Augmenting the Approximation

Exact solution
Brute-force DFS → optimal but exponential

Approximation
Greedy selection of highest-weight edge → fast but not always accurate

Let's analyze the polynomial time algorithm!

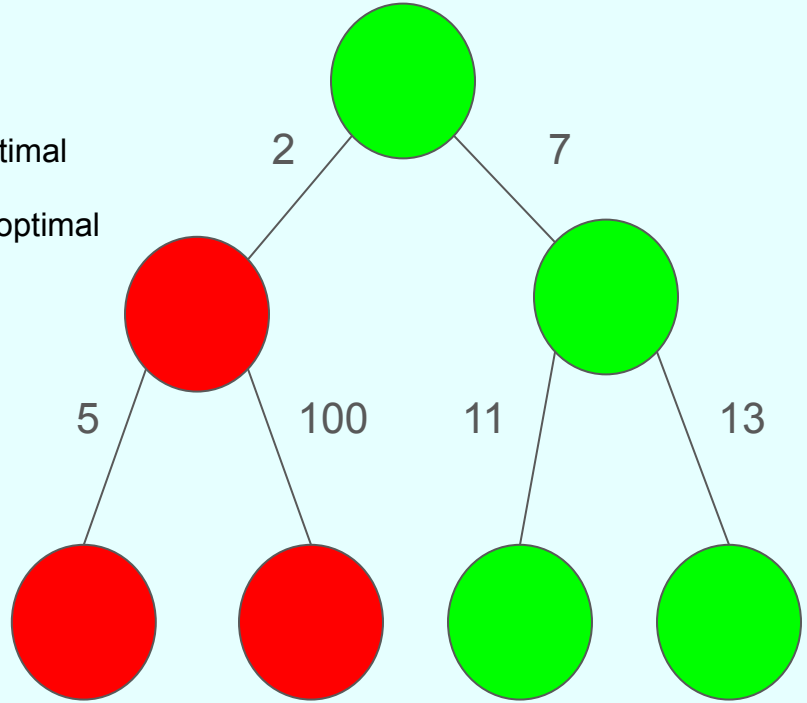
Observing Performance

Across 1,090 tests:

- Most approximation results fall between **70–90%** of optimal
- **5 test cases** showed *severe failures*: only **54–65%** of optimal

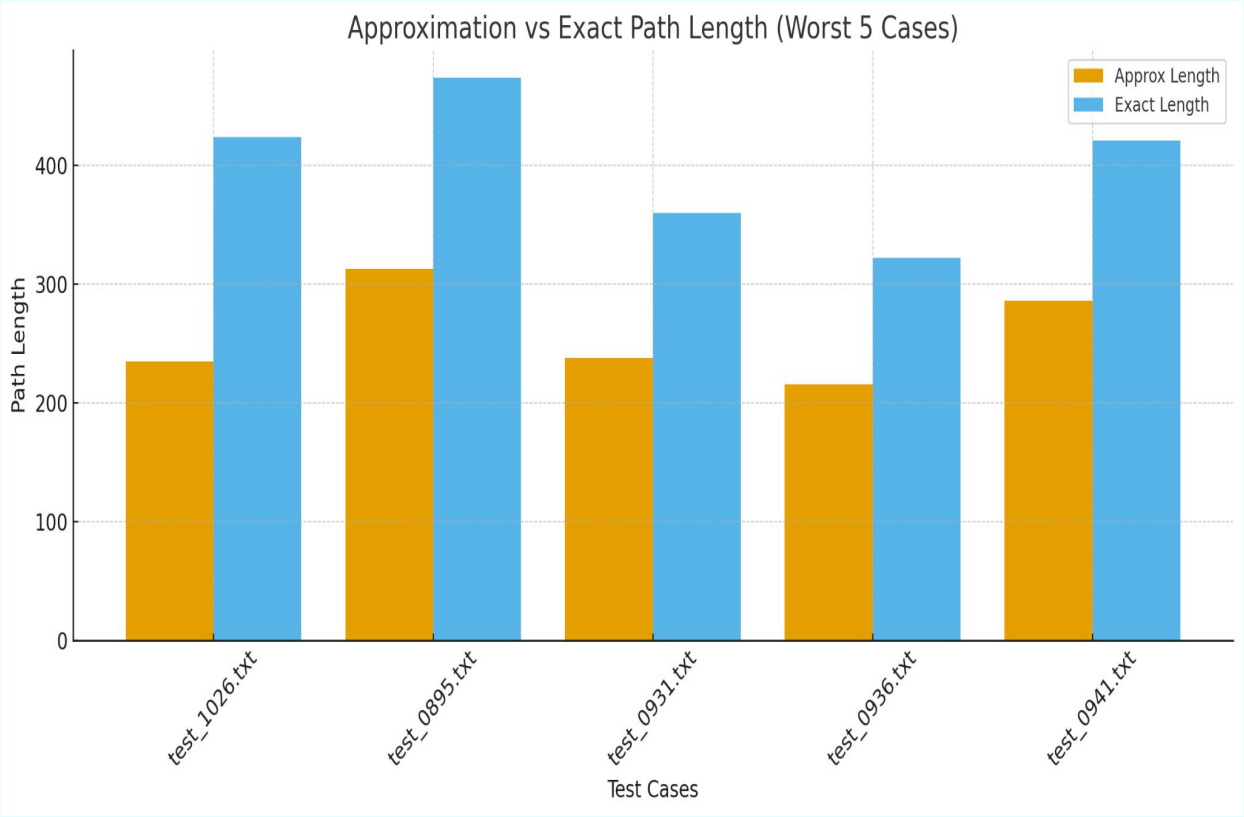
The greedy approach is usually OK…

…but occasionally collapses **dramatically**.



What Makes Greedy Fail?

- Choosing purley locally is globally destructive
- Therefore, the “solution” may get lead into dead regions
- Test cases with greedy “traps”



How Can We Make Suboptimal More Optimal?

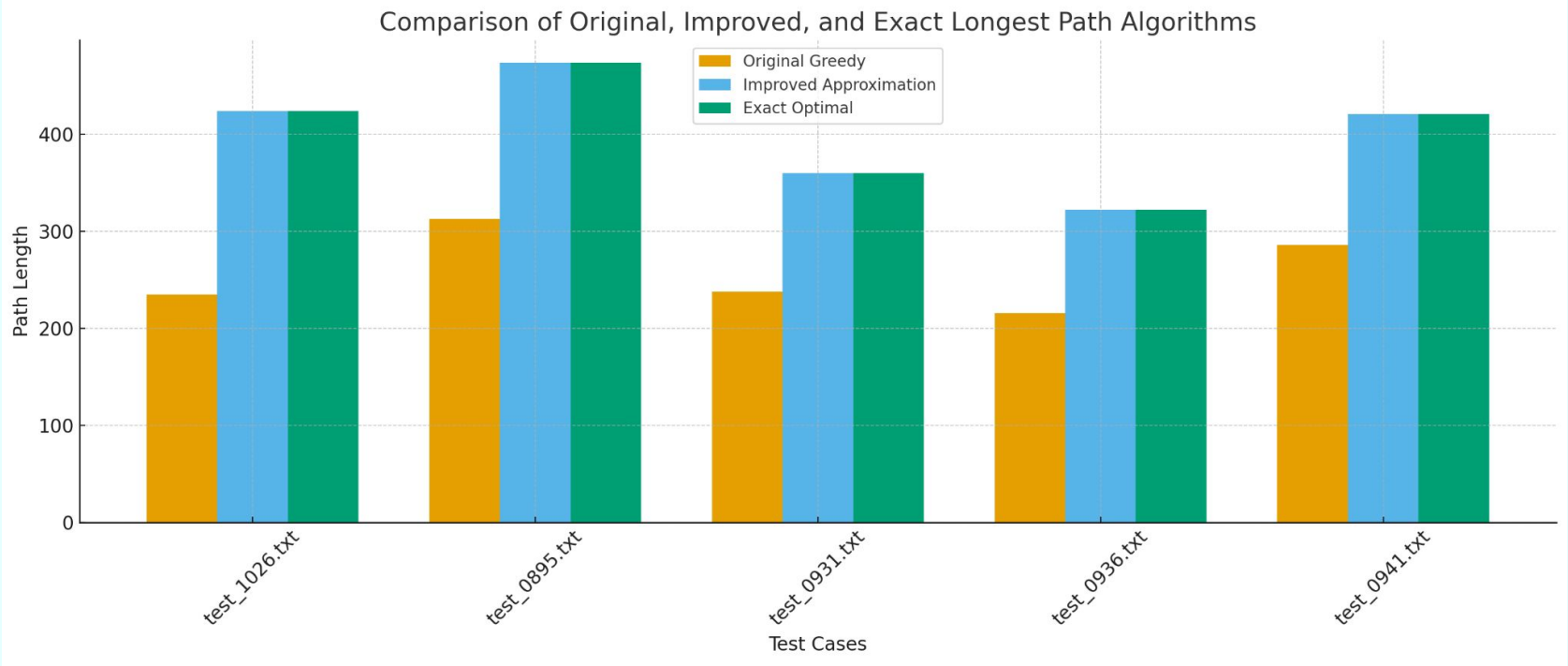
- Adding a Look Ahead :

"If I go here, What is the best thing i can do after that"

- Allowing a check for a better potential future to not lead into dead ends where the longest path cannot be achieved

- Changes from O(E) -> O(E * degree of vertex)

How the Look Ahead Augment Affected Accuracy on the Five worst test cases from before



Types of Graphs Where the Improved Approximation Algorithm failed

- The improved solution has a harder time with larger graphs
- > 9 Vertices
- > 15 edges