Project Checkpoint 2: Summary and Breakdown

Sara Baloch, 08586, Kulsoom Asim 08051

April 8, 2025

Published in: 2020 Springer Nature Switzerland AG
DOI: 10.1007/978-3-030-48966-3_13
GitHub Repository: https://github.com/kul2112311/ADAproject.git

1 Problem and Contribution Summary

The paper tackles the NP-hard problem of finding a minimal set of vertices to uniquely distinguish all simple paths between a source and destination in a graph. Its key contribution is polynomial-time algorithms for chordal graphs and tournaments (predictable flow connections), overcoming complex computational problems through structural graph analysis. The work demonstrates how domain-specific constraints enable efficient solutions to generally hard problems, with direct applications in network security (intrusion detection) and transportation logistics (route verification). While implementations face $O(mn^3)$ complexity for dense graphs, the theoretical framework provides a foundation for practical approximations in real-world systems.

2 Algorithm Description

2.1 Chordal Graphs

For these types of graphs, the algorithm exploits the chordal property (no induced cycles ≥ 4) by placing trackers at critical vertices where paths diverge, specifically focusing on common neighbors of edges in cycles.

Inputs

- A chordal graph $G = (V, E)$ (preprocessed with **Reduction Rule 1** to remove vertices/edges not on any s-t path).
- Terminal vertices $s, t \in V$.

Output

- A tracking set $T \subseteq V$ that uniquely distinguishes all s-t paths.

The steps mentioned in the paper

1. Start with an empty tracking set $T = \emptyset$.
2. For each edge $e = (a, b) \in E$:
   - Identify common neighbors $x \in N(a) \cap N(b)$ (potential trackers).
   - If removing x leaves an s-t path containing e, add x to T (ensures x is necessary to distinguish paths).
3. **Return** the final set T.

Notes: since cycles are "short" (triangles or have chords), so trackers need only be placed at vertices shared by multiple paths. Also, a common neighbor x of edge (a, b) can merge two paths into one if unmarked; adding x to T forces path uniqueness.

The algorithm runs in $O(m \cdot n^3)$ time, where m is the number of edges and n is the number of vertices. This is due to checking vertex-disjoint paths for each edge and its common neighbors, leveraging the chordal structure for efficiency.

2.2 Tournament Graphs

Inputs

- A tournament graph $G = (V, E)$ (preprocessed with **Reduction Rule 1** to remove vertices/edges not on any s-t path).
- Terminal vertices $s, t \in V$

Output

- A tracking set $T \subseteq V$ that uniquely distinguishes all s-t paths.

The steps mentioned in the paper

1. Start with an empty tracking set: $T = \emptyset$.
2. For each directed edge $e = (a, b) \in E$:
   - Compute the set $N^+(a) \cap N^-(b)$, where:
     - $N^+(a)$ is the set of out-neighbors of a.
     - $N^-(b)$ is the set of in-neighbors of b.

     Note: these candidate vertices are potential trackers because they serve as detour points that could affect the uniqueness of s-t paths.
   - For each candidate vertex x (that is not already in T):
     - In the graph $G - x$, determine whether there exists an s-t path that still includes the edge e.

– If yes: Add x to T. This ensures that the absence of x would allow two s-t paths to have identical tracker sequences, so including x is essential.

Notes: the algorithm identifies key "detour" vertices (from $N^+(a) \cap N^-(b)$) that could merge distinct paths. Removing a candidate and still finding an s-t path using the same edge indicates that the vertex is essential for differentiation. Including such vertices ensures every s-t path is uniquely tracked.

Overall Complexity: The resulting running time is approximately $O(m \cdot n^3)$, which is polynomial in the size of the tournament graph.

Comparison with Previous Or Existing Algorithms

Earlier algorithms introduced by Banik et al. and Eppstein et al. focused on tracking paths in planar graphs. Banik introduced a 2-approximation algorithm, but only for shortest s–t paths, while Eppstein expanded this to all s–t paths, though with a less precise 4-approximation. Both approaches were vertex-based and limited to planar graphs, meaning they couldn't be applied to more general or complex graph structures.

This paper goes beyond these limitations by introducing exact polynomial-time algorithms for broader graph classes, such as chordal and tournament graphs, which were previously unexplored for path tracking. It also shifts the focus from just vertices to edges. Edge-based tracking proves to be more efficient and is solved here using a reduction to the minimum feedback edge set problem, which can be done in $O(n^2)$ time. The paper also addresses bounded-degree graphs, proving NP-hardness when the degree $\delta \geq 6$, but still provides a $2(\delta+1)$-approximation in $O(n^2)$, which is a significant improvement over earlier approximations like Bilò et al.'s $\widetilde{O}(\sqrt{n})$.

Mathematical Tools

The paper utilizes several key mathematical techniques to address the tracking problem:

1. **Graph Decomposition:** Complex graphs, like chordal graphs, are broken down into simpler structures using perfect elimination orderings and clique trees. This helps in identifying optimal tracker placements by simplifying the graph structure.

2. **Cycle Analysis:** Feedback vertex and edge sets are used to monitor cycles in the graph. Since every cycle requires at least one tracker, these sets ensure proper path distinction. A $2(\delta+1)$ approximation is used for bounded-degree graphs, where δ represents the maximum vertex degree.

3. **NP-Hardness Reductions:** Proves problem difficulty by transforming known hard problems (like Vertex Cover) into tracking problems, showing they're equally challenging to solve

Implementation Challenges

Implementing the algorithm presents challenges, particularly in managing large, complex graph structures like chordal and tournament graphs. Edge-based tracking adds complexity, especially when computing the minimum feedback edge set efficiently for large graphs. Handling bounded-degree graphs with $\delta \geq 6$ is challenging due to NP-hardness, requiring approximations for larger instances. Additionally, managing data structures like adjacency lists and traversal flags while ensuring scalability adds to the implementation complexity.