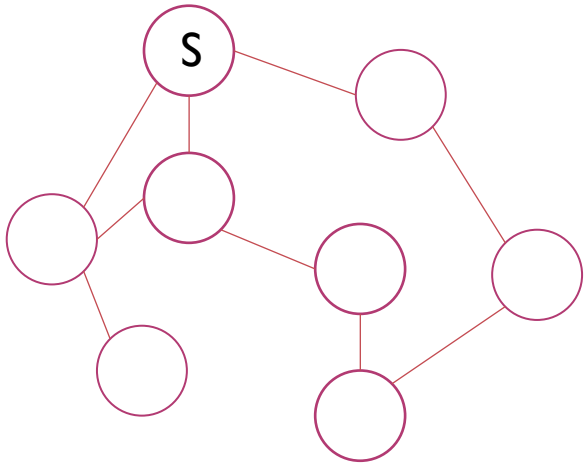
BRANCH AND BOUND

2LT JEREMY BANKS

APPLICATION TO RESEARCH

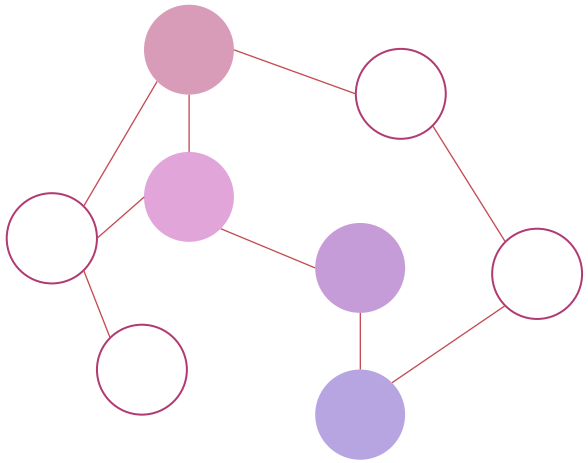
- Solving the WTA problem
  - Used by Gibbons (who took their solution from Ahuja)
- "has become the most commonly used tool for solving NP-hard optimization problems" – Wikipedia
- Searching for a faster way to achieve optimums

THE ALGORITHM



THE ALGORITHM

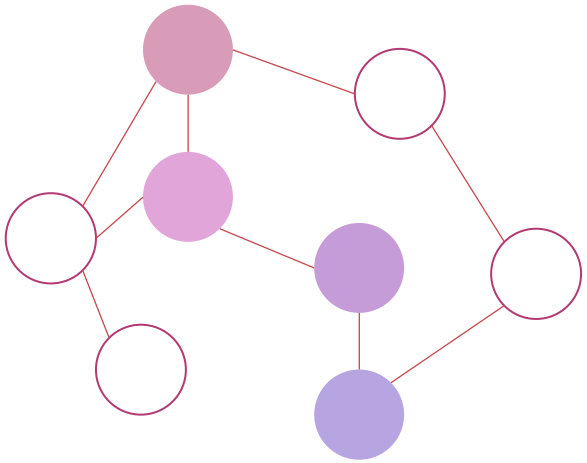
Use an upper-bound heuristic and store that solution as the Best-So-Far

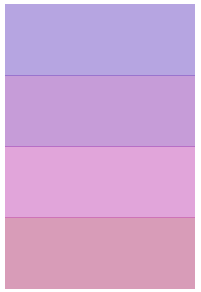


THE ALGORITHM

Use an upper-bound heuristic and store that solution as the Best-So-Far

Add all of the nodes in the heuristic solution to a stack





THE ALGORITHM

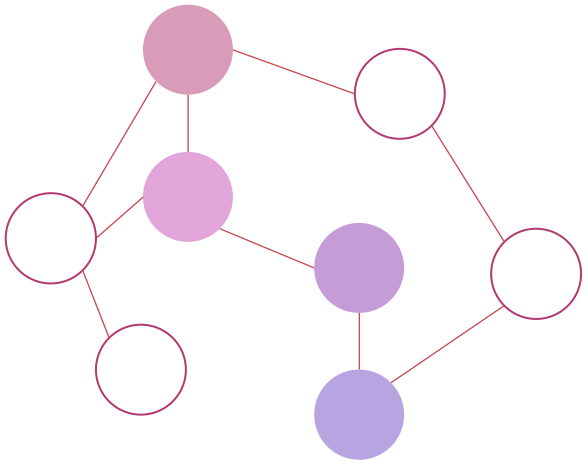
Use an upper-bound heuristic and store that solution as the Best-So-Far

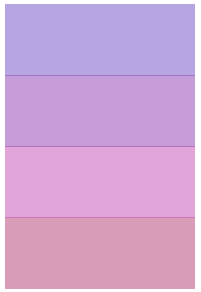
Add all of the nodes in the heuristic solution to a stack

Pop the last node in the stack

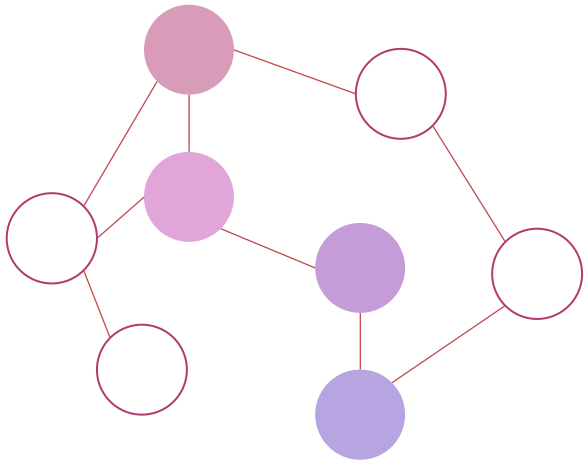
If the node is terminal, evaluate it and compare it with B

Store better solutions, discard all others





THE ALGORITHM



Use an upper-bound heuristic and store that solution as the Best-So-Far

Add all of the nodes in the heuristic solution to a stack

Pop the last node in the stack

If the node is terminal, evaluate it and compare it with B

Store better solutions, discard all others

If the node is non-terminal, *branch* on that node

Calculate lower-*bound*s on the child nodes

If the lower-*bound* $\geq$ B, we discard that node

otherwise it goes into the queue

A*

Use infinity and store that as the Best-So-Far

Add the root node to a heap

Pop the last node in the heap

If the node is terminal, evaluate it and compare it with B

Store better solutions, discard all others

If the node is non-terminal, *branch* on that node

Calculate lower-*bound*s on the child nodes

If the lower-*bound* $\geq$ B, we discard that node

otherwise it goes into the queue

COMPARISON

- Both algorithms can be admissible (capable of guaranteeing optimality)
- A* will need to hold all visited nodes, and all frontier nodes and search each list on every node expansion
- A* will visit all nodes that are closer than the goal, while B&B *may* visit nodes that are further than the optimal solution with growth relative to the following equation:

$$BranchFactor^{(HeuristicLength-OptimalLength)}$$

WHY NOT JUST USE A*?

- Time constraints
- Memory Constraints
- Practice makes perfect

SUMMARY

- Can guarantee an optimal solution
- Searches the space and doesn't need to reach completion for a valid solution
- Lower memory requirements than A*