# Breaking a Long-Standing Barrier: 2-ε Approximation for Steiner Forest

Peyman Jabbarzade

Ali Ahmadi

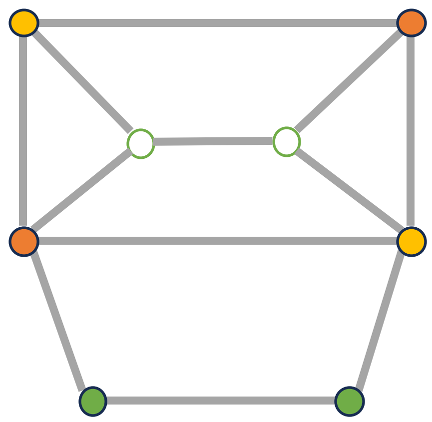
Iman Gholami

Mohammad T. Hajiaghayi

Mohammad Mahdavi

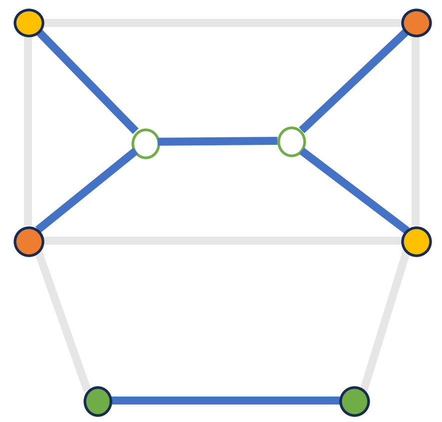
# Steiner Forest

- Given a weighted graph
- And a set of pairs of vertices



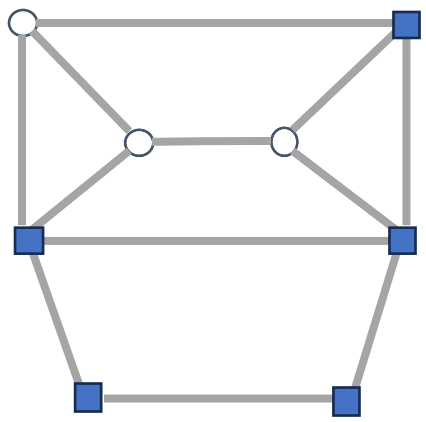
# Steiner Forest

- Given a weighted graph
- And a set of pairs of vertices
- Select a subset of edges to connect vertices in each pair
    - Minimize the total cost of selected edges
- Generalization of Steiner Tree



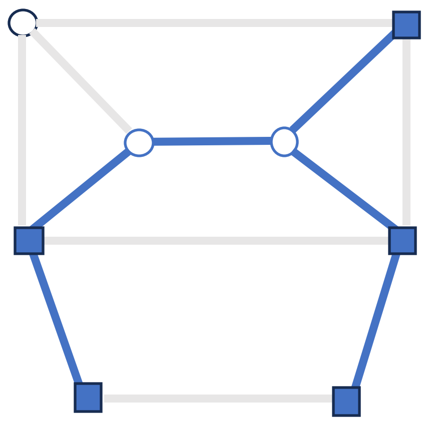
# Steiner Tree

- Given a weighted graph
- And a set of vertices as terminals



# Steiner Tree

- Given a weighted graph
- And a set of vertices as terminals
- Select a subset of edges to span the terminals
  - Minimize the total cost of selected edges
- Generalization of MST



# Previous Results

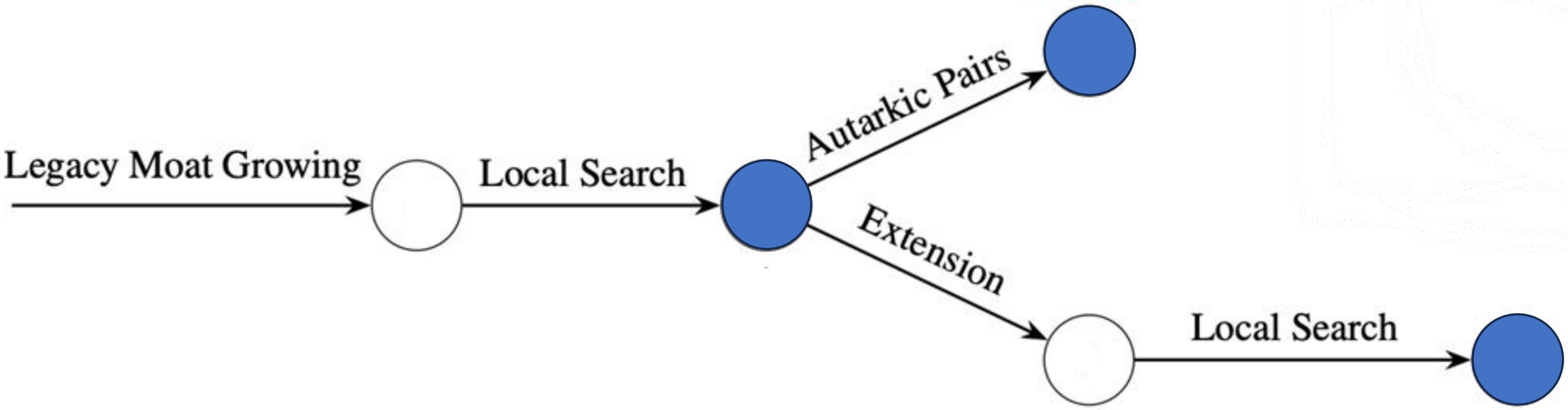
- Steiner Tree
  - Introduced in 1811
  - Trivial 2-approximation
  - 11/6-approx. [Zel, Algorithmica'93]
  - 1.65-approx. [KZ, J. Comb. Optim.'97]
  - 1.55-approx. [RZ, SIAM J. Discret. Math.'05]
  - 1.39-approx. [BGRS, STOC'10]
- Steiner Forest
  - 2-approx. [AKR, SIAM J. Comput'95]
    - Generalization and refinement [GW, SIAM J. Comput.'95]
  - 96-approx. Greedy [GK, STOC'15]
  - 69-approx. Local Search [GGKMSSV, ITCS'18]

# Our Contribution

- Steiner Tree
  - Introduced in 1811
  - Trivial 2-approx.
  - 11/6-approx. [Zel, Algorithmica'93]
  - 1.65-approx. [KZ, J. Comb. Optim.'97]
  - 1.55-approx. [RZ, SIAM J. Discret. Math.'05]
  - 1.39-approx. [BGRS, STOC'10]
- Steiner Forest
  - 2-approx. [AKR, SIAM J. Comput'95]
    - Generalization and refinement [GW, SIAM J. Comput.'95]
  - 96-approx. Greedy [GK, STOC'15]
  - 69-approx. Local Search [GGKMSSV, ITCS'18]

- Steiner Tree
  - 1.943-approx. [AGHJM'25]
- Steiner Forest
  - $(2-10^{-11})$-approx. [AGHJM'25]

# Our Algorithm Outline

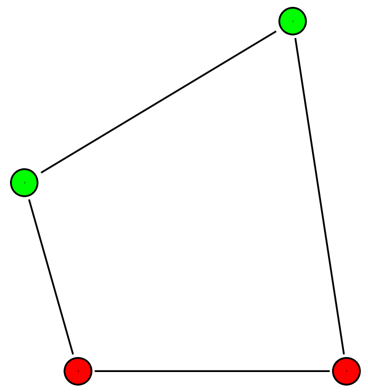


# Legacy Moat Growing Algorithm

The primal-dual 2-approximation algorithm

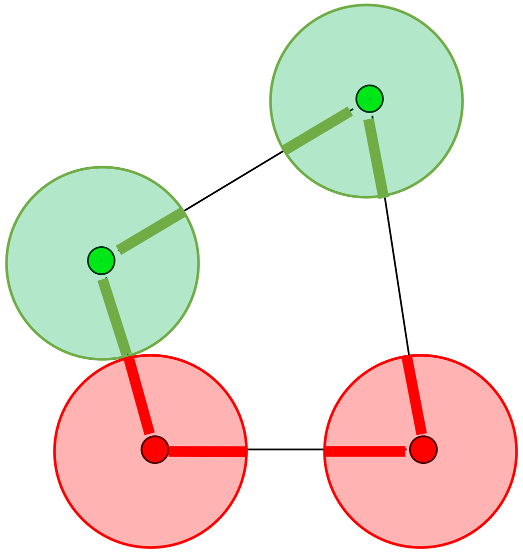
# Basics

- Edges are curves with length=cost
- Maintain a forest
  - Initially empty
- Active sets
  - Connected components need to extend



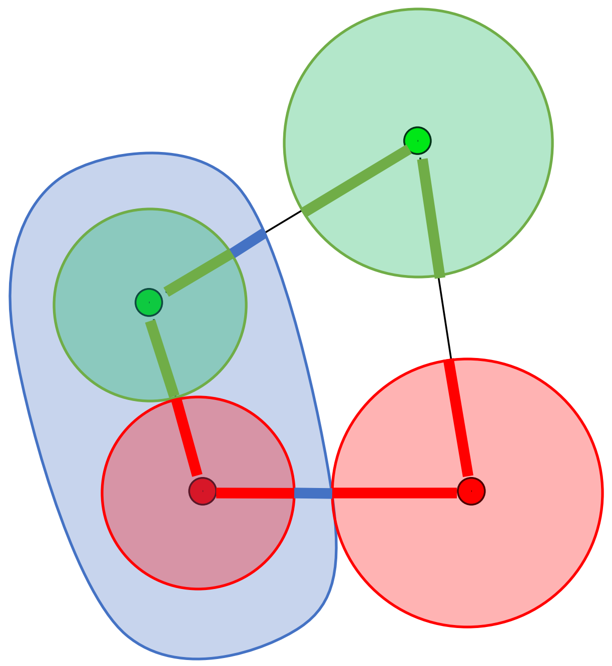
# Moat Growing

- Active sets color their adjacent edges
  - Edges with exactly one endpoint in them
- Once an edge becomes fully colored
  - Add the edge to our forest



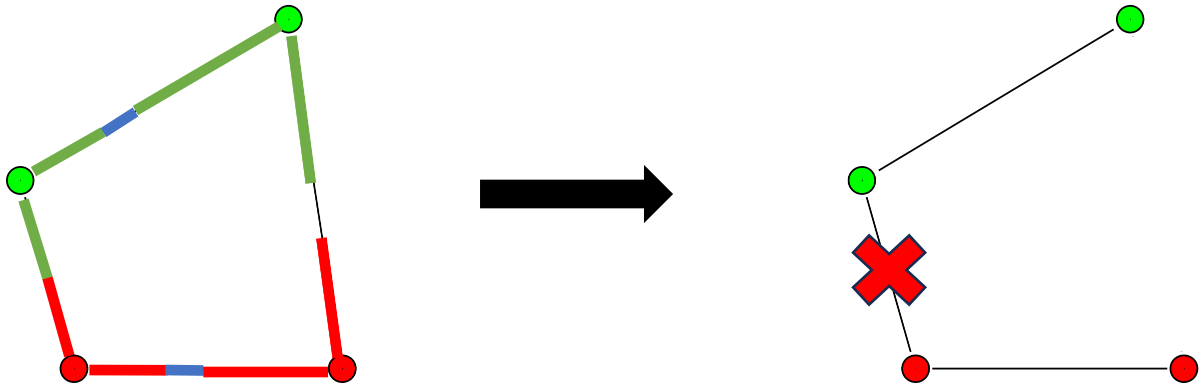
# Moat Growing

- Active sets color their adjacent edges
  - Edges with exactly one endpoint in them
- Once an edge becomes fully colored
  - Add the edge to our forest
  - Merge its endpoints connected components
  - Continue grow together (if still active)



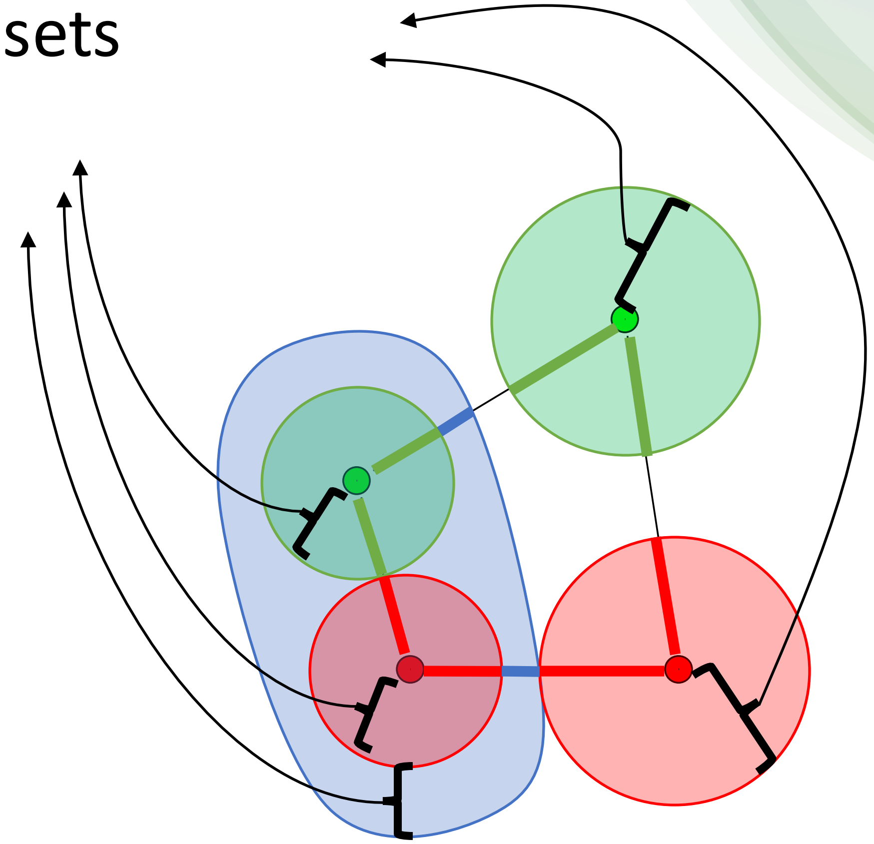
# Pruning Phase

- Remove redundant edges



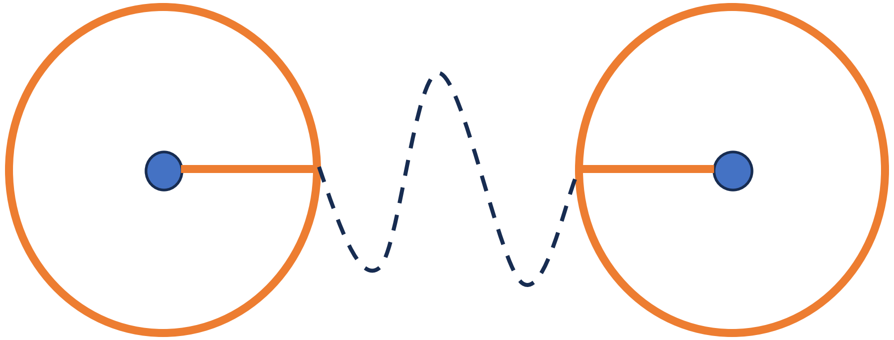
# Analysis

- R: Total growth duration across all active sets
- Optimal solution is at least R
- Our solution is at most 2R
- So, it is a 2-approximate solution



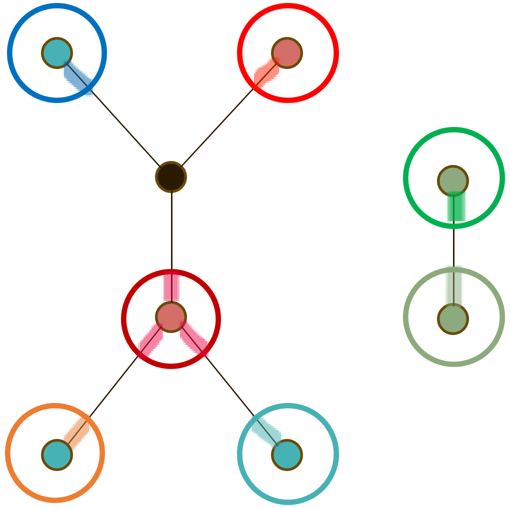
# OPT Lower Bound

- R: Total growth duration across all active sets
- Optimal solution is at least R
- A component is active for connecting some pairs
    - OPT connects those pairs
    - The active set cuts the path between them
    - So, it is coloring at least one edge of OPT



# Solution Upper Bound

- R: Total growth duration across all active sets
- Our solution is at most 2R
- All selected edges are fully colored
- At each moment of the algorithm
  - Active sets on average color at most two segments of it



# How to Improve the Algorithm?

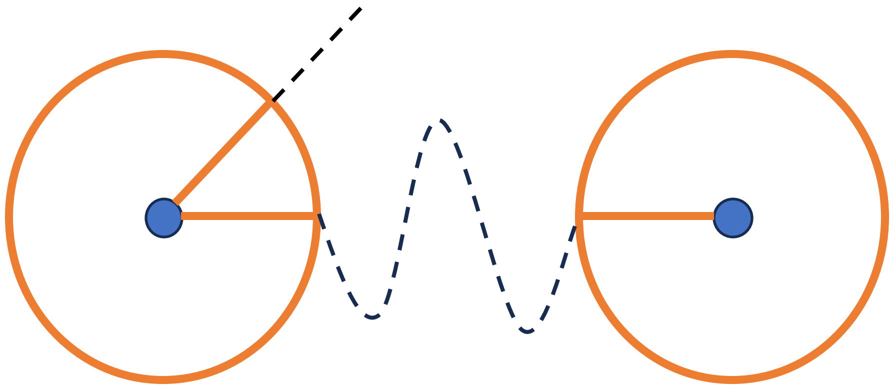
Observations on the analysis

# Recap Analysis

- R: Total growth duration across all active sets
- Optimal solution is at least R -> improve to $(1+\varepsilon)R$
- Our solution is at most 2R -> improve to $(2-\varepsilon)R$
- It is a 2-approximate solution -> $(2-\varepsilon)$-approximate solution

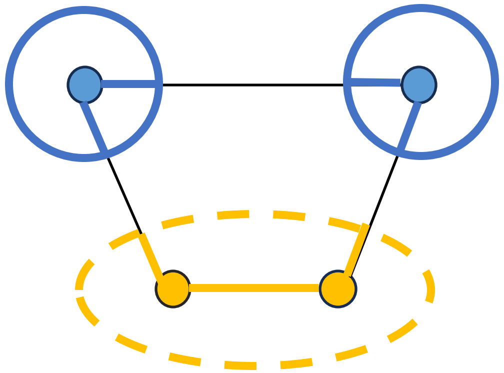
# Better Lower Bound for OPT

- R: Total growth duration across all active sets
- Optimal solution is at least R
- Active sets color at least one edge of OPT
- If they color multiple edges for ε fraction of total growth
  - OPT is larger than (1+ε)R



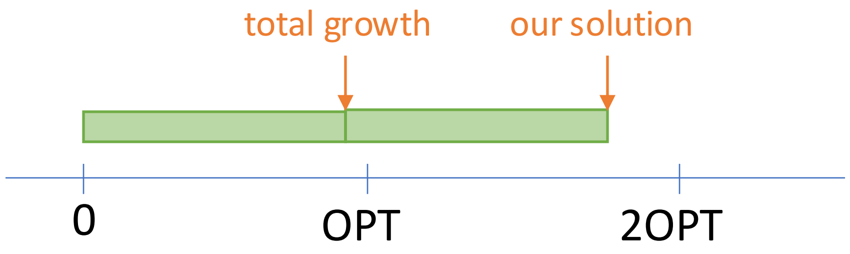
# Reduce Total Growth Duration (R)

- R: Total growth duration across all active sets
- Our solution is at most 2R
- Modifying active sets behavior
  - Reducing total growth by a constant factor
  - Resulting a new solution
    - Pairs should be still connected
- The new solution is at most (2-ε)R



# Objective

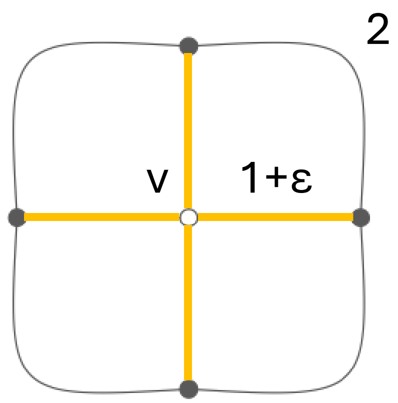
- R: Total growth duration across all active sets
- Optimal solution is at least R -> improve to (1+ε)R
- Our solution is at most 2R -> improve to (2-ε)R
- Reduce total growth to obtain
  - Total growth < (1-ε)OPT

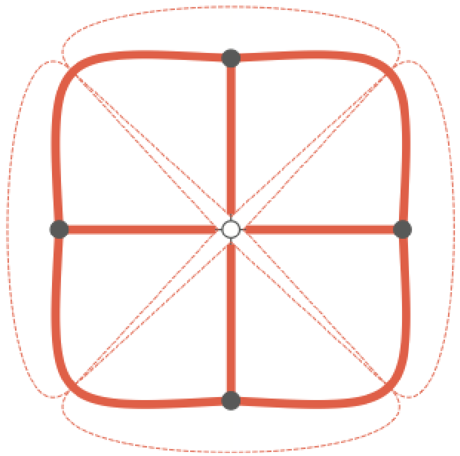


# Local Search

Incremental reduction on total growth

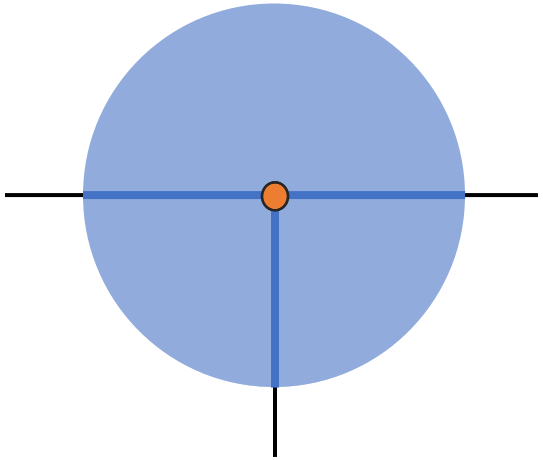
# An Example on Steiner Tree



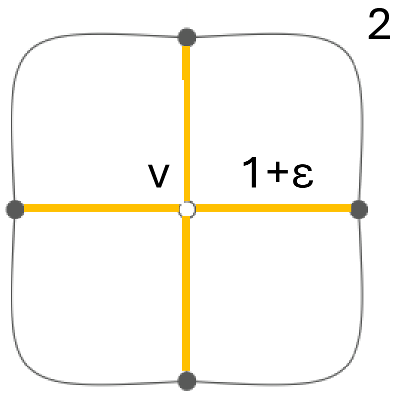



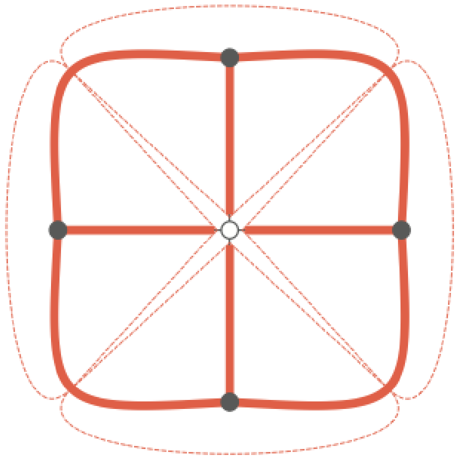
# Boost Action

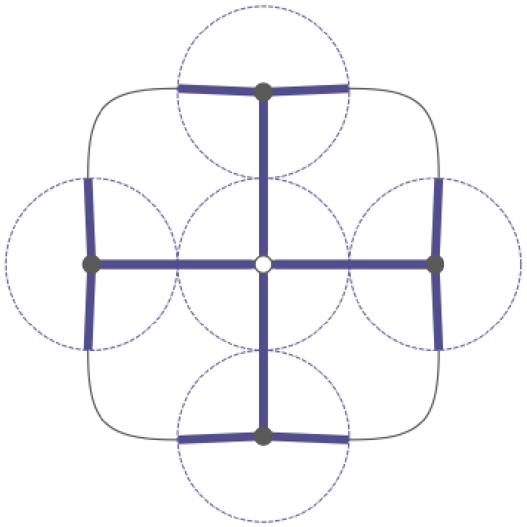
- Select a vertex and a specific time
  - Let the vertex remains active until that time
- Win: Reduction in total growth
- Loss: Additional growth introduced by the boost action
- Valuable boost action
  - Win > β.Loss
- Apply any valuable boost action while one exists



# An Example on Steiner Tree

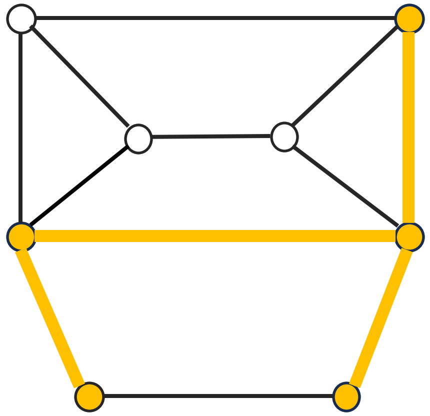




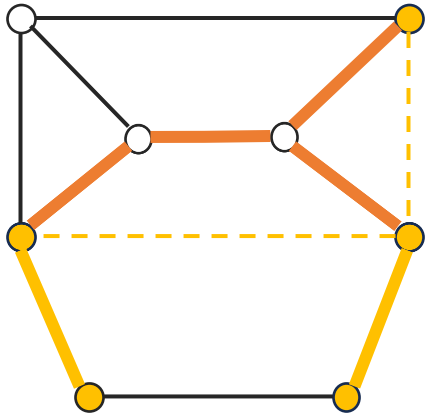
# Previous Local Searches

- Steiner Tree
  - 11/6-approx. [Zel, Algorithmica'93]
  - 1.55-approx. [RZ, SIAM J. Discret. Math.'05]
- Start with a 2-approximate solution
- Find a constant-size subgraph
  - Such that adding it improve the solution
- Do this until no improvement is possible



# Previous Local Searches

- Steiner Tree
  - 11/6-approx. [Zel, Algorithmica'93]
  - 1.55-approx. [RZ, SIAM J. Discret. Math.'05]
- Start with a 2-approximate solution
- Find a constant-size subgraph
  - Such that adding it improve the solution
- Do this until no improvement is possible

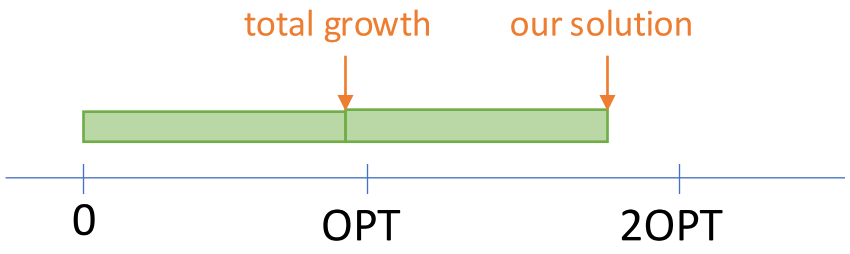


# Comparison with Previous Approaches

- Our objective is reducing total growth
  - Instead of the solution cost
- May lead to a worse solution
  - Give a better upper bound for the solution
  - And more structural properties
  - And make it possible to analysis with growth of active sets

# Recap: Objective

- R: Total growth duration across all active sets
- Optimal solution is at least R
- Our solution is at most 2R
- Reduce total growth to obtain (local search)
    - Total growth < (1-ε)OPT



# Claw Property

A result of our local search

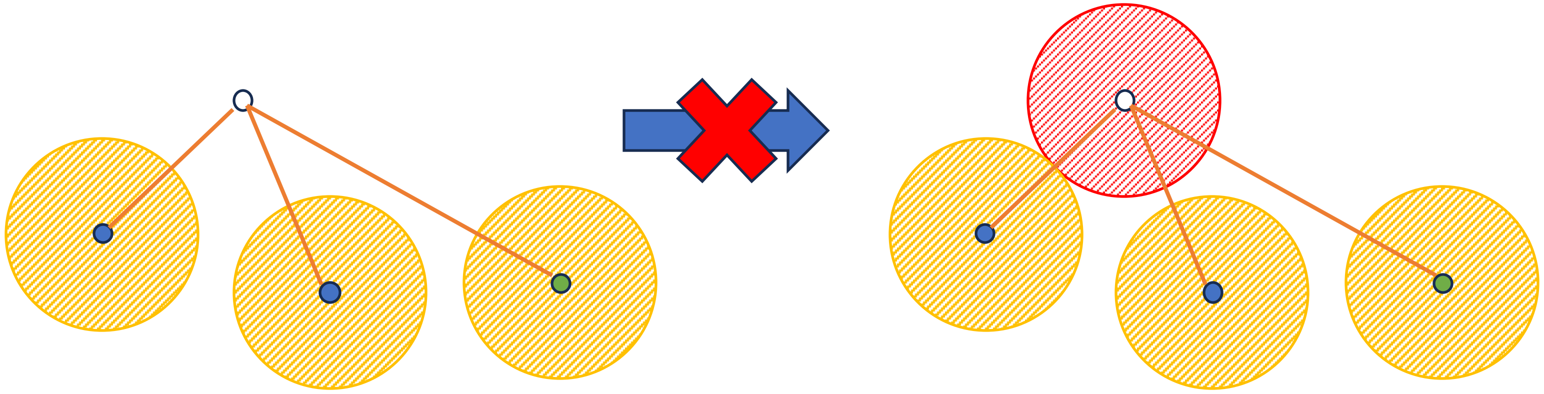
# Claw Property

- After our local search
  - No valuable boost action
    - Win < β.Loss
    - Win = current total growth – total growth after boost action

$\downarrow$ I∧

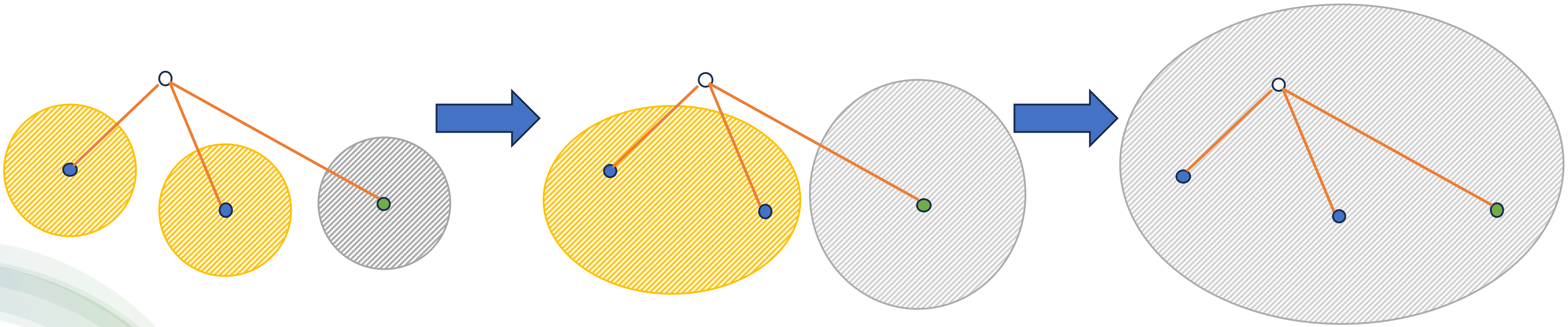
½ claw's cost + max growth

- Current total growth ≲ ½ claw's cost + max growth



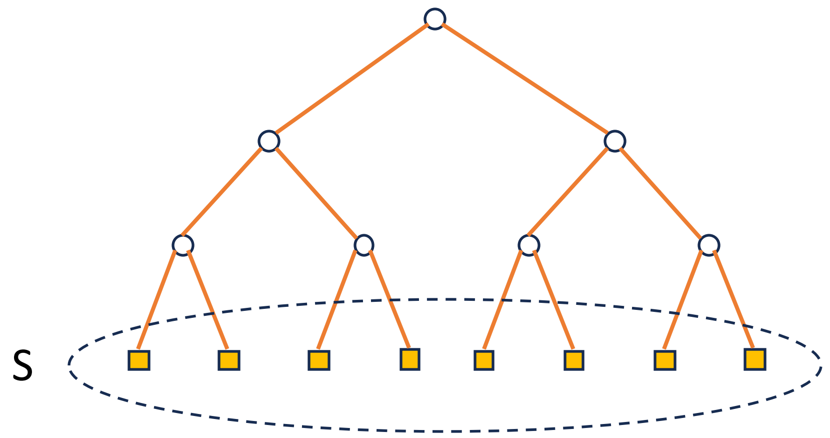
# Claw Property

- For any three vertices
- Total growth except max ≲ ½ claw's cost
  - Objective: Total growth < (1-ε)OPT



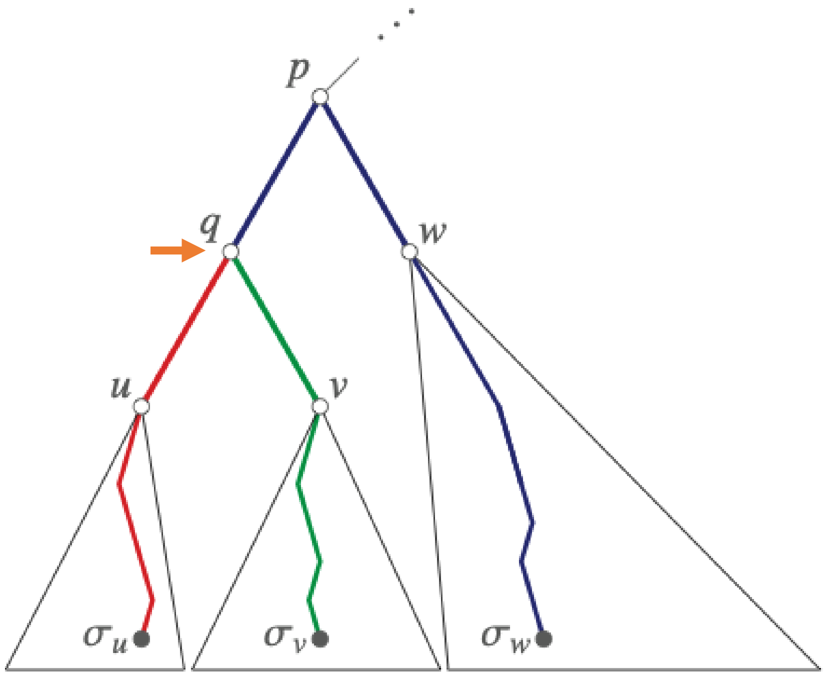
# Generalized Claw Property

- For any number of vertices $S$
- Bound the total growth corresponds to vertices in $S$
  - By a factor (1-ε) of the cost of any tree connecting them
- Assume WLOG that tree is binary with $S$ as leaves



# How Does It Work?

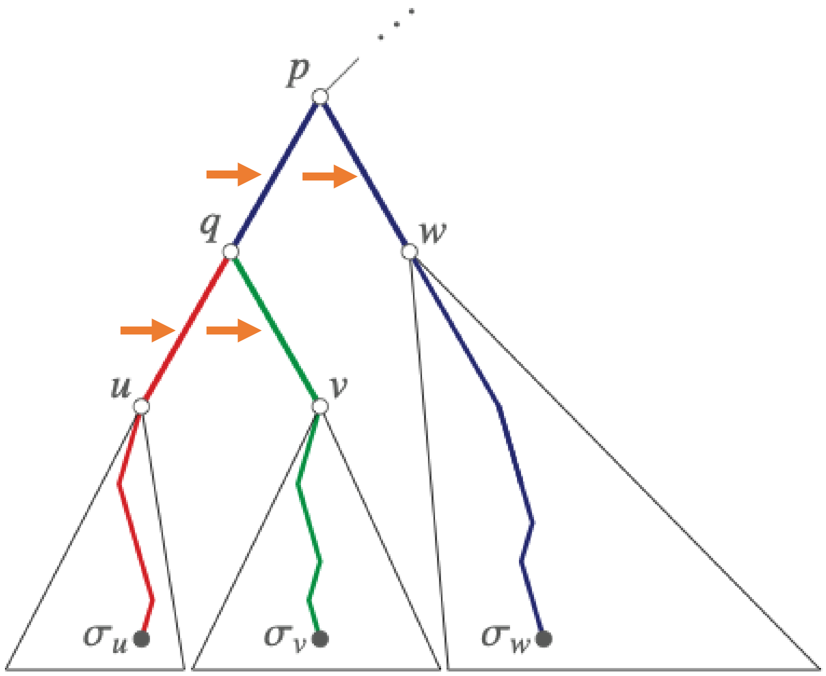
- For any vertex of $T$
  - Its representative is the nearest leaf in its subtree
  - Fix the three representatives of
    - Its two children and its sibling
  - Write claw property for these leaves
    - (total-max) growth $\lesssim$ ½ claw's cost
- Sum up the inequalities



# How Does It Work?

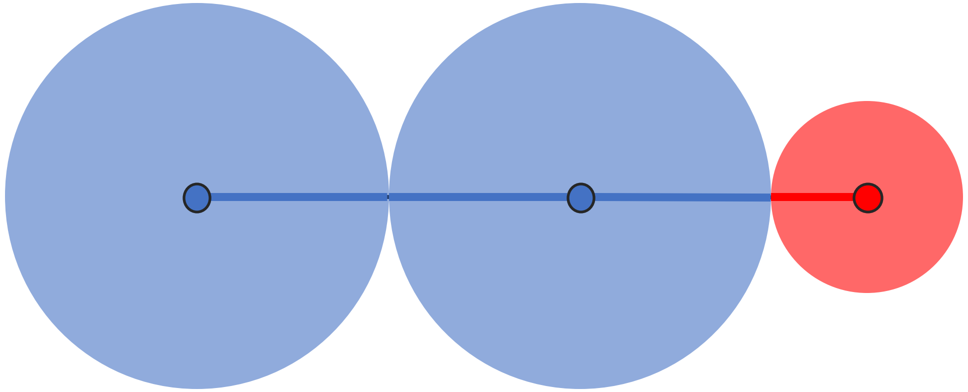
$$
\begin{array}{rl}
& \text{(total-max) claw growth} \lesssim \tfrac{1}{2}\text{ claw's cost} \\
+ & \text{(total-max) claw growth} \lesssim \tfrac{1}{2}\text{ claw's cost} \\
+ & \ldots \\
+ & \text{(total-max) claw growth} \lesssim \tfrac{1}{2}\text{ claw's cost} \\
\hline
& 3\text{(total–max) leaves growth} \lesssim 5/2\text{ tree's cost}
\end{array}
$$

- Total growth – max $\lesssim$ 5/6 cost of tree
  - Only works when vertices are actively connected



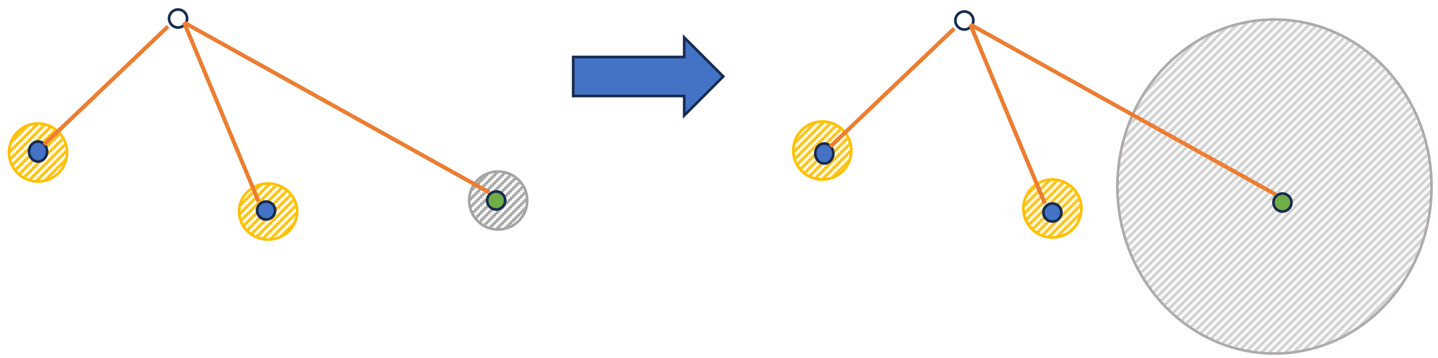
# Actively Connect

- Two vertices are actively connected if
  - They are in active sets until reaching each other



# Issue With Not Actively Connected

- Total growth except max $\lesssim$ ½ claw’s cost
- Only comparing small value with claw’s cost



# Recap: How Does It Work?

(total-max) claw growth $\lesssim$ ½ claw's cost

\+ (total-max) claw growth $\lesssim$ ½ claw's cost

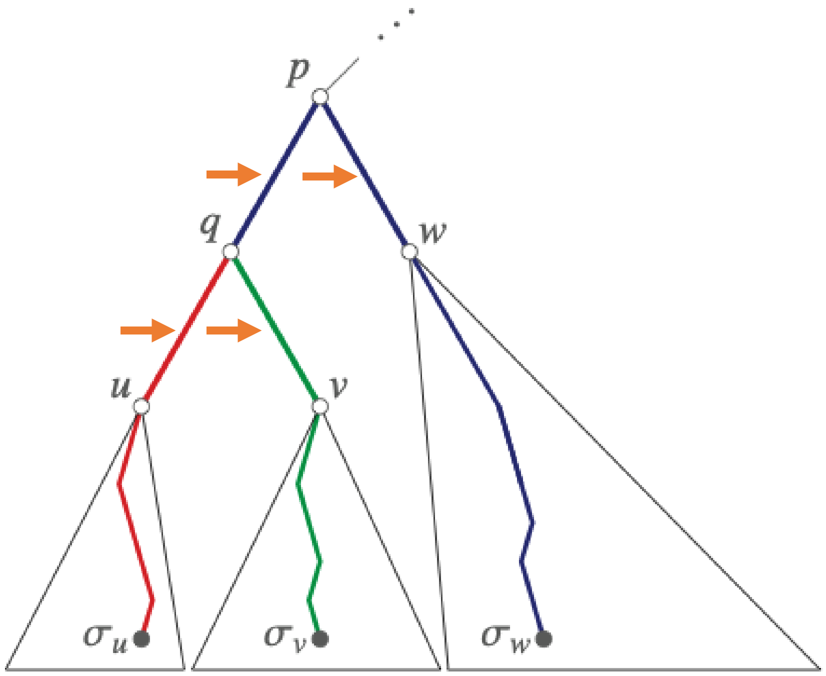
\+ …

\+ (total-max) claw growth $\lesssim$ ½ claw's cost

---

3(total–max) leaves growth $\lesssim$ 5/2 tree's cost

- Total growth – max $\lesssim$ 5/6 cost of tree
  - Only works when vertices are actively connected



# Claw Property in Steiner Tree

- All terminals are actively connected
- Generalized claw property applicable
- Local search achieves 1.943-approximate solution

# Claw Property in Steiner Forest

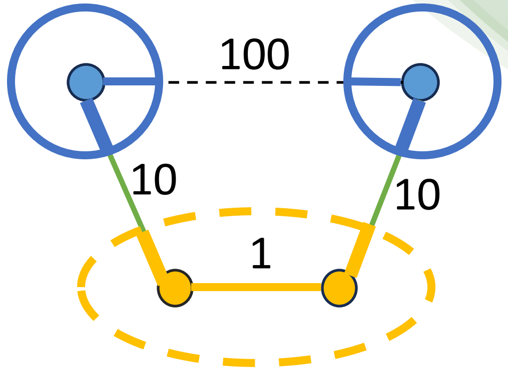
- For each connected component in the OPT
  - Want to bound the growth corresponds to its vertices
    - By the cost of that component
- But
  - Those vertices may not be actively connected
    - Extension
  - Bound excludes the maximum growth
    - Autarkic pairs

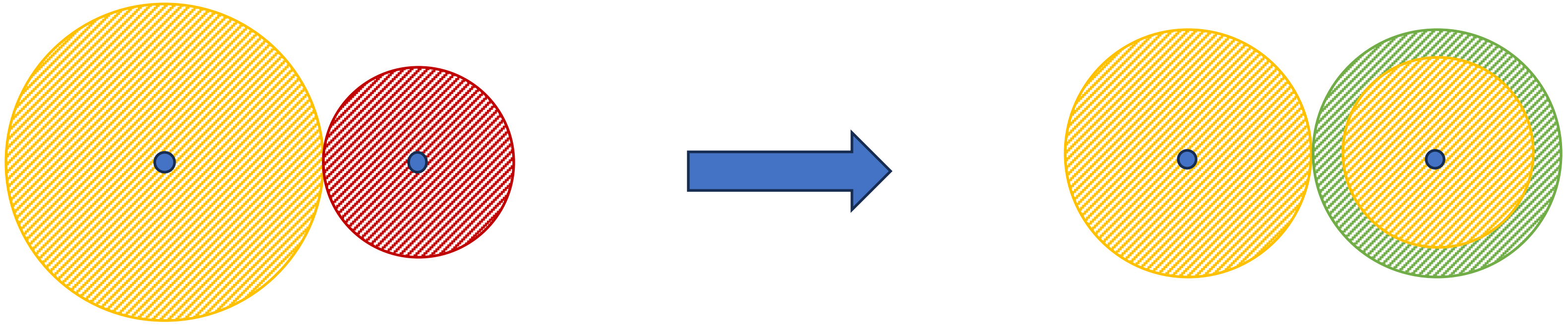
# Extension

When vertices are not actively connected

# Why Extension?

- For applying claw property
  - Vertices need to be actively connected
- Not guaranteed for components of OPT
- Idea: Let components remain active a bit more
- More vertices connect actively



# Extension

- Given a moat growing algorithm
- Assign a potential to each active set
  - $\epsilon$ fraction of its growth
- Run the moat growing algorithm again
- Sets consume potential of their subsets to remain active

# How Does It Help?

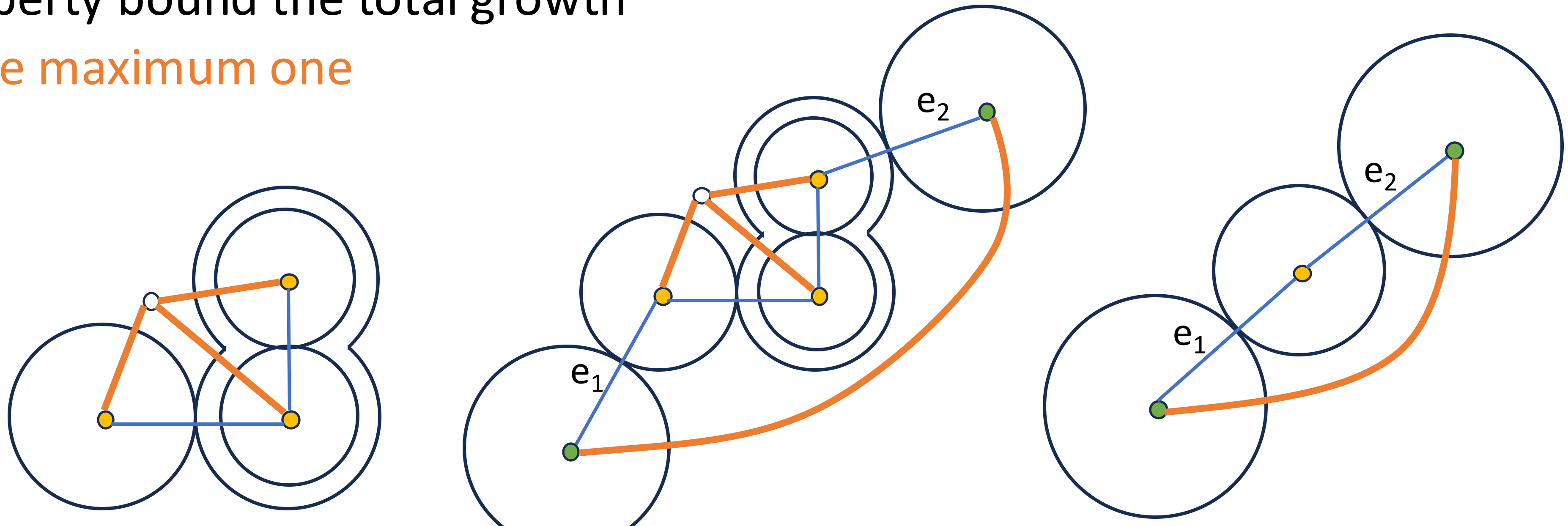
- If most vertices in a component of OPT actively connect
  - Use generalized claw property
- Otherwise
  - Vertices in that component must be far
  - Cost of OPT is already large compared to total growth
- We find a better lower bounds for OPT either way

# Autarkic Pairs

For large maximum growth

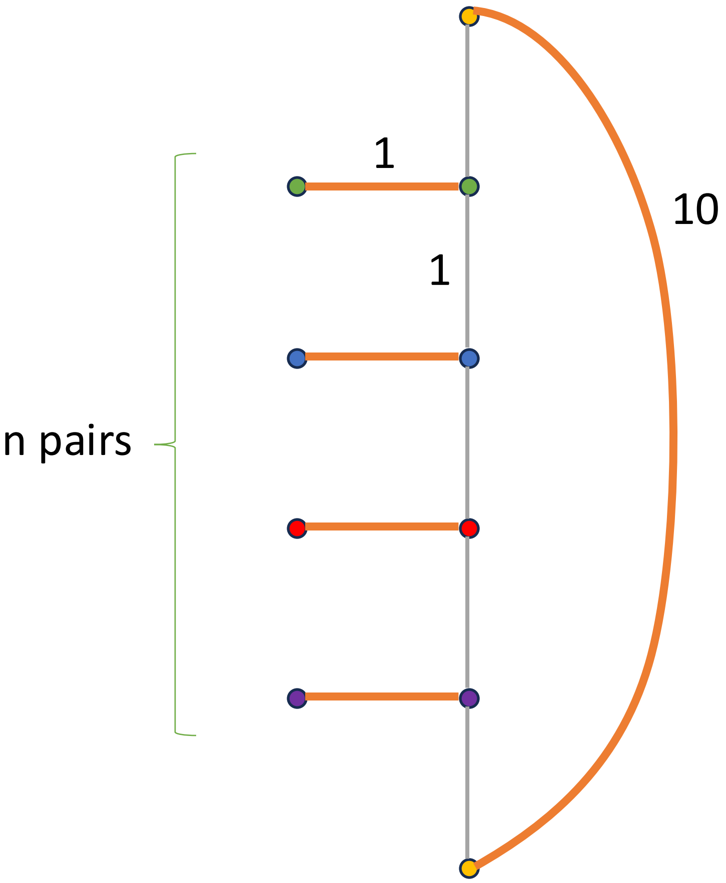
# Claw Property Shortcoming

- For applying claw property on a set of vertices
  - They need to be connected in the optimal solution
- For each OPT's component
  - Claw property bound the total growth
  - Except the maximum one

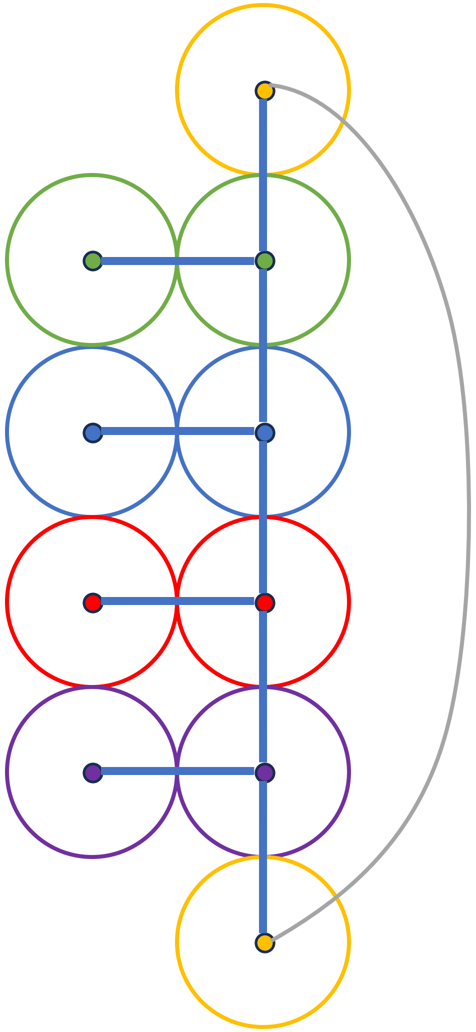



(1) (2) (3)

# Example



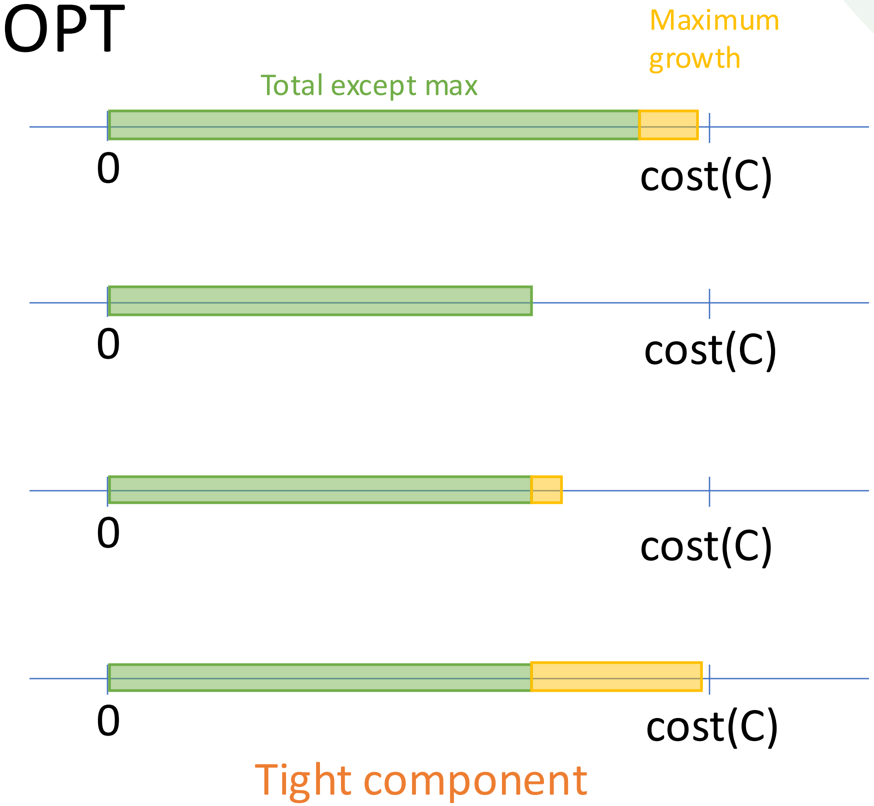

Cost: n+10



Cost: 2n+1

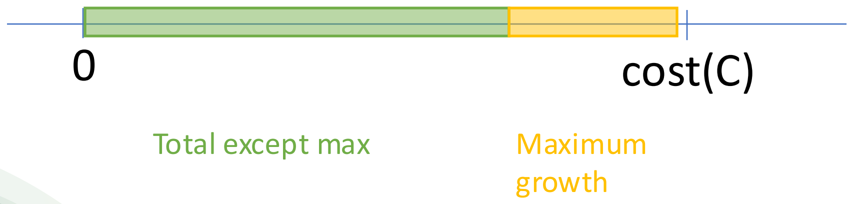
# Classify Components of OPT

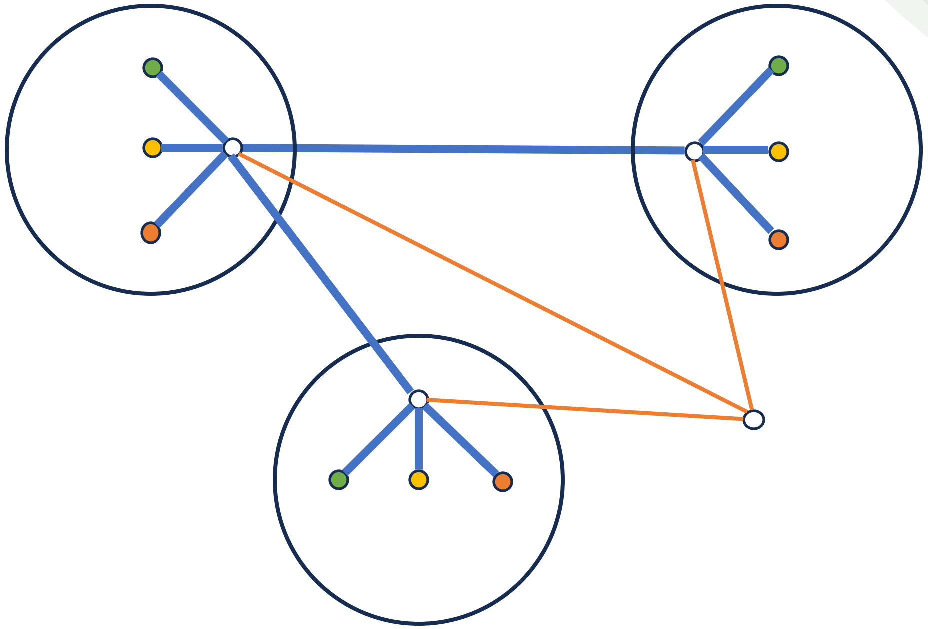
- Let C be a connected component in OPT
  - Total growth $\leq$ cost(C)
- Claw property
  - Total growth except max
- If maximum growth is small
  - Total growth is small
- Otherwise
  - Claw property is not enough



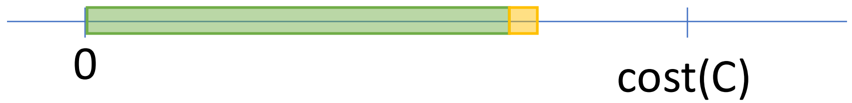
# Tight Components Structure

Tight component



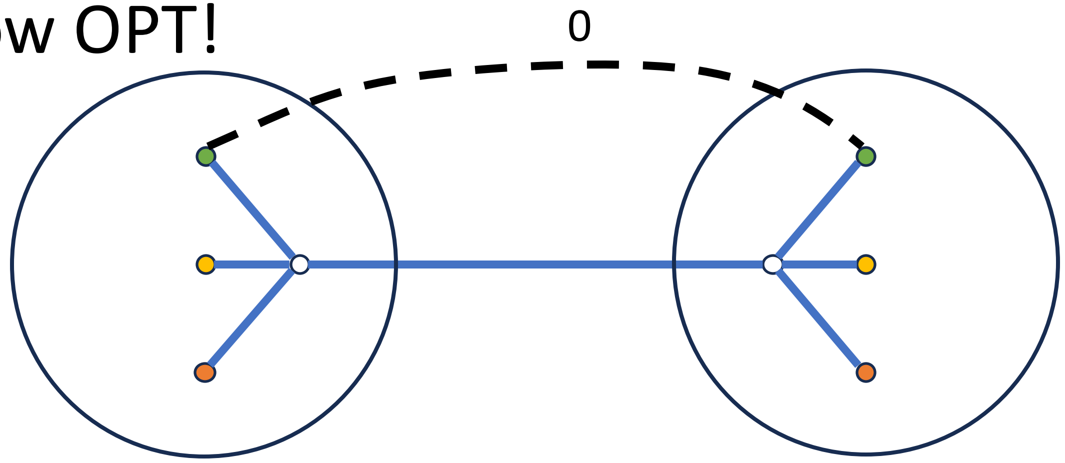



Use claw property
OPT connecting them is large
Total growth less than cost(C)



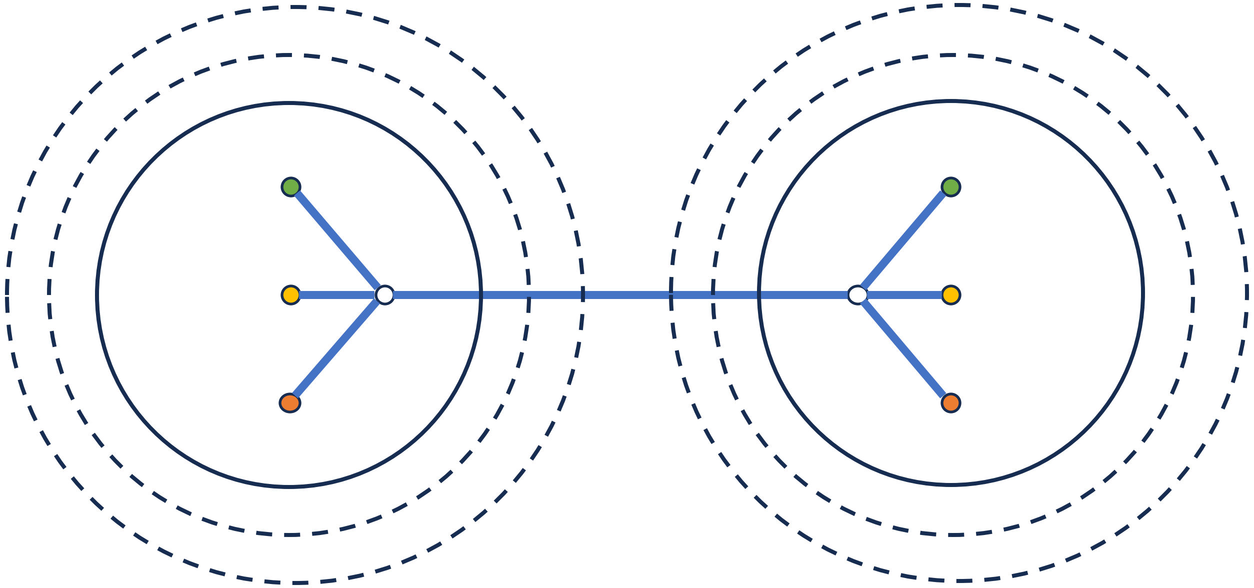
# How to Handle Tight Components?

- Since we have two far groups
- Buy shortest path for (only) one pair
- Add zero-cost edge between them
- Rerun the moat growing algorithm
  - Those groups do not need to grow excessively
- But we don't know OPT!



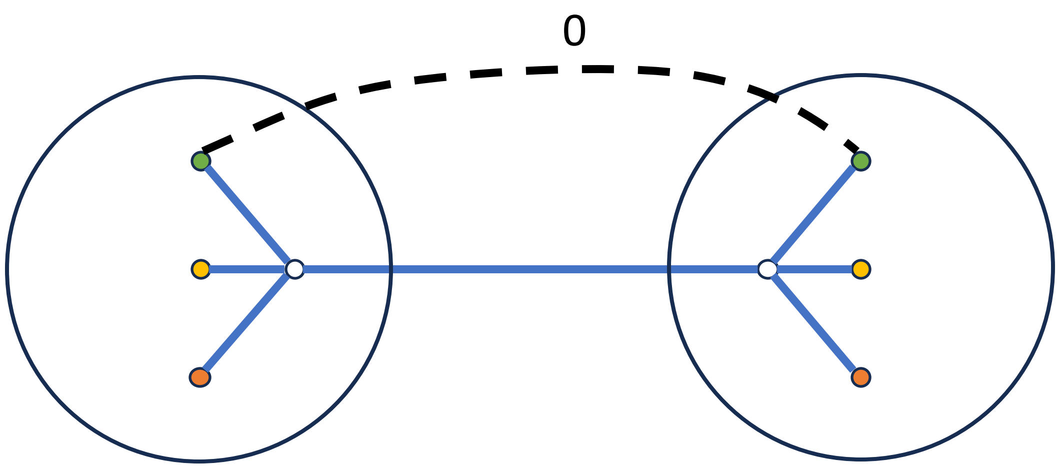
# Autarkic Pair

- Two group of unsatisfied vertices
- Pair of each other
- Vertices in each group grow only with each other
- Their distance almost match their growth

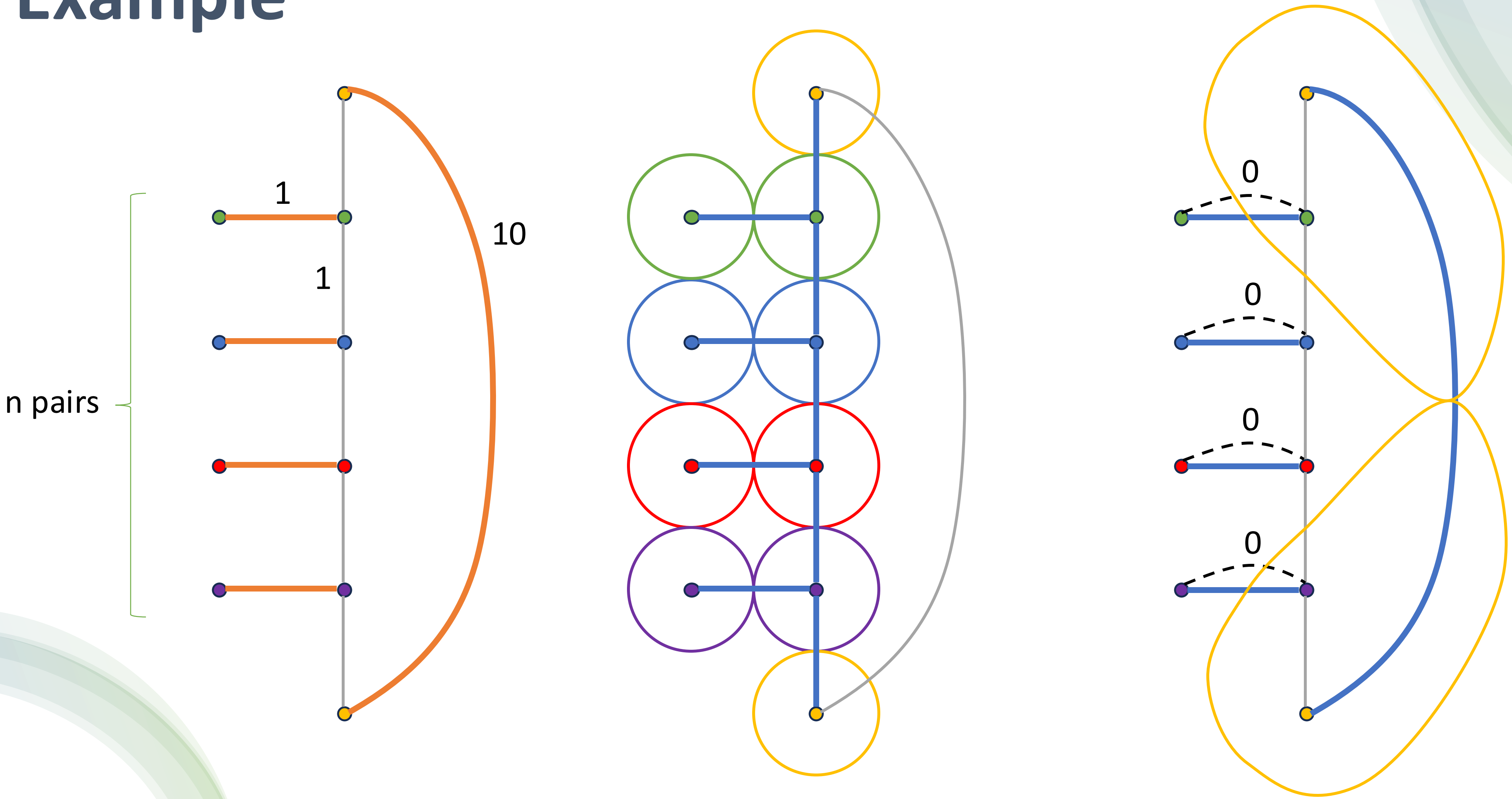


# Algorithm

- Find autarkic pairs
- Connect one pair of vertices from each group
- Add a zero-cost edge
- Rerun Legacy Moat Growing algorithm



# Example



# Intuition of Analysis

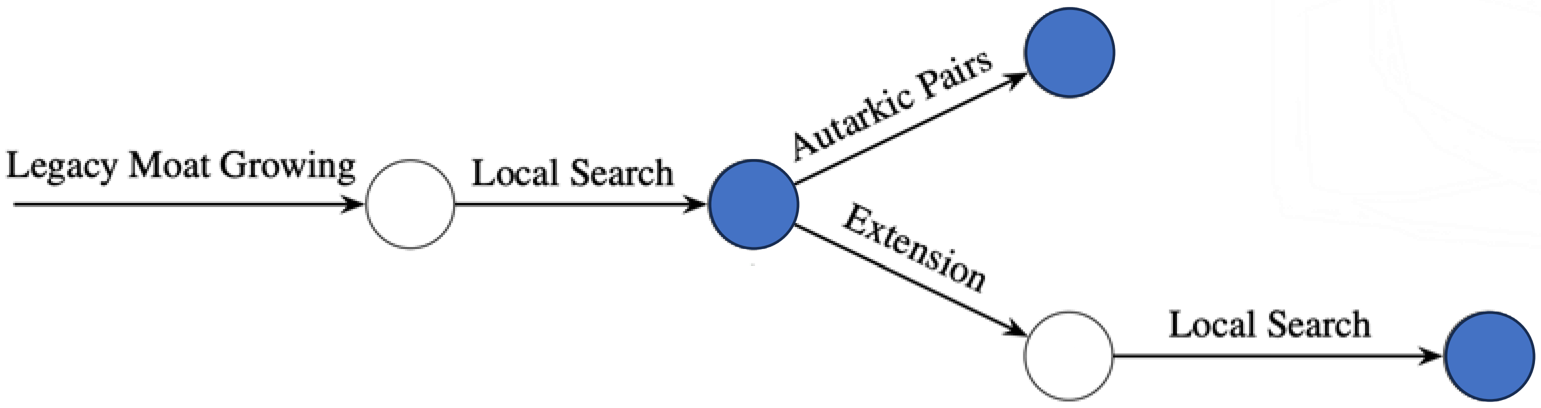
- A general idea to improve an $\alpha$–approx. algorithm ($\alpha$OPT)
- Find a structure with cost X and add it to our solution
  - Here structure is a subset of edges
- Set the structure cost equal to 0
- Let OPT' of the new instance be reduced by Y (OPT'=OPT-Y)
- Run $\alpha$–approx. algorithm on the new instance ($\alpha$(OPT-Y)+X)
- If $X < (\alpha-\varepsilon)Y$ (improvement condition)
  - And if X (or Y) is a constant fraction of OPT (significant condition)
    - We have an ($\alpha$-$\varepsilon$)-approx. algorithm

# Intuition of Analysis

- $\alpha$=2 (improving 2-approximation algorithm)
- The structure with cost X
  - Shortest paths between groups of each autarkic pair
- Set the structure cost equal to 0
- Let OPT' of the new instance be reduced by Y (OPT'=OPT-Y)
- Run 2–approx. algorithm on the new instance (2OPT-2Y+X)
- X < (2-ε)Y (X≈Y)
  - If tight components are significant (significant condition)
    - We have a (2-ε)-approx. algorithm

# Conclusion

- For tight components
  - Autarkic pair works well
- For others
  - The local search after the extension



# Future Work

- Improve the approximation factor
- Find a lower bound for this method
  - Currently, we have 1.5 for Steiner Tree
- Beat 2-approximation factor for generalizations
  - Survivable Network Design
  - Prize-Collecting Steiner Forest

# Questions?

Thank You!