Dynamic Directed Debt Graph for Efficient Settlement

Course Code: CSL2020

Course Name: Data Structures and Algorithms

Instructor: Prof. Dr. Suchetna Chakraborty
Mentor TA: Tammy Lalhmingthangi Ralte

Team Members:
Aashcharya Gorakh(B23ES1001)
Sparsh Gupta(B23CI1037)
Nikhil Thawani(B23CI1027)
Prayag Raghunandan(B23MT1031)

Github Link - https://github.com/Aashcharya1/Dynamic-Directed-Debt-Graph-for-Efficient-Settlement/tree/main

PROBLEM STATEMENT

Overview:
In social and financial networks, individuals often lend and borrow money, leading to complex chains or cycles of debt.

Objective:
This project aims to design a directed loan graph to track debts dynamically, and apply greedy algorithm to compute the most efficient way to settle debts, minimizing the number and value of transactions. We also want to calculate Most Influential user by using one of the shortest path algorithms. Expected Outcomes are minimized debt chains, optimal settlement paths, reduced number of transactions to clear debts.

Challenges:
Tracking and simplifying multiple debts among users. Identifying *key individuals* influencing the debt network. Comparing various shortest path algorithms to find the best one for our use case.

CURRENT STATUS AND RESEARCH

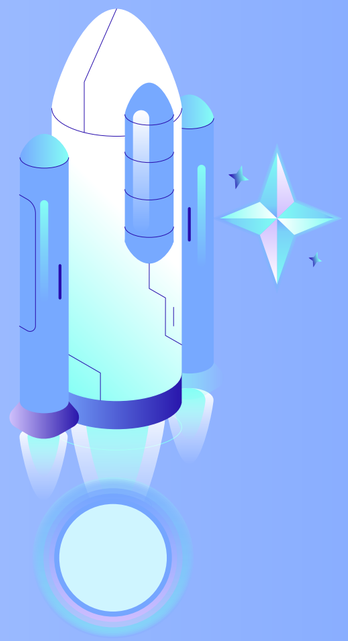
Existing Solutions:
Manual tracking or basic ledger systems.
Limited automation in debt simplification.

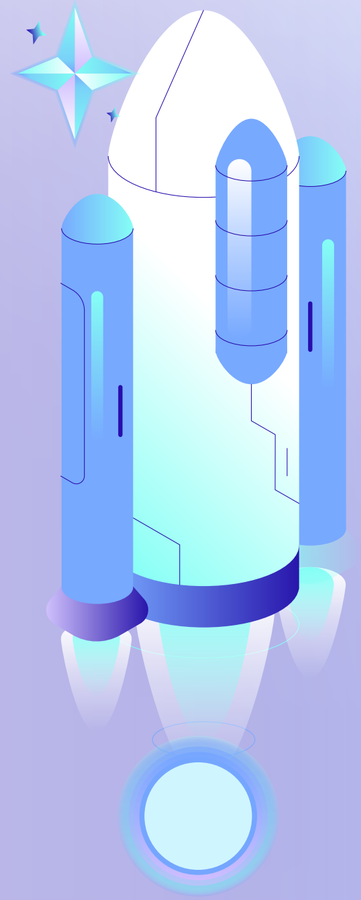
Research Insights:
Studies on network optimization and debt settlement algorithms.
Application of graph theory in financial networks.

Limitations-
Manual Systems:
Prone to errors and inefficiencies.

Basic Algorithms:
Lack of scalability for large networks.
Ineffective in identifying key influencers in the network.

PROPOSED IDEA





Debt Simplification Algorithm using Greedy Approach

1. Calculate Net Balances: For each user, compute how much they owe or are owed by summing all transactions.
2. Classify Users: Split users into debtors (net negative) and creditors (net positive).
3. Simplify Transactions: Match debtors with creditors to settle debts using the smallest possible transactions, reducing overall complexity.
4. Update Graph: Clear old debts and store only the simplified transactions in the graph.

Finding the most Influential Person in the Debt Graph by best Shortest path algorithm -

Compare various algos and choose best one to compute distance of all other users from each user and finding the user with minimum average path.

COMPARING VARIOUS SHORTEST PATH ALGORITHMS

We compared 4 algorithms namely -

1)Dijkstra's algorithm

2) Bellman Ford

3)Floyd Warshal

4) A*

Chronic Framework was used as a comparison metric

Chronic Documentation - https://cplusplus.com/reference/chrono/

Results were plotted using a Python Script

RESULTS

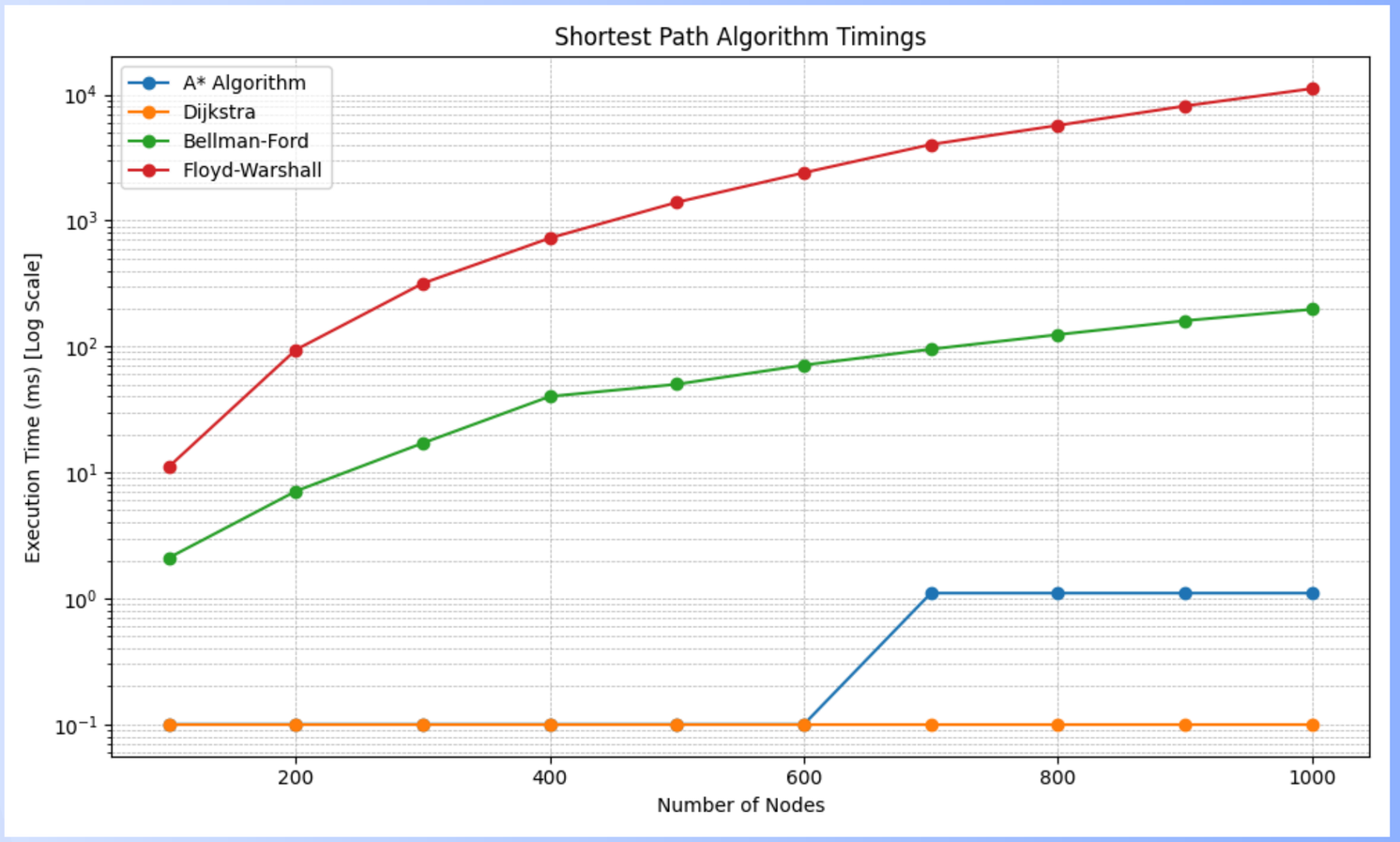
Dijkstra's Algorithm:
Time Complexity: O((V + E) log V)
with a min-heap.

Bellman-Ford Algorithm:
Time Complexity: O(VE)

Floyd-Warshall Algorithm:
Time Complexity: $O(V^3)$

A* Algorithm:
Time Complexity : O(|V| log |V|)

Conclusion:
Dijkstra's algorithm offers the best performance for our use case.



Hence Dijkstra’s algorithm provides best performance for our use case as there are no negative edge weights in our debt graph.

DEBT GRAPH SIMPLIFICATION – GREEDY ALGORITHM

🧠 Objective:

Minimize the number of transactions by removing cycles and simplifying debts between users.

🔧 Step-by-Step Greedy Method:

Net Balance Calculation

For each user:

Net = Total Received – Total Owed

Positive → Creditor

Negative → Debtor

Create Two Lists:

Creditors (sorted descending by credit)

Debtors (sorted ascending by debt)

Greedy Matching:

Match top debtor & top creditor by the minimumof their balances.

Settle partial or full amount.

Update balances and repeat.

Result:

Fewer transactions

All debts cleared

Cycles Eliminated

🧮 Example:

Initial Debts:

A → B: ₹100

B → C: ₹100

C → A: ₹100

✅ After Simplification: No transaction needed (all balances = 0)

⚡ Why Greedy?

Simple & intuitive

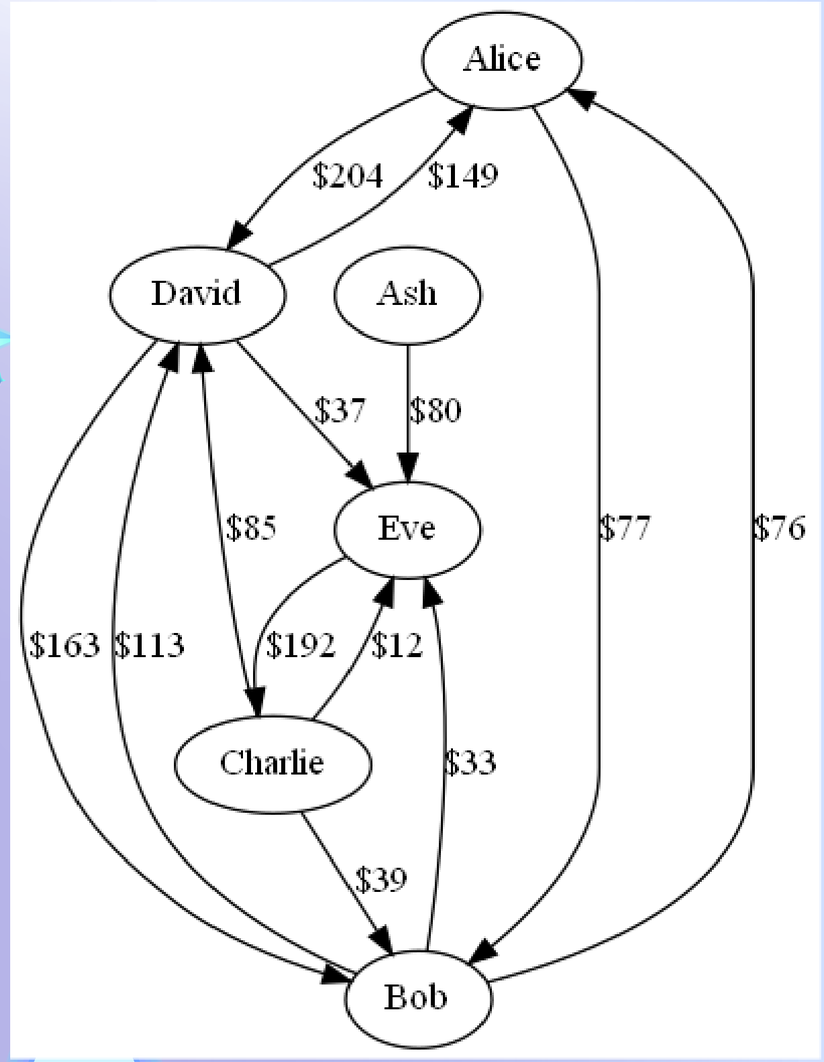
Efficient: O(N log N)

Avoids cycle detection

Near-optimal in practice

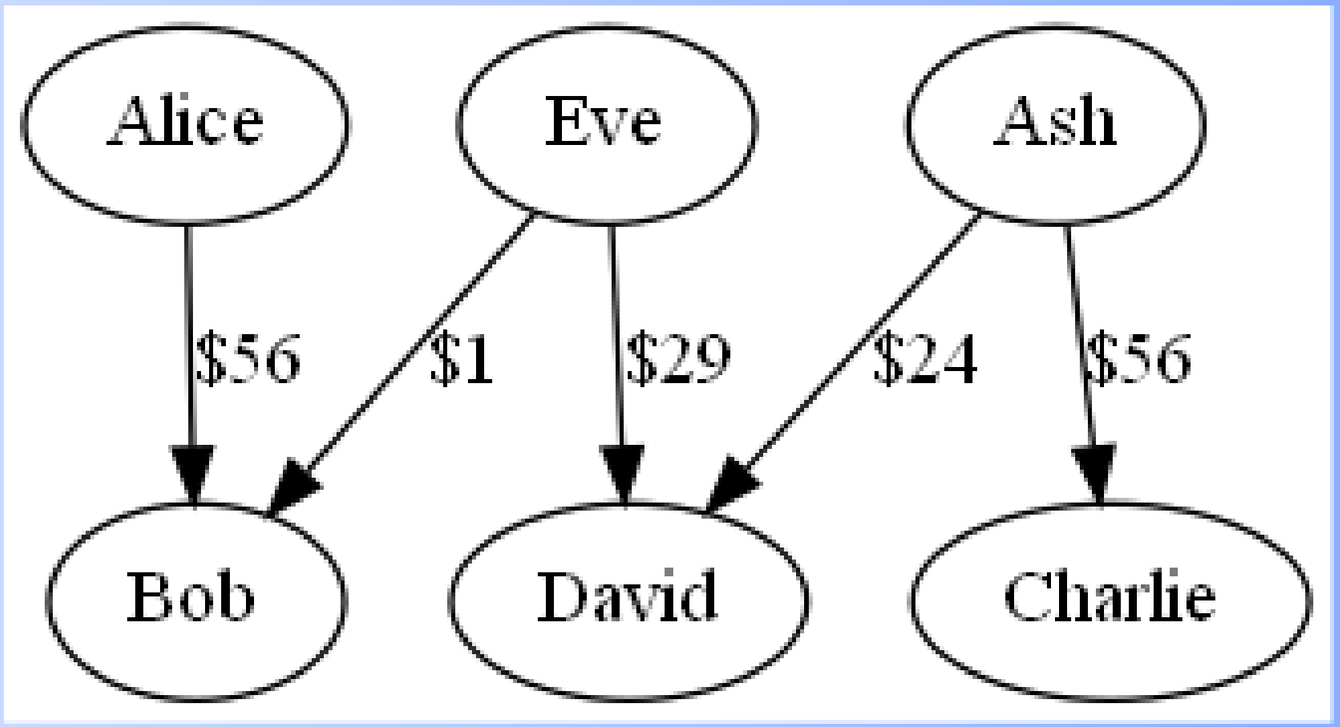
Github Link - https://github.com/Aashcharya1/Dynamic-Directed-Debt-Graph-for-Efficient-Settlement/tree/main

RESULTS



Initial Raw Graph
Number of edges = 13

Graph visualized using Graphviz Framework
Graphviz Documentation - https://graphviz.org/documentation/



Simplified Graph using our devised approach
Number of edges reduced to 5
Therefore over 50% reduction in complexity was obtained in most of the cases

CONCLUSION

Key Learnings

- Graph-based Modeling: Represented complex real-world debts as directed weighted graphs.
- Algorithmic Thinking: Designed a custom debt cycle simplification algorithm to minimize redundant transactions.
- DSA Application: Compared and identified best shortest path algorithm in this use case as Dijkstra's. Used priority queues and Dijkstra's Algorithm to determine the most influential user.

Scope for Future Extension

- Web-Based Interface: Visualize graphs interactively using frontend tools (e.g., D3.js).
- User Authentication: Add login systems for personalized debt tracking.
- Database Integration: Store user data and transactions persistently.
- Cycle Detection: Optimize further by directly eliminating cycles as they form.

Innovation

- Developed a real-time debt simplification engine that updates on every transaction.
- Enabled raw vs simplified graph visualization using Graphviz.
- Identified influential users based on graph connectivity and average path cost.

THANK YOU!

- Special thanks to Dr. Suchetana and Tammy ma’am for consistent guidance and project insights.
- Grateful to GeeksforGeeks for comprehensive explanations on:
- Graph Representations
- Dijkstra’s Algorithm
- Cycle Detection and Net Balancing
- Thanks to Striver’s DSA Sheet and YouTube series for helping solidify core DSA concepts.
- Appreciation to C++ STL documentation for efficient implementation support using:
- unordered_map, vector, priority_queue
- Thanks to peers for suggestions during testing and review phases.
- Also acknowledge the Graphviz community and documentation for enabling clear graph visualizations.