Flow Decomposition into Paths and Cycles

Tan Tran (tat5593)
Farzad Azarmi (fka5196)

1 Problem Overview

We are given a directed graph $G = (V, E)$ with a designated source vertex $s \in V$ and sink vertex $t \in V$, along with an *integral feasible s–t flow* $f : E \to \mathbb{Z}_{\geq 0}$. We wish to decompose this flow into a collection of s–t paths and directed cycles.

Formally, the output consists of:

- a set of s–t paths P,
- a set of directed cycles C,
- a weight function w defined on $P \cup C$,

such that for every edge $e \in E$,

$$\sum_{p \in P : e \in p} w(p) + \sum_{c \in C : e \in c} w(c) = f(e).$$

The goal is to minimize the sum $|P| + |C|$. Since the project focuses on heuristic solutions, exact optimality is not required; instead, the emphasis will be on running time, correctness, and practical performance.

2 Algorithm

Our algorithm is a greedy, two-phase heuristic that iteratively decomposes the given flow:

1. **Path decomposition phase:** repeatedly extract s–t paths with positive residual flow.
2. **Cycle decomposition phase:** decompose any remaining flow into directed cycles.

At each step, flow is subtracted from edges as soon as it is assigned to a path or cycle. The algorithm terminates when all residual flows become zero.

3 Graph Representation

An adjacency list representation of the graph is kept. Every vertex u maintains a list of outgoing edges (u, v) each with an associated mutable flow value. This allows in-place updates while doing the decomposition. Vertices are numbered from 1 to $|V|$. $s = 1$ and $t = |V|$ are kept as required by the input format.

All flows are integral and no capacity constraints are necessary since the flow is given as input.

4 Path Decomposition Phase

To extract s–t pathways, we run a BFS from s into the graph with positive residual edges. If t is reachable, we recreate the path with parent pointers.

The weight of the path is determined by the minimum remaining capacity on its edges (the bottleneck). This weight is deducted from all edges on the path and added to all reverse edges. The path is then applied to the solution. The operation is continued until no s–t paths remain.

This greedy method assures that each iteration removes at least one positive-flow edge while preserving flow feasibility.

5 Cycle Decomposition Phase

After removing all s-t pathways, the remaining flow can only take directed cycles. We begin by attempting to find a path from v to u using the BFS on positive edges for each edge $\langle u, v\rangle$ that has positive residual capabilities.

If that's the case, the cycle is directed. The cycle weight is the minimum residual flow on all edges of the cycle, including *(u,v)*. It is deducted from each edge of the cycle, which is then indicated.

The cycles are explicitly printed as closed walks, with the first vertex printed at the end, as specified by the output format.

6 Correctness Argument

We briefly justify correctness:

- Each extracted path or cycle follows directed edges with positive residual flow, and thus represents a valid flow component.
- Flow conservation is preserved, since every subtraction from an edge is accounted for in exactly one path or cycle weight.
- Each iteration strictly reduces the total remaining flow, and since all flows are integral, the algorithm must terminate.

Therefore, the resulting sets (P, C, w) form a valid representation of the input flow.

7 Complexity Analysis

Let $V = |V|$ and $E = |E|$. Each BFS runs in $O(E)$ time. Since each iteration reduces at least one edge flow to zero, the total number of iterations is bounded by $O(E)$.

Thus, the overall time complexity is $O(E^2)$, and the memory usage is $O(V + E)$, well within the project limits.

8 Design Choices

The algorithm does not guarantee the minimal value of $|P| + |C|$. Greedy path extraction can increase the number of paths in some instances. On the other hand, this method is quick, simple to implement, and reliable across a wide range of inputs.

In fact, the algorithm generates compact decompositions and performs well under the provided scoring conditions due to its ability to handle cyclic structures effectively.

9 Conclusion

We presented a straightforward and efficient heuristic for dividing an integral s–t flow into pathways and cycles. The algorithm is valid, terminates quickly, and adheres to all of the project's restrictions. While optimality is not guaranteed, a good balance of solution quality and computing efficiency has been achieved.