather than a single cut.

By the max–flow min–cut theorem, $f(s)$ is equivalent to the value of the maximum flow from s to t. Therefore, solving the GMMC-FS problem can also be interpreted as identifying the node s that achieves the maximum value of the minimum capacity s–t cut (or equivalently, the maximum $s \to t$ flow).

2.2. Computational Challenge

A naive approach to solving the GMMC-FS problem would involve computing the minimum s–t cut (or maximum flow) for every $s \in V \setminus \{t\}$. Each such computation requires invoking a flow oracle, whose running time we denote by $\mathsf{T}_{\mathrm{flow}}(n, m)$, depending on the algorithm used (e.g., Edmonds–Karp [4], Dinic [5], Push–Relabel [6], or Orlin's [7] algorithm). In dense graphs, where $m = \Theta(n^2)$, such computations for all nodes become very expensive.

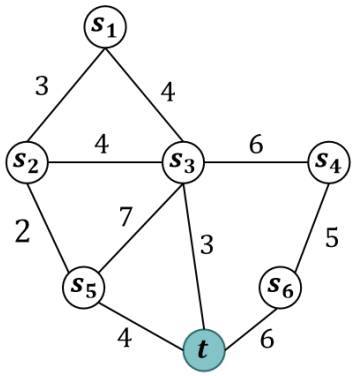
Hence, a more efficient strategy is required to avoid exhaustively evaluating all possible source nodes, particularly for large-scale or dense networks. Our proposed approach addresses this challenge by using recursive reduction with the cut information, which narrows the search space while maintaining correctness guarantees.

2.3. Illustrative Example

To better illustrate the structure and objective of the GMMC-FS problem, we provide a concrete example. Figure 1a shows a small undirected graph consisting of six source nodes $s_1, \ldots, s_6$ and one fixed sink node t. Each edge is labeled with its corresponding capacity, and all capacities are assumed to be non-negative integers. For each source node s_i, we compute the minimum s_i–t cut value, denoted $f(s_i)$. These values represent the smallest total capacity of edges that need to be removed to disconnect s_i from t. The results are summarized directly in the associated Figure 1b.

This example highlights that different source nodes yield different minimum cut values to the sink. For instance, $f(s_3) = 12$ and $f(s_5) = 12$ are the highest, whereas $f(s_1) = 7$ is the smallest. Therefore, the nodes s_3 and s_5 are optimal source nodes in this instance. The example demonstrates two key aspects of the GMMC-FS problem: (i) it illustrates the variability of connectivity strength between sources and the sink, and (ii) it

shows that even on small graphs, computing all possible minimum cuts quickly becomes nontrivial, motivating the need for more efficient, scalable algorithms.



(**a**) A random graph with six sources and one sink.

Source	Min-Cut to t
s_1	7
s_2	9
s_3	12
s_4	11
s_5	12
s_6	11

(**b**) Minimum cut values from each source to the fixed sink.

Figure 1. An illustrative GMMC-FS instance.

3. Recursive Reduction Algorithm

We now present the proposed *recursive reduction (RR)* algorithm, designed to efficiently identify the source node with the largest minimum s–t cut value in a capacitated undirected graph $G = (V, E)$. While a straightforward solution requires computing $|V| - 1$ max-flow problems against a fixed sink t, RR leverages the structural properties of s–t cuts to eliminate groups of nodes at once, substantially reducing the number of required computations.

3.1. Algorithm Overview

The recursive reduction algorithm builds on the simple yet powerful observation that once the minimum cut between a chosen pivot node r and the sink t is computed, all other nodes lying on the same side of the cut as r cannot have a strictly larger cut value than r itself. These nodes can therefore be safely discarded from further consideration. By iteratively selecting pivots, evaluating their minimum cuts, and pruning entire subsets of vertices at each step, the algorithm progressively narrows the candidate set of sources. The search thus shifts from evaluating all possible sources to a recursive elimination process, which substantially reduces the number of max-flow computations required in practice while still guaranteeing that the true maximizer is preserved.

3.2. Algorithm Description

Let $\mathcal{C} \subseteq V \setminus \{t\}$ denote the current candidate set of sources. The algorithm begins with all non-sink vertices and iteratively selects one pivot from $\mathcal{C}$ according to a prescribed strategy, such as choosing the node of highest weighted degree (RR-MaxDeg), selecting randomly (RR-Random), or prioritizing boundary vertices revealed by previous cuts (RR-Boundary). For the chosen pivot, a minimum cut with respect to t is computed, and the resulting cut value $\kappa = f(r)$ is compared with the best solution found so far. If it improves upon the current record, both the maximum value and the corresponding source are updated. Next, the algorithm eliminates from $\mathcal{C}$ all nodes that lie on the same side of the cut as the pivot, since their cut values are upper bounded by the pivot's. The process then continues with the reduced candidate set. Once $\mathcal{C}$ becomes empty, the algorithm returns the source achieving the largest recorded cut value. The full procedure is summarized in Algorithm 1.

Algorithm 1 Recursive reduction (RR) algorithm

Input: An undirected capacitated graph $G = (V, E)$ with a fixed sink t
Output: A source $s^\star$ maximizing the s–t minimum cut value

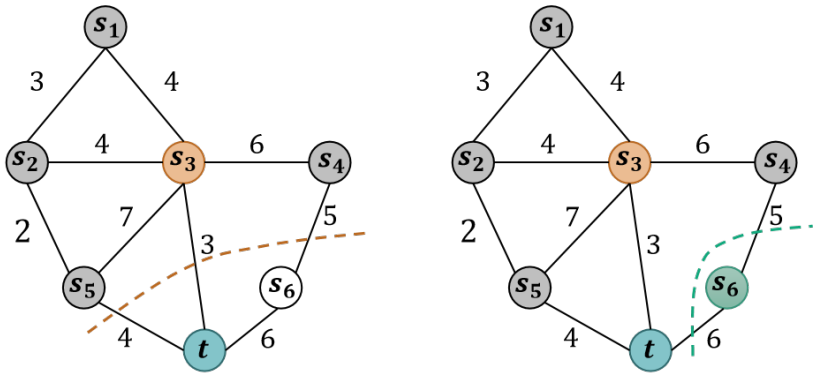
Initialize candidate set $\mathcal{C} \leftarrow V \setminus \{t\}$, best value $best\_val \leftarrow -\infty$, and $s^\star \leftarrow \varnothing$.
while $\mathcal{C} \neq \varnothing$ **do**
  Select a pivot $r \in \mathcal{C}$ according to a chosen strategy (e.g., random, maximum weighted degree, or boundary proximity).
  Compute the $r \to t$ minimum cut (S, T) with $r \in S$, $t \in T$ and cut value $\kappa = f(r)$.
  if $\kappa > best\_val$ **then**
    Update $best\_val \leftarrow \kappa$ and $s^\star \leftarrow r$.
  if $r \in S$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cap (T \setminus \{t\})$
  else
    $\mathcal{C} \leftarrow \mathcal{C} \cap (S \setminus \{t\})$
return $s^\star$

3.3. Illustrative Example

Figure 2, which adopts the same base graph as Figure 1, illustrates the recursive reduction process step by step. In the first iteration, the pivot node is selected as s_3 (colored orange) according to the maximum weighted degree strategy. The corresponding minimum s_3–t cut is shown in orange dashed lines, with cut value 12. All vertices on the same side of the cut as s_3, namely $\{s_1, s_2, s_3, s_4, s_5\}$, are removed from the candidate set since their cut values cannot exceed 12. The remaining candidate set reduces to $\{s_6\}$ (colored green).

In the second iteration, s_6 is the only remaining candidate. Its minimum cut shown in green dashed lines with t has value 11, which is strictly smaller than the current best value 12. Therefore, no update is made to the best solution. As the candidate set is now empty, the recursion terminates and the algorithm returns s_3 as the optimal source, achieving the global maximum minimum cut value of 12.

This example highlights the essence of the recursive reduction mechanism: by successively choosing pivots, evaluating their minimum cuts, and discarding dominated subsets, the candidate set shrinks rapidly while preserving correctness. In practice, this leads to substantial reductions in the number of max-flow computations compared to exhaustive enumeration.



(**a**) Iteration 1: pivot node s_3, *min-cut* value 12; remove $\{s_1, s_2, s_3, s_4, s_5\}$ and retain $\{s_6\}$.

(**b**) Iteration 2: pivot s_6, *min-cut* = 11; no remaining candidates, return optimal source s_3.

Figure 2. Recursive reduction process in RR: the candidate set is *shrunk* at each step.

3.4. Summary and Design Rationale

The recursive reduction (RR) algorithm is designed with two complementary goals: correctness and efficiency. Its correctness lies in the fact that each elimination step preserves

the optimal source, a property that is formally established in Section 4.1. Its efficiency comes from recursive pruning: instead of exhaustively evaluating all $|V| - 1$ possible sources, the algorithm progressively discards large subsets of candidates and quickly narrows the search space. As shown in Section 4.2, this mechanism guarantees that the expected number of iterations is only $\mathcal{O}(\log n)$ on typical random graphs, yielding a total expected running time of $\mathcal{O}(\log n \cdot \mathsf{T}_{\text{flow}}(n, m))$, significantly improving over enumeration. This combination of provable optimality and practical efficiency constitutes the central design rationale of RR, making it a robust and scalable alternative to exhaustive max-flow computations.

4. Theoretical Analysis

We now establish the theoretical foundations of the recursive reduction (RR) algorithm. Our analysis is divided into two parts: first, we prove the correctness of the algorithm, showing that it always identifies a source node attaining the maximum s–t minimum cut value. Second, we analyze its computational complexity, contrasting the worst-case behavior with the typical performance observed in random graph models.

4.1. Correctness Guarantee

We now establish that Algorithm 1 always identifies a source node achieving the maximum s–t cut value. Recall that for any $s \in V \setminus \{t\}$,

$$f(s) \;=\; \min\{\mathrm{cap}(\delta(X)) : \; s \in X,\; t \notin X\}, \qquad s^\star \;=\; \arg\max_{s \in V \setminus \{t\}} f(s).$$

4.1.1. Safe Elimination via Pivot Cuts

At each iteration, the algorithm selects a pivot r and computes a minimum r–t cut (S, T) with value $\kappa = f(r)$. Since every $u \in S$ is separated from t by the same cut, $f(u) \le f(r)$. Thus, if the true maximizer $s^\star$ lies in S, then $f(s^\star) = f(r)$ and the optimum has already been attained; otherwise, $s^\star \in T$ and remains in the candidate set. Hence discarding S is always safe.

4.1.2. Invariant and Termination

This argument yields a simple invariant: after every iteration, either the optimum value has already been recorded, or the true maximizer remains in the candidate set. Since the candidate set shrinks strictly each round, the algorithm halts in at most $n - 1$ iterations.

4.1.3. Main Theorem

Theorem 1 (Correctness of RR)**.** *Upon termination, Algorithm 1 returns a source s with $f(s) = f(s^\star)$.*

Proof sketch. By the invariant, the optimum is never lost. When the candidate set becomes empty, the recorded best value must equal $f(s^\star)$, and the corresponding source is returned. Formal details are deferred to Appendix A. □

4.2. Complexity Analysis

We analyze the iteration complexity of the RR framework. In each round of RR, a pivot vertex r is selected from the current candidate set $C \subseteq V \setminus \{t\}$, and the minimum r–t cut is computed to eliminate vertices. Since every iteration removes at least one candidate, in the worst case, RR halts after $n - 1$ iterations. This $\Theta(n)$ bound is tight but arises only in highly adversarial constructions, and is rarely encountered in practice. We therefore focus on two more structured regimes where substantially faster convergence can be proved: (*i*) graphs where star cuts dominate the minimum cut structure, and (*ii*) graphs where every

minimum cut is balanced. Finally, we present a unified upper bound that combines the two cases.

4.2.1. Star-Cut-Dominated Graphs

We first consider the regime where the minimum r–t cut typically isolates only the pivot vertex. Formally, for a vertex $r \in V$, we call the cut $(\{r\}, V \setminus \{r\})$ a *star cut* . In sparse random graphs, such star cuts dominate the cut structure with high probability.

Assumption 1 (Erdős–Rényi random graph with i.i.d. capacities)**.** *Let $G \sim G(n,p)$ with $p \geq C_0 \frac{\log n}{n}$ for some absolute constant $C_0 > 1$. Each potential edge e independently exists with probability p. Edge capacities $\{c_e\}$ are i.i.d., independent of the topology, with mean $\mu > 0$, variance $\sigma_c^2 > 0$, and finite third absolute moment.*

Theorem 2 (Logarithmic iterations under star-cut dominance)**.** *Under the above assumption, with high probability the minimum r–t cut is a star cut for every pivot r. Consequently, the RR algorithm terminates in $\mathcal{O}(\log n)$ iterations in expectation.*

Proof sketch. For a vertex r, write $\deg_w(r) := \sum_{e \in \delta(r)} c_e$ for the capacity of its star cut $\delta(r)$. Under i.i.d. edge capacities with mean μ and light tails (e.g., bounded or sub-Gaussian), $\deg_w(r)$ concentrates around its expectation $\mathbb{E}[\deg_w(r)]$ with high probability. Moreover, any r–t cut strictly larger than the star cut has a capacity larger by a constant factor with high probability; hence, star cuts dominate. Because pivots are symmetric and chosen uniformly at random from the candidate set, each iteration removes a uniformly random candidate, so the candidate set shrinks geometrically in expectation, which yields an expected $\mathcal{O}(\log n)$ number of iterations. □

4.2.2. Non-Star Balanced-Cut Graphs

We next consider graphs where star cuts are not dominant, but balanced cuts occur with non-negligible probability. Intuitively, as long as each iteration has a constant probability of producing a cut that eliminates a constant fraction of the candidate set, the algorithm still converges in logarithmically many rounds in expectation.

Assumption 2 (Balanced-cut condition)**.** *There exist constants $\alpha, \beta \in (0,1]$ such that for any iteration i and candidate set C_i, the pivot cut eliminates at least an α-fraction of C_i with probability at least β.*

Lemma 1 (Geometric shrinkage in expectation)**.** *Under Assumption 2,*

$$\mathbb{E}[\,|C_{i+1}| \mid C_i\,] \;\leq\; (1-\alpha\beta)\,|C_i|.$$

Proof. If the balanced-cut event occurs (probability $\geq \beta$), then at least $\alpha|C_i|$ candidates are eliminated, yielding

$$|C_{i+1}| \;\leq\; (1-\alpha)|C_i|.$$

Otherwise, fewer may be eliminated. Taking expectation over the two cases proves the stated inequality. □

Theorem 3 (Expected $\mathcal{O}(\log n)$ iterations)**.** *By iterating Lemma 1,*

$$\mathbb{E}[\,|C_t|\,] \;\leq\; (1-\alpha\beta)^t |C_0| \;\leq\; e^{-\alpha\beta t} n.$$

Hence the expected number of iterations until termination satisfies

$$\mathbb{E}[\tau] \;=\; \mathcal{O}\Big(\tfrac{1}{\alpha\beta}\log n\Big).$$

Thus, under the balanced-cut condition, RR terminates in expected $\mathcal{O}(\log n)$ iterations. The overall expected running time is therefore

$$\mathbb{E}[T_{\mathrm{RR}}(n,m)] \;=\; \mathcal{O}\Big(\tfrac{1}{\alpha\beta}\log n \cdot T_{\mathrm{flow}}(n,m)\Big),$$

where $T_{\mathrm{flow}}(n,m)$ denotes the complexity of a maximum flow computation.

4.2.3. Complementary Upper Bound

Theorem 4 (Unified pathwise bound)**.** *Let N_{star} denote the realized number of iterations in which the minimum cut is a star cut, and assume the remaining iterations satisfy the β-balanced condition. Then for every run, the total number of iterations R satisfies*

$$R \;\le\; N_{\mathrm{star}} + \Big\lceil \log_{1/(1-\beta)} n \Big\rceil.$$

Proof sketch. Each star-cut iteration eliminates one candidate, while each β-balanced iteration shrinks the candidate set by a constant fraction. Combining linear progress from N_{star} rounds with geometric progress from the balanced rounds yields the bound. □

In addition, if one introduces a probabilistic model of star-cut occurrence, taking expectations gives

$$\mathbb{E}[R] \;\le\; \mathbb{E}[N_{\mathrm{star}}] \;+\; \frac{1}{\alpha\beta}\ln n.$$

If further each iteration produces a star cut with probability at least $\rho > 0$, then

$$\mathbb{E}[N_{\mathrm{star}}] \;\le\; \frac{n}{\rho},$$

so that

$$\mathbb{E}[R] \;\le\; \frac{n}{\rho} \;+\; \frac{1}{\alpha\beta}\ln n.$$

This shows that frequent star cuts (ρ constant) keep the expected number of iterations logarithmic, while rare star cuts ($\rho \to 0$) lead to near-linear behavior.

4.2.4. Summary

The iteration complexity of RR depends on the structural properties of the underlying graph and detailed proofs are provided in Appendix B. Table 1 summarizes the main regimes: in the worst case adversarial constructions graphs may require $\Theta(n)$ iterations, while star-cut-dominated or β-balanced graphs admit $\mathcal{O}(\log n)$ convergence in expectation. Classical random graph families naturally align with these categories: ER graphs typically fall in the star-cut-dominated regime due to homogeneous degrees and weak clustering; RRG and RGG exhibit balanced partitions thanks to uniform connectivity and symmetry; while WS, BA, and SBM display mixed structures combining hubs, clustering, or community boundaries.

Table 1. Iteration complexity regimes of RR and representative graph families.

Graph Assumption	Iteration Complexity	Representative Families
None	$\Theta(n)$	Adversarial constructions
Star-cut-dominated	$\mathcal{O}(\log n)$	Erdős–Rényi (ER)
β-balanced cuts	$\mathcal{O}(\log n)$	Random regular (RRG), random geometric (RGG)
Mixed structure	$S + \mathcal{O}(\log n)$	Watts–Strogatz (WS), Barabási–Albert (BA), stochastic block models (SBM)

4.2.5. Discussion

The iteration complexity of RR is highly sensitive to the cut structure of the graph. Star-cut dominance and balanced cuts both lead to logarithmic performance, while the absence of such structure results in linear behavior.

A key algorithmic factor is the rule for selecting the pivot vertex r. Three natural strategies illustrate how pivoting interacts with graph structure:

- *Uniform random selection.* In symmetric settings such as Erdős–Rényi graphs, uniformly random pivots exploit star-cut dominance, leading to the expected $\mathcal{O}(\log n)$ bound.
- *Boundary-based selection.* In graphs with balanced cuts, choosing pivots near the cut boundary increases the likelihood of large reductions in the candidate set, consistent with the β-balanced cut guarantee.
- *Maximum-degree selection.* Choosing the vertex with the highest weighted degree leverages structural heterogeneity, particularly in hub-dominated or scale-free graphs. This strategy significantly raises the chance of isolating a t-star cut, which both the theoretical analysis and empirical results show to yield the fastest convergence.

While poor pivot choices can in principle realize the pessimistic $\Theta(n)$ bound, such cases are rare in practice unless pivoting is adversarial. Overall, the pivot selection rule serves as the practical bridge between the structural regimes identified in theory and the observed efficiency of RR across diverse graph families.

5. Experimental Evaluation

5.1. Experimental Setup

We evaluate the proposed recursive reduction (RR) algorithm on synthetic graphs generated from the Erdős–Rényi (ER) random graph model $G(n, p)$, which has been widely used for analyzing algorithmic scalability under controlled structural conditions. In all ER instances we set $p = \bar{d}/(n-1)$, where $\bar{d}$ denotes the target average degree of the random graph and p is the corresponding edge probability (not to be confused with the pivot r). Edge capacities $\{c_e\}$ are sampled i.i.d. as $c_e \sim \mathrm{Unif}\{1, \ldots, 20\}$, independent of the topology. Choosing `avg_deg` $\geq C_0 \log n$ ensures $pn \geq C_0 \log n$, which matches the assumptions in Section 4.2 and yields connected graphs with high probability.

The comparison focuses on four algorithms:

- **RR-Random:** RR with pivot nodes selected uniformly at random.
- **RR-MaxDeg:** RR with pivot nodes chosen as those with the largest weighted degree.
- **RR-Boundary:** RR with pivot nodes chosen as the vertices closest to the most recently discovered cut boundary.
- **Enum:** A baseline that computes the s–t minimum cut for every $s \in V \setminus \{t\}$ using a standard max-flow routine, thereby guaranteeing optimality at the cost of maximal runtime.

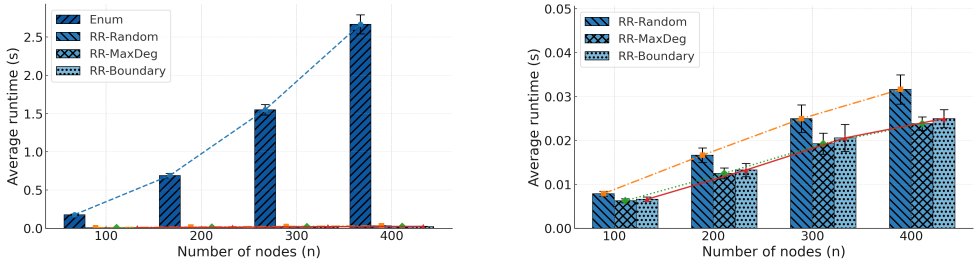
Unless otherwise specified, we vary the number of nodes n, average degree $\bar{d}$, and capacity ranges to test scalability and robustness. Each data point represents the average over multiple independent ER graph realizations, ensuring statistical stability. Runtime is measured in wall-clock seconds. All max-flow computations are implemented using Dinic's algorithm (worst-case complexity $O(V^2E)$).

5.2. Results and Analysis

5.2.1. Scalability with Graph Size

We evaluate scalability by varying the number of nodes $n \in [100, 400]$ while fixing the average degree at $\bar{d} = 99$. The results are summarized in Figure 3 and Table 2. Figure 3a shows that the baseline Enum exhibits rapidly increasing runtime as n grows, reflecting the fact that it requires $\Theta(n)$ independent max-flow computations and each invocation of Dinic's algorithm becomes more expensive with larger graphs. In contrast, all RR variants remain below a few hundredths of a second even at $n = 400$, demonstrating a clear scalability advantage. Figure 3b focuses on the RR-based methods: RR-MaxDeg consistently achieves the lowest runtime, followed closely by RR-Boundary, while RR-Random is somewhat slower but still far superior to Enum. The advantage of degree- and boundary-based pivoting is consistent with the theoretical analysis, which shows that increasing the likelihood of selecting a pivot with $\deg_w(r) \geq \deg_w(t)$ raises the probability of triggering the t-star event and thereby accelerates convergence.

Table 2 reports the average iteration counts required by each RR variant until convergence. All RR variants require only a handful of iterations, well below the $\log n$ benchmark, providing strong empirical evidence for the logarithmic iteration bound established in Section 4.2. Among the strategies, RR-MaxDeg yields the fewest iterations across all tested sizes, RR-Boundary performs second best, and RR-Random incurs the largest number. This ordering parallels the runtime results in Figure 3b and is naturally explained by the star-dominance and two-star approximate symmetry established in our analysis: degree-based pivoting drives the constant probability β close to one, boundary-based pivoting provides a moderate improvement, and random pivoting realizes the baseline constant. Although the average iteration counts fluctuate slightly with n rather than increasing monotonically, the overall runtime nonetheless grows with graph size because the cost of each max-flow call rises with n. These results confirm that the RR framework achieves consistently small iteration counts in practice, with the total runtime dominated by the complexity of the underlying max-flow oracle.



(**a**) Runtime for all algorithms (**b**) Runtime for RR-based algorithms

Figure 3. The performance of the algorithms with varying numbers of nodes.

Table 2. Average number of iterations for RR-based algorithms.

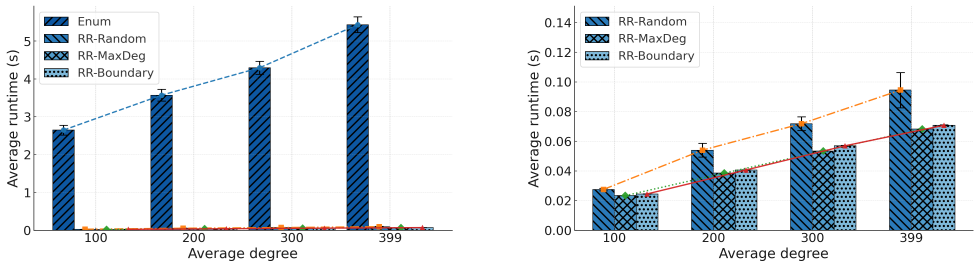
n	$\log n$ (Theory)	RR-Random	RR-MaxDeg	RR-Boundary
100	≈6.64	5.07	2.18	2.66
200	≈7.64	6.46	2.19	2.67
300	≈8.23	7.46	2.79	3.29
400	≈8.64	6.70	1.40	1.89

5.2.2. Sensitivity to Graph Density

The impact of graph density is examined by fixing the number of nodes at $n = 400$ and varying the average degree $\bar{d}$ from 100 to 399, with edge capacities sampled i.i.d. as $c_e \sim \text{Unif}\{1, \ldots, 20\}$. Figure 4a reports the average runtime of all algorithms. The Enumeration baseline shows runtime growing nearly linearly with $\bar{d}$, consistent with the quadratic scaling of edge counts in dense Erdős–Rényi graphs and the repeated invocations of Dinic's algorithm. By contrast, the RR-based approaches remain one to two orders of magnitude faster across the entire density spectrum, and their runtime curves increase only modestly with $\bar{d}$.

The differences among RR variants are highlighted in Figure 4b. RR-MaxDeg consistently achieves the lowest runtime across all densities, benefiting from pivot selections that favor highly connected nodes and thereby accelerate candidate elimination. RR-Boundary follows closely, with slightly higher runtimes that nonetheless remain well below those of RR-Random. The latter exhibits the steepest runtime growth among the three as $\bar{d}$ increases, reflecting the weaker structural guidance of random pivoting in dense topologies. Despite this relative disadvantage, even RR-Random substantially outperforms Enumeration by more than an order of magnitude.

These observations indicate that recursive pruning remains effective in the high-density regime. Structural pivot rules such as MaxDeg and Boundary are particularly advantageous, as they exploit connectivity patterns that are amplified with higher degrees, enabling the algorithm to maintain small iteration counts and controlled runtime growth.



(**a**) Runtime for all algorithms (**b**) Runtime for RR-based algorithms

Figure 4. The performance of the algorithms with varying graph density.

5.2.3. Effect of Edge Capacity Distribution

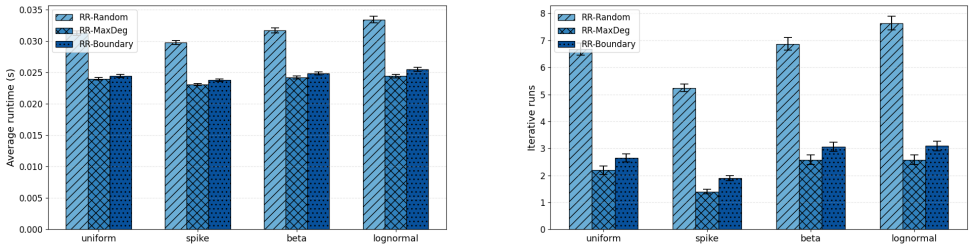
We next evaluate the impact of edge capacity distributions while fixing $n = 200$ and $\bar{d} = 100$. Four representative families are considered: (i) Uniform$\{1, \ldots, 20\}$, serving as a baseline with moderate variance; (ii) Spike, where a small fraction of edges are assigned very large capacities while the majority remain near the minimum, producing extreme variance; (iii) Beta, biased toward small capacities and emphasizing weaker links; and (iv) Lognormal, rescaled to $\{1, \ldots, 20\}$, introducing a heavy right tail and pronounced heterogeneity.

Figure 5a reports runtimes for the three RR variants. Across all distributions, runtimes remain under 0.04 seconds, confirming that efficiency is robust to capacity heterogeneity. Nevertheless, heavier-tailed distributions such as Spike and Lognormal incur slightly

higher costs, reflecting more irregular cut structures that reduce pruning efficiency. Uniform and Beta distributions exhibit lower runtimes, consistent with their more balanced edge weights.

Figure 5b shows the corresponding iteration counts. The results parallel the runtime trends: RR-Random consistently requires the largest number of iterations, particularly under Lognormal and Spike where the probability of immediate elimination is reduced, while RR-MaxDeg and RR-Boundary achieve substantially smaller values across all cases. In every distribution, iteration counts remain within small constant factors of the logarithmic baseline, providing empirical support that the star-dominance and two-star approximate symmetry mechanisms are insensitive to the exact capacity distribution.

Overall, these findings demonstrate that RR maintains both correctness and near-logarithmic efficiency across a wide spectrum of capacity distributions. While extreme heterogeneity modestly increases runtime variance, pivot strategies that exploit degree or boundary information mitigate this effect and deliver consistently faster convergence than random pivoting.



(**a**) Runtime for RR-based algorithms (**b**) Iterations for RR-based algorithms

Figure 5. The performance of the algorithms across different weight distributions.

5.2.4. Robustness to Structural Variability

To evaluate robustness across network structures, we fix the configuration at $n = 200$ nodes, average degree $\bar{d} = 100$, and edge capacities sampled i.i.d. as $c_e \sim \text{Unif}\{1, \ldots, 20\}$. Under this setting, one graph instance is generated from each of six classical random families: Erdős–Rényi (ER), Watts–Strogatz small-world graphs (WSs), Barabási–Albert preferential attachment graphs (BAs), random regular graphs (RRGs), random geometric graphs (RGGs), and stochastic block models (SBMs). These models capture a wide spectrum of structural features, ranging from homogeneous connectivity (ER, RRG) to strong clustering (WS, RGG), heavy-tailed degree distributions (BA), and explicit community organization (SBM).

Table 3 reports the average number of iterations of RR-based algorithms across the six graph families. The results confirm that the logarithmic iteration bound remains a reliable predictor across structurally diverse networks, although the precise iteration counts vary with topology. For ER and RRG, which closely match the assumptions of our theoretical model, the observed values of RR-Random are close to or slightly below the $\log n$ benchmark, while RR-MaxDeg and RR-Boundary achieve substantially fewer iterations. For WS and RGG, where local clustering and geometric constraints reduce uniform expansion, iteration counts remain near the theoretical guideline but show moderate deviations, reflecting weaker star-dominance. The BA model demonstrates the opposite effect: the presence of high-degree hubs strongly accelerates convergence, so all RR variants terminate in nearly constant iterations, far below $\log n$. In contrast, SBM exhibits the slowest convergence: explicit community bottlenecks degrade star-dominance, reducing the probability of the t-star event and leading to iteration counts moderately exceeding the $\log n$ benchmark.

Across all families, RR-MaxDeg consistently yields the fewest iterations, confirming the benefit of exploiting degree heterogeneity. These variations align with the theoretical

characterization and discussion in Section 4.2: the logarithmic expectation is robust under homogeneous random models, while heavy-tailed or community-structured graphs shift the effective constant β, producing either faster convergence (BA) or slower convergence (SBM).

Table 3. Average number of iterations of RR-based algorithms across different graph families.

Graph Family	$\log n$ (Theory)	RR-Random	RR-MaxDeg	RR-Boundary
ER	≈7.64	6.46	2.19	2.67
WS	≈7.64	6.54	2.98	3.50
BA	≈7.64	1.25	1.00	1.09
RRG	≈7.64	6.32	1.99	2.50
RGG	≈7.64	5.96	1.99	2.48
SBM	≈7.64	8.28	4.96	5.50

6. Related Work

The study of network connectivity optimization has its roots in the classical maximum–flowand minimum cut problems. Foundational deterministic algorithms such as Edmonds–Karp [4] and Dinic [5], together with later improvements like Push–Relabel [6] and Orlin's algorithm [7], provide efficient polynomial-time solutions for computing a single source–sink min-cut. However, applying these algorithms to all possible source nodes relative to a fixed sink, as required in the GMMC-FS problem, incurs $\Theta(n)$ flow computations. For dense graphs with $|E| = \Theta(|V|^2)$, this approach quickly becomes computationally infeasible.

Randomized contraction algorithms, pioneered by Karger [13] and refined in the Karger–Stein algorithm [8], shifted attention to the global minimum cut problem. These methods achieve near-optimal runtimes in unweighted graphs with high probability and have been extended through sparsification [14], dynamic contraction thresholds [15], and parallel implementations [16]. While highly successful for global connectivity, contraction-based methods do not directly address the fixed-sink max–min formulation, where the objective is to maximize the minimum cut over all sources to a designated sink.

For multi-terminal and all-pairs cut problems, the Gomory–Hu tree [2] offers a compact representation of all pairwise min-cuts using only $O(n)$ flow computations. Subsequent work has developed faster variants [9,17], as well as extensions to hypergraphs [18] and dynamic settings [19]. Despite their generality, Gomory–Hu-based approaches still require $\Theta(n)$ flow computations, which are unnecessarily expensive when only cuts relative to a fixed sink are needed, as in GMMC-FS.

Beyond theory, practical algorithm engineering has produced solvers optimized for large-scale graphs. For example, VieCut [20] integrates inexact preprocessing heuristics with exact cut routines to achieve significant speedups in practice. In distributed and parallel computation models, particularly the Congested Clique and MPC settings, researchers have developed near-optimal min-cut algorithms with sublinear round complexity [15,19], and streaming algorithms have been proposed for processing massive graphs under memory constraints [21,22]. These paradigms demonstrate strong scalability for general min-cut computations, but their overhead remains substantial for dense networks and fixed-sink formulations.

In networking applications, min-cut-based metrics have been widely adopted in traffic engineering [11], resilient server placement, and fault-tolerant multicast design. The max–min connectivity objective studied in robust network design [12,23] is conceptually related to GMMC-FS, as both focus on maximizing bottleneck connectivity to a sink or terminal. Existing approaches, however, either perform a sequence of full max-flow compu-

tations or rely on linear programming relaxations, both of which struggle to scale in high-density topologies.

In contrast to these prior approaches, the GMMC-FS problem—identifying the source node with the largest minimum cut to a fixed sink—has remained underexplored. Existing deterministic and randomized algorithms either solve the problem indirectly via exhaustive enumeration or rely on structures designed for global or all-pairs settings. Our work fills this gap by introducing the recursive reduction (RR) algorithm, which leverages pivot-based elimination and recursive pruning to reduce the candidate set aggressively. Rather than computing all $|V| - 1$ flows, RR guarantees correctness while requiring only $O(\log n)$ iterations in expectation on typical random graphs, as established in Sections 4.1 and 4.2. This combination of provable optimality and practical scalability distinguishes RR from contraction-based min-cut algorithms and Gomory–Hu tree methods, offering the first exact and efficient solution tailored to the fixed-sink max–min cut problem.

7. Conclusions

This paper addressed the Global Maximum Minimum Cut with Fixed Sink (GMMC–FS) problem, which seeks the source node with the largest minimum cut to a designated sink in a capacitated undirected graph. To solve this problem efficiently, we proposed the recursive reduction (RR) algorithm, which applies a pivot-based elimination strategy to recursively prune dominated candidates while ensuring exact correctness. Specifically, we provided a rigorous theoretical analysis establishing both correctness and logarithmic expected complexity, and demonstrated through extensive experiments on synthetic and real-world graphs that RR consistently achieves the optimal solution with substantially lower runtime than classical enumeration. The results confirm that recursive pruning is highly effective in reducing the search space and that RR scales robustly across graph sizes, densities, and structures.

The proposed RR framework establishes a foundation for efficient max–min source selection in undirected static networks. A promising direction for future work is to extend RR to more general settings, particularly to directed or weighted dynamic graphs, where edge capacities or directions evolve over time. These extensions would enable the framework to handle asymmetric communication networks and time-varying topologies. Another promising direction is to integrate RR with approximation or learning-based strategies to further reduce computational costs on large-scale systems. Such developments could broaden the applicability of RR in dynamic and data-driven network optimization scenarios.

Author Contributions: Conceptualization, X.H., J.W. and S.Q.; methodology, X.H., J.W. and S.Q.; software, X.H.; validation, X.H., J.W. and S.Q.; formal analysis, X.H.; investigation, X.H.; resources, X.H.; writing—original draft preparation, X.H.; writing—review and editing, X.H., J.W. and S.Q.; visualization, X.H.; supervision, J.W.; project administration, J.W.; funding acquisition, J.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Proofs for Correctness Guarantee

We provide detailed proofs for the results stated in Section 4.1. Throughout, for $s \in V \setminus \{t\}$,

$$f(s) \;=\; \min\{\mathrm{cap}(\delta(X)) : s \in X,\, t \notin X\}, \qquad s^\star \;=\; \arg\max_{s \in V\setminus\{t\}} f(s).$$

Appendix A.1. A Universal Upper Bound

Lemma A1 (Feasible-cut upper bound). *For any vertex set $X \subseteq V$ with $t \notin X$ and any $u \in X$,*

$$f(u) \;\le\; \mathrm{cap}(\delta(X)).$$

Proof. By definition, $f(u)$ is the minimum capacity of all u–t cuts. Since $\delta(X)$ separates u from t, it constitutes a feasible u–t cut with capacity $\mathrm{cap}(\delta(X))$. Hence $f(u) \le \mathrm{cap}(\delta(X))$. □

Appendix A.2. Safe Elimination via the Pivot Cut

Fix an iteration of RR. Let the current candidate set be C, let $r \in C$ be the chosen pivot, and let (S, T) be a minimum r–t cut with

$$r \in S, \quad t \in T, \quad \text{and value } \kappa \;=\; f(r) \;=\; \mathrm{cap}(\delta(S)).$$

Corollary A1 (Domination on the pivot side). *For every $u \in S$, one has $f(u) \le f(r) = \kappa$.*

Proof. Apply Lemma A1 with $X = S$ and $u \in S$. □

Lemma A2 (If the optimum lies in S, the pivot is optimal). *If $s^\star \in S$, then $f(r) = f(s^\star)$.*

Proof. By Corollary A1, $f(s^\star) \le f(r)$. By the optimality of $s^\star$, $f(r) \le f(s^\star)$. Hence the equality holds. □

Consequently, the RR elimination rule is safe: the algorithm discards $S \setminus \{t\}$ and keeps $C \leftarrow C \cap (T \setminus \{t\})$. If $s^\star \in S$, then by Lemma A2 the pivot already attains $f(s^\star)$; otherwise $s^\star \in T$ and remains in the candidate set.

Appendix A.3. Invariant and Termination

Let $(C_i,\ best\_val_i)$ denote, respectively, the candidate set and the best recorded value after the i-th iteration.

Proposition A1 (Safety invariant). *For all $i \ge 0$, one of the following holds:*

$$(A)\ \ best\_val_i = f(s^\star), \qquad or \qquad (B)\ \ s^\star \in C_i \ \ and \ \ best\_val_i < f(s^\star).$$

Proof. Initially, (B) holds. Suppose (B) holds at iteration i. If $s^\star \in S$, then by Lemma A2 the newly computed value equals $f(s^\star)$, so *best_val* is updated to $f(s^\star)$ and (A) holds. If $s^\star \in T$, then $s^\star \in C_{i+1}$ while *best_val* remains below $f(s^\star)$, so (B) persists. Once (A) holds, it continues to hold since *best_val* is nondecreasing across iterations. □

Lemma A3 (Termination). *At each iteration, $|C_{i+1}| < |C_i|$. Hence RR halts in at most $n - 1$ iterations.*

Proof. By the elimination rule, $C_{i+1} \subseteq T \setminus \{t\}$. Since $r \in S$ and $r \in C_i$, we have $r \notin C_{i+1}$, so $|C_{i+1}| < |C_i|$. Thus RR terminates after at most $n - 1$ iterations. □

Appendix A.4. Main Theorem

Theorem A1 (Correctness of RR)**.** *Upon termination, Algorithm 1 returns a source s with $f(s) = f(s^\star)$.*

Proof. By Lemma A3, RR stops with $C_\tau = \emptyset$ for some $\tau \le n-1$. By Proposition A1, case (B) cannot hold at this point; thus (A) must hold, i.e., $best\_val_\tau = f(s^\star)$. The algorithm returns the source stored when *best_val* was last updated, which by construction has value $f(s^\star)$. □

Appendix B. Proofs for Complexity Analysis

We provide detailed proofs for the results stated in Section 4.2. *Notation.* For a vertex v, write $\deg_w(v) := \sum_{e\in\delta(v)} c_e$ for the (weighted) capacity of the star cut at v, and $\deg(v) := |\delta(v)|$ for its (unweighted) degree when we need to count incident edges (e.g., inside expectations or concentration bounds).

Appendix B.1. Worst-Case Tightness

Proposition A2. *For arbitrary graphs, the RR algorithm requires at most $n-1$ iterations and this bound is tight.*

Proof. *Upper bound.* In each iteration, at least one candidate vertex is removed from C. Thus the algorithm halts after at most $n-1$ rounds.

Tightness. For every n, construct a graph G with vertex set $\{t, v_1, \ldots, v_{n-1}\}$. Connect each v_i to t with an edge of capacity 1. Between the vertices $\{v_1, \ldots, v_{n-1}\}$, add edges of capacity $M \gg n$. For each pivot $r = v_i$, the minimum r–t cut is the star $\{r\}$ with $\deg_w(r) = 1$. Thus each iteration eliminates only one vertex, requiring exactly $n-1$ iterations. □

Appendix B.2. Star-Cut-Dominated Graphs

Theorem A2 (Restatement of Theorem 2)**.** *Let $G \sim G(n,p)$ with $p \ge C_0 \frac{\log n}{n}$ for some $C_0 > 1$. Suppose edge capacities are i.i.d. with mean $\mu > 0$, variance $\sigma_c^2 > 0$, and finite third absolute moment. Then with high probability, for all $r \in V \setminus \{t\}$, the minimum r–t cut is the star cut $\delta(r)$. As a consequence, under* uniform random pivoting*, the RR algorithm terminates in $\mathcal{O}(\log n)$ iterations in expectation.*

Proof. We divide the proof into three parts.

Appendix B.2.1. (i) Degree Concentration

For each vertex r, the (unweighted) degree $\deg(r) \sim \mathrm{Bin}(n-1, p)$. By Chernoff bounds, for any $\varepsilon \in (0,1)$,

$$\Pr\big[\,|\deg(r) - (n-1)p| \ge \varepsilon(n-1)p\,\big] \le 2\exp\Big(-\tfrac{\varepsilon^2}{3}(n-1)p\Big).$$

Since $p \ge C_0\frac{\log n}{n}$ with $C_0 > 1$, we have $(n-1)p = \Omega(\log n)$. A union bound over all r yields, with high probability,

$$\deg(r) = (1 \pm o(1))\, np \qquad \text{for all } r.$$

Appendix B.2.2. (ii) Capacity Concentration for Star Cuts

For each r, the star-cut capacity is

$$X_r \;=\; \deg_w(r) \;=\; \sum_{e\in\delta(r)} c_e,$$

a sum of $\deg(r)$ i.i.d. random variables with mean μ and variance σ_c^2. Conditioning on $\deg(r)$ and applying Bernstein's inequality (finite third absolute moment), for any $\varepsilon > 0$,

$$\Pr\big[\,\big|X_r - \mu\,\deg(r)\big| \ge \varepsilon\,\deg(r)\;\big|\;\deg(r)\big] \;\le\; 2\exp\big(-\Theta(\varepsilon^2\deg(r))\big).$$

Using part (i) to lower bound $\deg(r) = \Omega(\log n)$ and unconditioning, we obtain

$$X_r \;=\; (1\pm o(1))\,\mu\,np \qquad \text{for all } r$$

with probability at least $1 - n^{-\Omega(1)}$ by a union bound over r.

Appendix B.2.3. (iii) Non-Star Cuts Are Strictly Larger

Consider any cut $(S, \bar{S})$ with $r \in S$, $t \in \bar{S}$, and $|S| \ge 2$. Its expected capacity is

$$\mathbb{E}[c(S,\bar{S})] \;=\; \mu\cdot|S|(n-|S|)p \;\ge\; 2\mu(n-2)p.$$

By part (ii), $X_r = \deg_w(r) = (1\pm o(1))\,\mu np$ with high probability. Thus, for sufficiently large n,

$$\mathbb{E}[c(S,\bar{S})] \;\ge\; (2-o(1))\,\mu np \;>\; (1+o(1))\,\mu np,$$

so the mean of any non-star r–t cut exceeds the typical (concentrated) value of the star cut. Applying Bernstein's inequality to $c(S,\bar{S})$ and taking a union bound over all subsets S (with $|S| \ge 2$) yields that there exists a constant $\gamma > 0$ such that, with high probability,

$$c(S,\bar{S}) \;\ge\; (1+\gamma)\,X_r \qquad \text{uniformly for all such } S.$$

Appendix B.2.4. Conclusions

Combining (ii) and (iii), with probability $1 - n^{-\Omega(1)}$, every non-star cut has capacity strictly larger than the corresponding star cut, hence the minimum r–t cut is exactly $\delta(r)$ for all r. Under *uniform random pivoting*, symmetry implies each iteration removes a uniformly random candidate, so $|C|$ shrinks geometrically in expectation, and the expected number of iterations is $\mathcal{O}(\log n)$. □

Appendix B.3. Balanced-Cut Graphs

Theorem A3 (Restatement of Theorem 3)**.** *Suppose that for every pivot r, the minimum r–t cut $(S,\bar{S})$ satisfies*

$$\min\{|S\cap C|, |\bar{S}\cap C|\} \;\ge\; \beta\,|C|$$

for some constant $\beta \in (0, 1/2]$. Then the RR algorithm halts in $\mathcal{O}(\log n)$ iterations.

Proof. Let $|C_0| = n$ be the initial candidate set size. If $|C_k|$ is the candidate set size after k iterations, then

$$|C_{k+1}| \;\le\; (1-\beta)\,|C_k|.$$

By induction, $|C_k| \le (1-\beta)^k n$. Choosing

$$k \;=\; \Big\lceil \frac{\log n}{\log \frac{1}{1-\beta}} \Big\rceil$$

ensures $|C_k| < 1$, so the algorithm terminates. Hence the number of iterations is $\mathcal{O}(\log n)$. □

Lemma A4 (Restatement of Lemma 1)**.** *If each minimum cut removes at least $k \geq 2$ candidates, then RR halts in at most $(n-1)/k$ iterations.*

Proof. In each iteration, $|C|$ decreases by at least k. Initially $|C| = n$. After m iterations, $|C| \leq n - mk$. Thus $|C| = 0$ after at most $m \leq (n-1)/k$ iterations. □

Appendix B.4. Complementary Bounds

Theorem A4 (Restatement of unified pathwise bound)**.** *Let N_{star} be the number of iterations in which the minimum cut is a star cut, and assume the remaining iterations satisfy the β-balanced condition. Then for every run, the total number of iterations R satisfies*

$$R \;\leq\; N_{\text{star}} + \left\lceil \log_{1/(1-\beta)} n \right\rceil.$$

Proof. Each star-cut iteration decreases $|C|$ by 1, while each β-balanced cut iteration reduces $|C|$ by at least a β fraction. Combining the two effects yields the stated bound. □

Theorem A5 (Unified expected bound with star-cut probability)**.** *Suppose further that each iteration produces a star cut with probability at least $\rho > 0$, independently of past history. Then the expected number of iterations satisfies*

$$\mathbb{E}[R] \;\leq\; \frac{n}{\rho} \;+\; \frac{1}{\alpha\beta} \ln n,$$

where $\alpha\beta$ reflects the contraction factor guaranteed by β-balanced cuts.

Proof. Taking expectations in the pathwise bound gives

$$\mathbb{E}[R] \;\leq\; \mathbb{E}[N_{\text{star}}] \;+\; \frac{1}{\alpha\beta} \ln n.$$

Since at most n candidates can be removed by star cuts, and each such removal requires, in expectation, at most $1/\rho$ iterations, it follows that $\mathbb{E}[N_{\text{star}}] \leq n/\rho$. Substituting completes the proof. □

References

1. Kleinberg, J.; Tardos, E. *Algorithm Design*; Pearson Education: Boston, MA, USA, 2006.
2. Gomory, R.E.; Hu, T.C. Multi-terminal network flows. *J. Soc. Ind. Appl. Math.* **1961**, *9*, 551–570. [CrossRef]
3. Ford, L.R.; Fulkerson, D.R. Maximal flow through a network. *Can. J. Math.* **1956**, *8*, 399–404. [CrossRef]
4. Edmonds, J.; Karp, R.M. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM (JACM)* **1972**, *19*, 248–264. [CrossRef]
5. Dinic, E.A. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Sov. Math. Dokl.* **1970**, *11*, 1277–1280.
6. Goldberg, A.V.; Tarjan, R.E. A new approach to the maximum-flow problem. *J. ACM (JACM)* **1988**, *35*, 921–940. [CrossRef]
7. Orlin, J.B. Max flows in O (nm) time, or better. In Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, Palo Alto, CA, USA, 1–4 June 2013; pp. 765–774.
8. Karger, D.R.; Stein, C. A new approach to the minimum cut problem. *J. ACM (JACM)* **1996**, *43*, 601–640. [CrossRef]
9. Hao, J.; Orlin, J.B. A faster algorithm for finding the minimum cut in a graph. *J. Algorithms* **1994**, *17*, 424–446. [CrossRef]
10. Singla, A.; Hong, C.Y.; Popa, L.; Godfrey, P.B. Jellyfish: Networking data centers, randomly. In Proceedings of the USENIX NSDI, San Jose, CA, USA, 25–27 April 2012; pp. 225–238.
11. Al-Fares, M.; Loukissas, A.; Vahdat, A. Scalable flow-based networking with DiffServ. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 343–352.

12. Chekuri, C.; Khanna, S.; Shepherd, B. Approximation algorithms for node-weighted buy-at-bulk network design. *SIAM J. Comput.* **2006**, *35*, 995–1016.
13. Karger, D.R. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, USA, 25–27 January 1993; pp. 21–30.
14. Benczúr, A.; Karger, D.R. Approximating s-t minimum cuts in $O(n^2)$ time. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 47–55.
15. Ghaffari, M.; Nowicki, K.; Thorup, M. Near-optimal minimum cut algorithms in the distributed, streaming, and MPC models. In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, Chicago, IL, USA, 22–26 June 2020; pp. 1451–1464.
16. Geissmann, B.; Gianinazzi, L. Minimum cuts and network reliability in the congested clique model. In Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, Vienna, Austria, 16–18 July 2018; pp. 327–336.
17. Moitra, A. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, Atlanta, GA, USA, 25–27 October 2009; pp. 3–12.
18. Chekuri, C.; Xu, C. Minimum cuts and sparsification in hypergraphs. In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, Barcelona, Spain, 16–19 January 2017; pp. 1537–1554.
19. Ghaffari, M.; Nowicki, K. Fully dynamic connectivity in $O(\log n(\log \log n)^2)$ amortized update time. In Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, USA, 7–10 January 2018; pp. 118–132.
20. Noe, A.; Hagerup, T.; Dementiev, R. Practical minimum cut algorithms. In Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX), Virtual, 10–11 January 2021; pp. 84–96.
21. Kale, S.; Muthukrishnan, S.; Vassilvitskii, S. The geometry of graph streaming: Cut sketches and linear algebra. In Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, New York, NY, USA, 4–6 January 2009; pp. 123–134.
22. Ahn, K.J.; Guha, S.; McGregor, A. Graph sketches: Sparsification, spanners, and subgraphs. In Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Scottsdale, AZ, USA, 21–23 May 2012; pp. 5–14.
23. Gupta, A.; Newman, I.; Rabinovich, Y.; Sinclair, A. Cuts, trees and l_1-embeddings. In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Las Vegas, NV, USA, 14–17 October 2001; pp. 399–408.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.