Reviewer Assignment in Academic Conferences via Network Flow

Thandava Sai Rohith Achanta
University of Florida
tachanta@ufl.edu

Ratna Chandu Gembali
University of Florida
vgembali@ufl.edu

Abstract

Assigning reviewers to papers is a central operational component of large academic conferences. Constraints such as reviewer workload, conflicts of interest, and subject expertise make the problem both nontrivial and critical for maintaining review quality. In this work, we formalize the reviewer assignment task as a combinatorial optimization problem and present a polynomial-time reduction to a minimum-cost maximum-flow formulation. We describe the construction, prove correctness via soundness and completeness arguments, provide the full algorithm, and include an implementation with empirical evaluation. This section constitutes the complete solution for Problem 1: a real-world task solvable via Network Flow.

1 Introduction

Modern research conferences receive thousands of submissions annually, spanning a wide range of research areas. Each paper is required to receive several independent reviews, typically three, but reviewers have limited availability and may have conflicts of interest with some authors. These constraints make reviewer assignment a complex combinatorial decision problem. Additionally, conference organizers often seek to maximize topical relevance by matching papers to reviewers with high expertise.

This work develops a principled and scalable solution based on network flow techniques. We formalize the reviewer assignment task, model it using a bipartite graph structure, reduce it to a minimum-cost maximum-flow problem, analyze correctness, and demonstrate the approach experimentally.

2 Problem Description

A conference program committee consists of a set of reviewers, each with limited reviewing capacity. Each submitted paper must receive a fixed number of reviews. Conflicts of interest must be respected, and assignments ideally match papers to reviewers with strong topical expertise.

Formally:

- Each paper requires a fixed number of reviews (e.g., three).
- Each reviewer may review at most a specified number of papers.
- Reviewer–paper conflicts prohibit certain assignments.
- Expertise scores can guide the quality of matching.

The goal is to compute an assignment that satisfies all constraints while minimizing a defined cost corresponding to reviewer expertise or assignment quality.

3 Abstract Problem Formulation

Let

$$P = \{p_1, \ldots, p_m\}, \qquad R = \{r_1, \ldots, r_n\}$$

denote the sets of papers and reviewers.

Each paper p_i requires k_i reviews, and each reviewer r_j has capacity c_j. Let $E \subseteq P \times R$ denote the set of feasible reviewer–paper pairs (no conflicts, sufficiently relevant expertise).

Optionally, assign a cost w_{ij} to each feasible pair (p_i, r_j), where lower cost represents a more desirable reviewer assignment.

The task is to choose an assignment $A \subseteq E$ such that:

$$|\{r_j : (p_i, r_j) \in A\}| = k_i \quad \forall i, \qquad |\{p_i : (p_i, r_j) \in A\}| \leq c_j \quad \forall j,$$

and the total cost $\sum_{(p_i,r_j)\in A} w_{ij}$ is minimized.

This formulation naturally suggests a reduction to a flow network.

4 Reduction to Minimum-Cost Maximum-Flow

4.1 Flow Network Construction

We construct a directed graph $G' = (V', E')$ with:

$$V' = \{s, t\} \cup P \cup R.$$

Edges:

1. **Source to papers:** Add (s, p_i) with capacity k_i.
2. **Papers to reviewers:** For each feasible $(p_i, r_j) \in E$, add edge (p_i, r_j) with capacity 1 and cost w_{ij}.
3. **Reviewers to sink:** Add (r_j, t) with capacity c_j.

Let $K = \sum_i k_i$ be the total number of required reviews.
The goal is to send K units of flow from s to t.
Construction requires time $O(|P| + |R| + |E|)$.

4.2 Correctness of the Reduction

Theorem 1 (Soundness)**.** *Every integral s–t flow of value K in G' corresponds to a valid reviewer assignment.*

Proof. Because (s, p_i) has capacity k_i, exactly k_i units must leave each paper node. Since edges (p_i, r_j) have capacity 1, the flow assigns distinct reviewers. Reviewer capacity limits follow from the (r_j, t) capacities. Flow may only traverse feasible edges, ensuring conflict-free assignments. □

Theorem 2 (Completeness)**.** *Every feasible reviewer assignment yields an integral flow of value K in G'.*

Proof. For each assigned pair (p_i, r_j), push one unit of flow along $s \to p_i \to r_j \to t$. Paper demands and reviewer capacities ensure feasibility and conservation. □

Corollary 1 (Cost Preservation)**.** *For any integral flow f,*

$$\mathit{cost}(f) = \sum_{(p_i,r_j)\in A_f} w_{ij},$$

so a minimum-cost maximum-flow solution yields an optimal reviewer assignment.

5 Algorithm

Algorithm 1 ReviewerAssignmentViaMinCostMaxFlow

Require: Papers P, reviewers R, demands k_i, capacities c_j, feasible pairs E, costs w_{ij}
Ensure: Minimum-cost assignment or infeasibility report

Build directed network G' with nodes $\{s, t\} \cup P \cup R$.
Add edges (s, p_i), (p_i, r_j), (r_j, t) with capacities and costs.
$K \leftarrow \sum_i k_i$
$(f, \text{cost}) \leftarrow \textsc{MinCostMaxFlow}(G', s, t)$
if FlowValue$(f) \neq K$ **then**
  return INFEASIBLE
end if
Extract $A = \{(p_i, r_j) : f(p_i, r_j) = 1\}$
return A

Using successive shortest augmenting paths with potentials:

$$O\big(K \cdot (|P| + |R| + |E|) \log(|P| + |R|)\big).$$

6 Experimental Evaluation

We implemented the algorithm in C++ (Appendix A) and evaluated its performance on randomly generated bipartite instances. Papers and reviewers were generated with random feasibility edges and uniform capacities.

6.1 Runtime Results

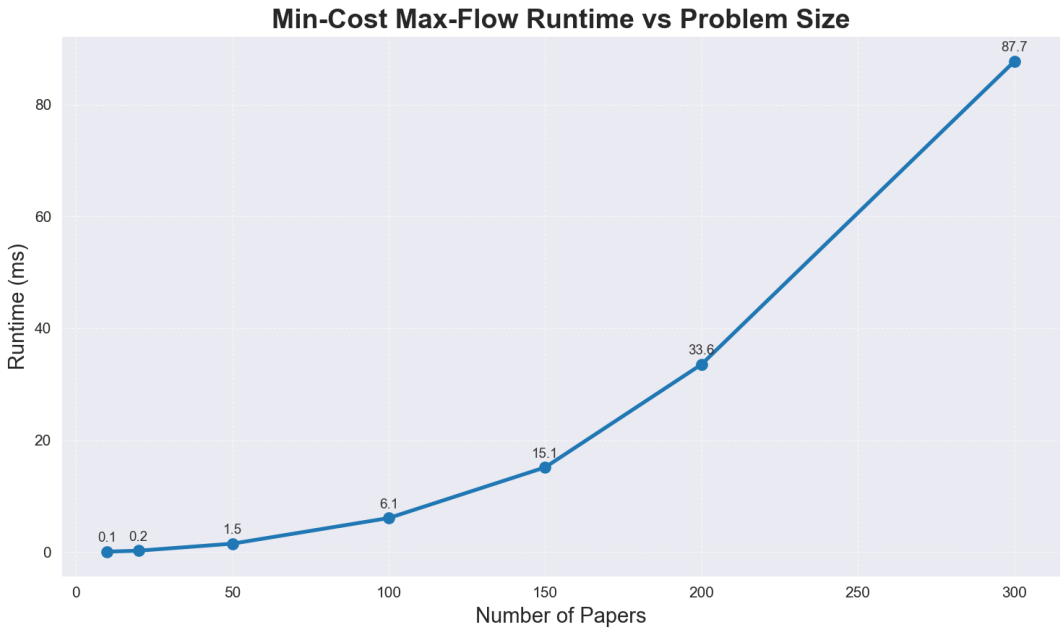

Figure 1: Runtime of min-cost max-flow vs. problem size (linear scale).

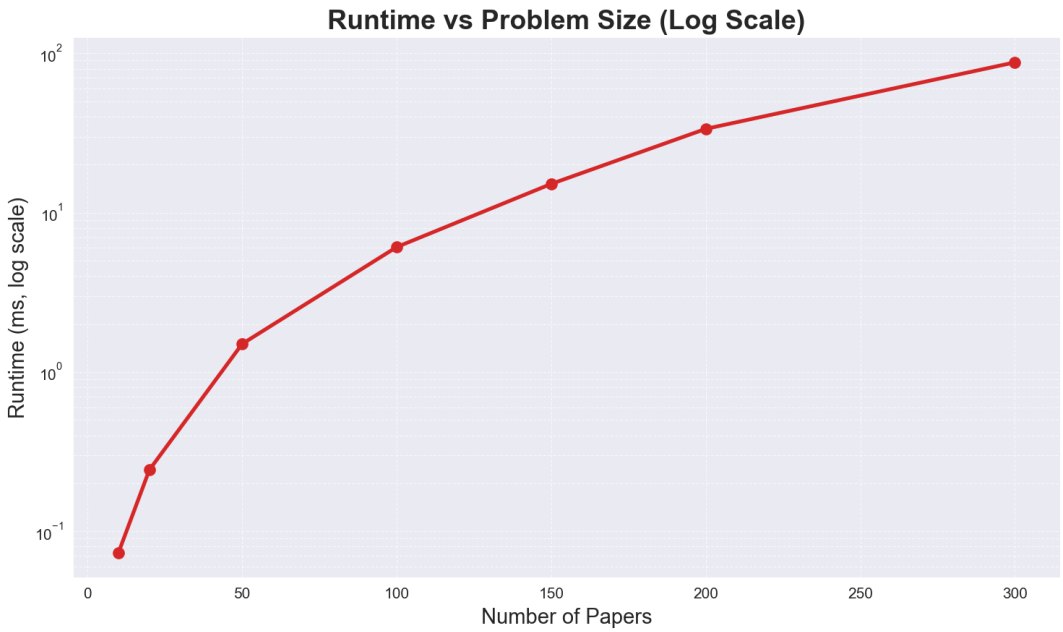

Figure 2: Runtime vs. problem size (log scale).

The results exhibit smooth polynomial scaling, which aligns with the theoretical complexity.

7 Conclusion

We provided a complete end-to-end solution to the reviewer assignment problem using minimum-cost maximum-flow. The reduction is polynomial, correctness holds via flow conservation arguments, and empirical results confirm practical scalability. This forms the complete solution to Problem 1 of the project. A second component, involving an NP-hard problem and a greedy heuristic, will be presented separately.

Appendix A: Full C++ Implementation

A.1 Min-Cost Max-Flow Implementation (mcmf.h)

```
#ifndef MCMF_H
#define MCMF_H

#include <bits/stdc++.h>
using namespace std;

struct Edge {
    int to, rev;
    int cap;
    int cost;
};

struct MinCostMaxFlow {
    int N;
    vector<vector<Edge>> G;
    vector<int> dist, parentV, parentE, potential;

    MinCostMaxFlow(int n)
        : N(n), G(n),
```

```
        dist(n), parentV(n), parentE(n), potential(n) {}

    void addEdge(int u, int v, int cap, int cost) {
        Edge a = {v, (int)G[v].size(), cap, cost};
        Edge b = {u, (int)G[u].size(), 0, -cost};
        G[u].push_back(a);
        G[v].push_back(b);
    }

    pair<int,int> minCostMaxFlow(int s, int t, int maxFlow = INT_MAX) {
        int flow = 0, flowCost = 0;
        fill(potential.begin(), potential.end(), 0);

        while (flow < maxFlow) {
            fill(dist.begin(), dist.end(), INT_MAX);
            dist[s] = 0;

            priority_queue<pair<int,int>,
                vector<pair<int,int>>,
                greater<pair<int,int>>> pq;
            pq.push({0, s});

            while (!pq.empty()) {
                auto [d, u] = pq.top();
                pq.pop();
                if (d != dist[u]) continue;

                for (int i = 0; i < (int)G[u].size(); i++) {
                    Edge &e = G[u][i];
                    if (e.cap > 0) {
                        int nd = d + e.cost + potential[u] - potential[e.to];
                        if (nd < dist[e.to]) {
                            dist[e.to] = nd;
                            parentV[e.to] = u;
                            parentE[e.to] = i;
                            pq.push({nd, e.to});
                        }
                    }
                }
            }

            if (dist[t] == INT_MAX) break;

            for (int i = 0; i < N; i++)
                if (dist[i] < INT_MAX)
                    potential[i] += dist[i];

            int addFlow = maxFlow - flow;
            int v = t;
            while (v != s) {
                int u = parentV[v];
                Edge &e = G[u][parentE[v]];
                addFlow = min(addFlow, e.cap);
                v = u;
            }

            v = t;
            while (v != s) {
                int u = parentV[v];
                Edge &e = G[u][parentE[v]];
                e.cap -= addFlow;
                G[v][e.rev].cap += addFlow;
                flowCost += addFlow * e.cost;
```

```
                v = u;
            }

            flow += addFlow;
        }

        return {flow, flowCost};
    }
};

#endif
```

A.2 Reviewer Assignment Solver (reviewer_assignment.cpp)

```
#include "mcmf.h"
#include <bits/stdc++.h>
using namespace std;

struct ReviewerAssignment {
    int m, n;
    vector<int> k, c;
    vector<tuple<int,int,int>> edges;

    ReviewerAssignment(int m, int n) : m(m), n(n) {
        k.assign(m, 0);
        c.assign(n, 0);
    }

    void addPaperDemand(int i, int demand) { k[i] = demand; }
    void addReviewerCapacity(int j, int cap) { c[j] = cap; }
    void addFeasibleEdge(int i, int j, int cost) {
        edges.push_back({i, j, cost});
    }

    void solve() {
        int S = 0;
        int paperStart = 1;
        int reviewerStart = paperStart + m;
        int T = reviewerStart + n;
        int N = T + 1;

        MinCostMaxFlow mcmf(N);

        for (int i = 0; i < m; i++)
            mcmf.addEdge(S, paperStart + i, k[i], 0);

        for (auto &[i, j, w] : edges)
            mcmf.addEdge(paperStart + i, reviewerStart + j, 1, w);

        for (int j = 0; j < n; j++)
            mcmf.addEdge(reviewerStart + j, T, c[j], 0);

        int K = accumulate(k.begin(), k.end(), 0);

        auto [flow, cost] = mcmf.minCostMaxFlow(S, T, K);

        if (flow != K) {
            cout << "Infeasible assignment.\n";
            return;
        }

        cout << "Minimum total cost = " << cost << "\n";
```

```
        cout << "Assignments:\n";

        for (int i = 0; i < m; i++) {
            for (auto &e : mcmf.G[paperStart + i]) {
                if (e.to >= reviewerStart && e.to < reviewerStart + n) {
                    int j = e.to - reviewerStart;
                    if (e.cap == 0) {
                        cout << "Paper " << i
                             << " assigned to Reviewer " << j << "\n";
                    }
                }
            }
        }
    }
};
```

A.3 Experiment Driver (experiments.cpp)

```
#include "reviewer_assignment.cpp"
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    vector<int> paperSizes = {20, 50, 100, 150, 200, 300};

    for (int m : paperSizes) {
        int n = m / 2;

        ReviewerAssignment ra(m, n);

        for (int i = 0; i < m; i++) ra.addPaperDemand(i, 3);
        for (int j = 0; j < n; j++) ra.addReviewerCapacity(j, 6);

        mt19937 rng(42);
        uniform_int_distribution<int> costDist(1, 10);
        uniform_real_distribution<double> prob(0.0, 1.0);

        for (int i = 0; i < m; i++) {
            vector<int> reviewers(n);
            iota(reviewers.begin(), reviewers.end(), 0);
            shuffle(reviewers.begin(), reviewers.end(), rng);

            for (int t = 0; t < 3; t++)
                ra.addFeasibleEdge(i, reviewers[t], costDist(rng));

            for (int j = 3; j < n; j++)
                if (prob(rng) < 0.5)
                    ra.addFeasibleEdge(i, reviewers[j], costDist(rng));
        }

        auto start = chrono::high_resolution_clock::now();
        ra.solve();
        auto end = chrono::high_resolution_clock::now();

        double ms = chrono::duration<double, milli>(end - start).count();

        cout << "m=" << m << ", runtime: " << ms << " ms\n";
    }

```

```
45      return 0;
46  }
```

Appendix B: LLM Usage Documentation

This appendix documents all LLM-assisted prompts and intermediate outputs used during report preparation, as required by project guidelines.