NUS | Computing

National University of Singapore

CS3243 Tutorial 4

Local Search

Gu Zhenhao

February 15, 2023

Notes

- **Project 1** due this week.
- **Midterm exam** will be on Monday, Week 7.
  - You will get your feedback for this tutorial assignment during the recess week, through telegram.

Proving Admissibility

How to prove if a heuristic is good?

Building Heuristics

Idea 1: Find the ***optimal cost*** of a relaxed (simplified) problem.

Strategies:

- Relax the rules.
- Solve a sub-problem.
- Learn the heuristics.

Idea 2: Combine several heuristics together.

If h_1, h_2 admissible/consistent, then $\max\{h_1, h_2\}$ is also admissible/consistent and dominates h_1 and h_2.

Proving Properties of Heuristics

- **To prove admissibility**:
  - show that **for all** state n, we have $h(n) \leq h^*(n)$, or
  - show that h is the **optimal** cost to a goal of a **relaxed problem**, or
  - show that h is dominated by another **admissible heuristic** h'.
- **To disprove admissibility**:
  - find **just one** state s, such that $h(s) > h^*(s)$.

Similar strategy applies to consistency, dominance, etc.

Tutorial 3, Problem 4

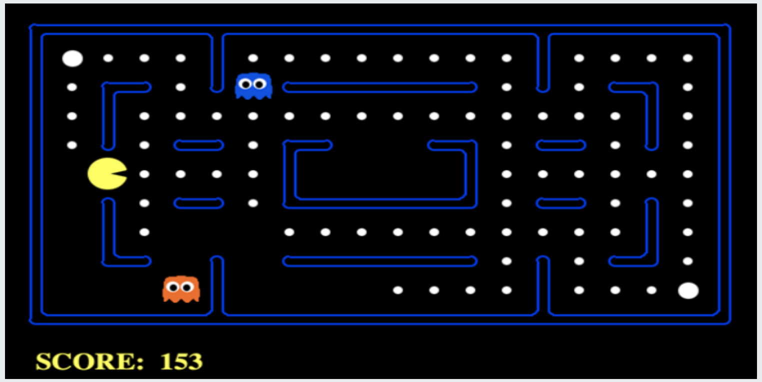


Figure. Pac-man.

- **State**: position of pac-man and positions remaining pallets.
  - **Initial state**: starting position of pac-man and grid entirely filled with pallets.
  - **goal state**: no pallets left on field.
- **Action**: move up/down/left/right.
  - **Action cost**: 1.
- **Transition model**: returns the field after the pac-man moved and consumed pallet.

Tutorial 3, Problem 4

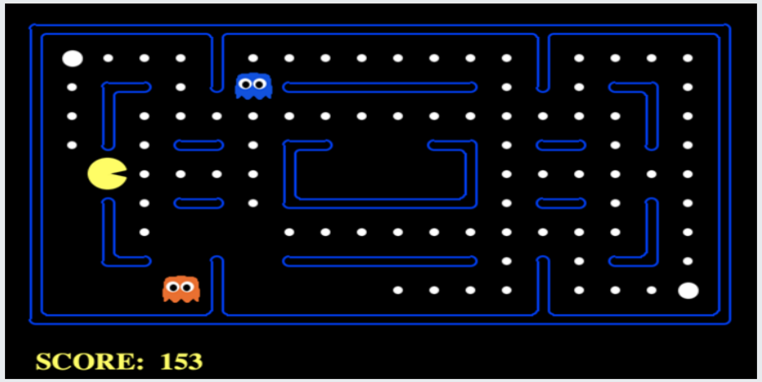


Figure. Pac-man.

- $h_1(n) = \#$ remaining pallets,

Admissible! This is a **relaxation of rule** "Pac-man can only move up/down/left/right".

Proof. **Observation**: an action reduces $h_1(n)$ by either 0 or 1.

$\Rightarrow$ we need at least $h_1(n)$ steps to get from state n to a goal.

$\Rightarrow h_1(n) \leq h^*(n)$ for all n.

Tutorial 3, Problem 4

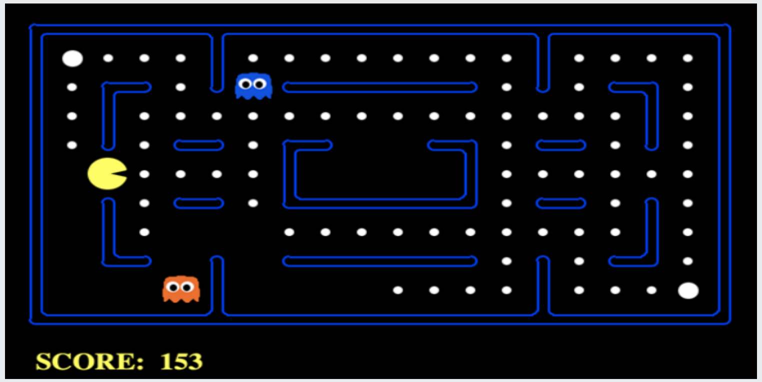
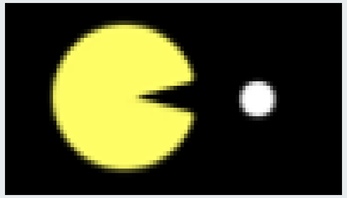


Figure. Pac-man.

- $h_2(n) = \#$ remaining pallets $+$ Manhattan distance to closest pallet,

Not admissible! Be careful with addition of two heuristics...

Proof. Consider a state s where there's only one remaining pallet in the neighbourhood of Pac-man.



$h_2(s) = 1 + 1 = 2$, while $h^*(s) = 1$.

Tutorial 3, Problem 4

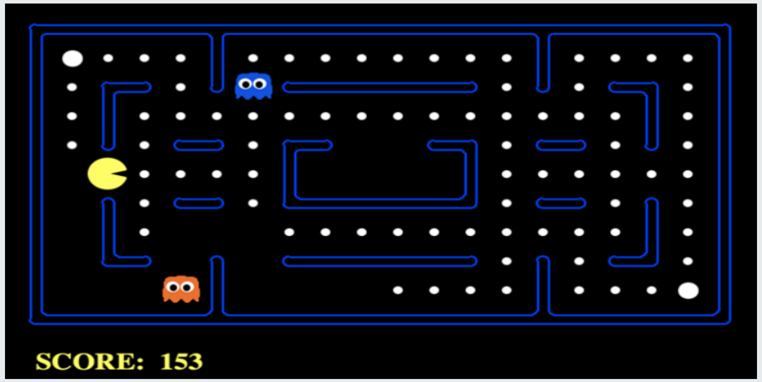


Figure. Pac-man.

- $h_3(n)$ = maximum Manhattan distance to a remaining pallet,

Admissible! This is both a relaxation of rule "cannot pass through obstacles" and a **relaxation of goal** "Pac-man must consume all pallets".

Proof. We need at least $h_3(n)$ steps to consume the farthest pallet.

$\Rightarrow$ At least $h_3(n)$ steps to get to a goal.

$\Rightarrow$ $h_3(n) \leq h^*(n)$ for all n.

Tutorial 3, Problem 4

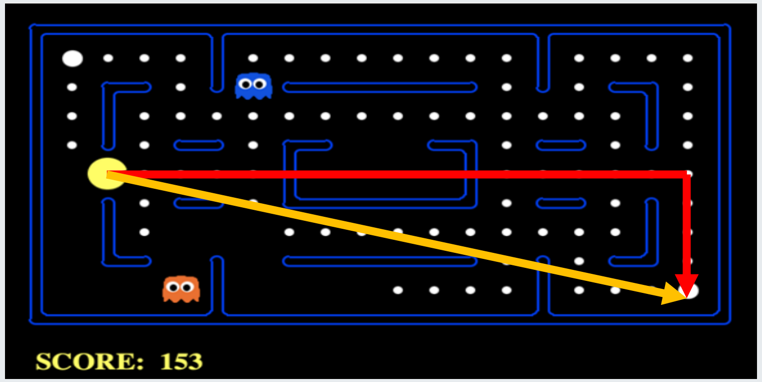


Figure. Pac-man.

- $h_4(n)$ = average Euclidean distance to all pallets.

Admissible! This is again both a relaxation of rule and a relaxation of goal.

Proof. $h_4(n) \leq$ max Euclidean distance to all pallets (max is at least average)

$\leq$ max Manhattan distance to all pallets (by triangle inequality)

$\leq h_3(n)$ for all n. h_3 dominates h_4.

Local Search

What if we only care about reaching a goal?

Local Search Overview

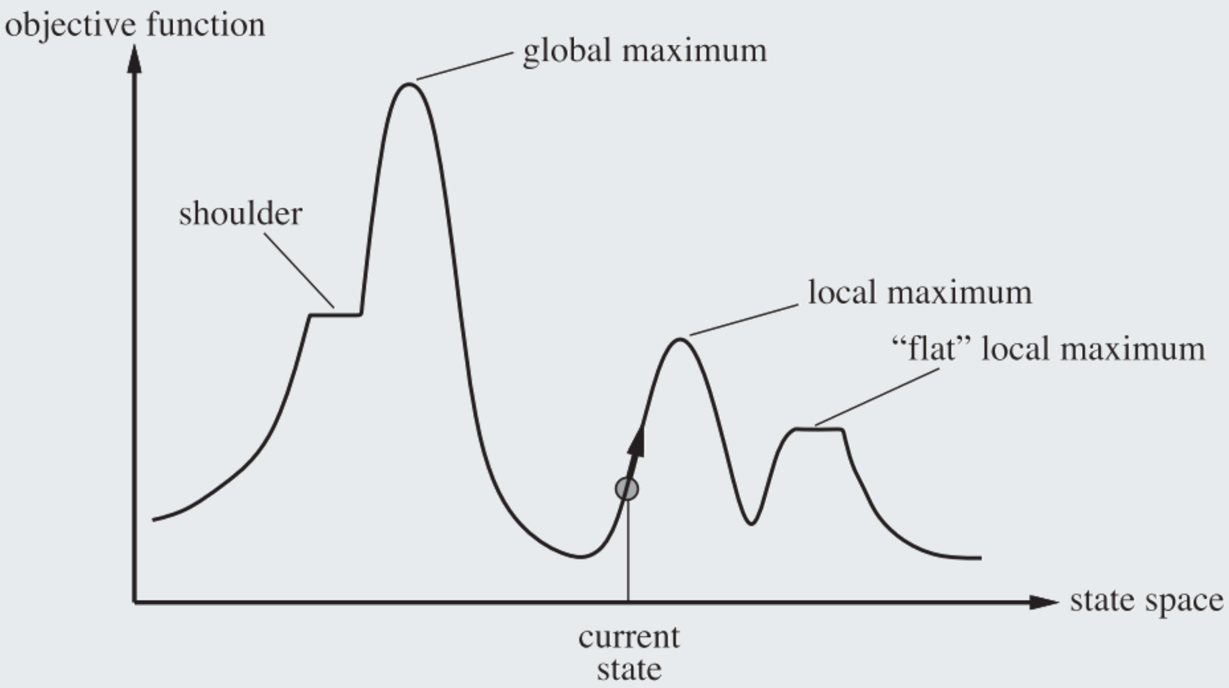


Figure. Maximizing a function.

Target problem:

- Very large state space,
- can be formulated as an optimization problem (find solution with maximum f).

General Strategy:

- move from one solution to (neighbouring) solutions.
- hope to find the optimal one.

Hill Climbing Algorithm

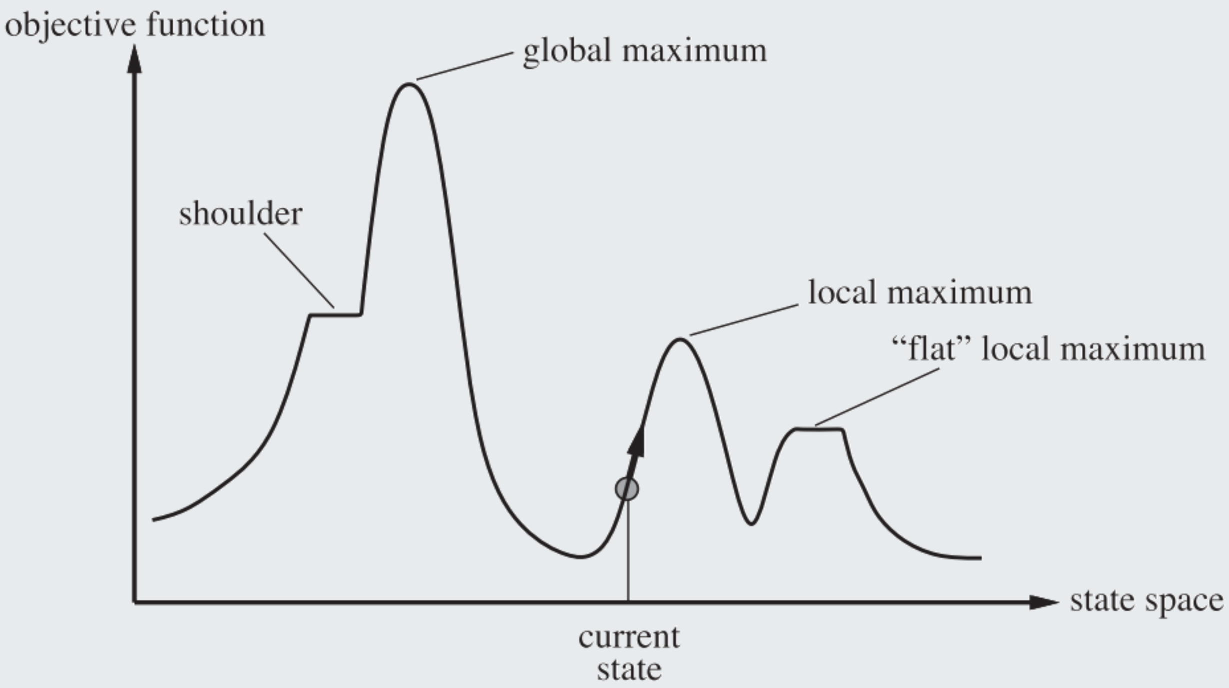


Figure. Hill Climbing Algorithm.

- **Intuition**: always look for the best neighbour.
- **Pros**:
  - Possibly fast for well defined f.
  - Memory-saving.
- **Cons**:
  - Not complete.
  - Easily stuck in local maximum of f.

Some Variants

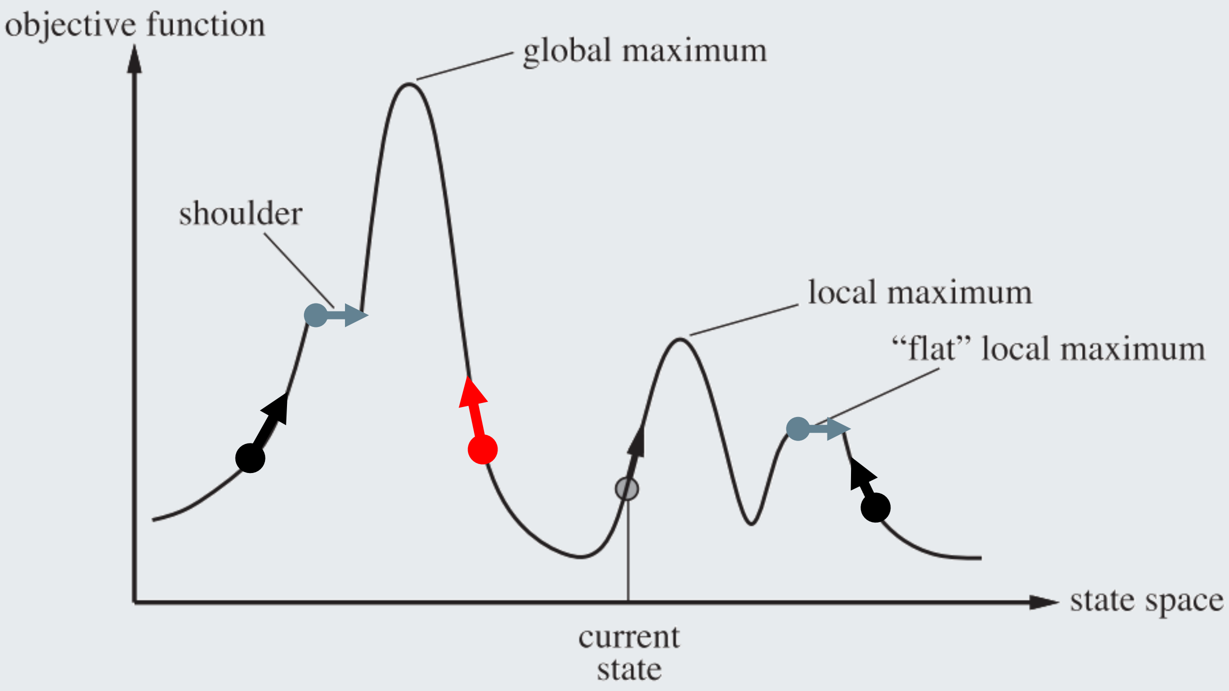


Figure. Hill Climbing Algorithm.

- **Stochastic hill climbing**.
  randomly choose among all uphill moves.
- **Random-restart hill climbing**.
  try multiple initial states.
- **Sideway moves.**
  Continue search when there is a neighbor with equal f value.

Other Popular Variants

General idea: Allow more diversity in paths explored.

- ***Simulated annealing****.
  Combine hill climbing and random walk (choose next successor randomly).
- ***Gradient descent****.
  Choose the neighbour with the steepest increase in f, based on ∇f.
- ***Local (stochastic) beam search***.
  Keep the best (random) k successors and explores more paths.
- ***Genetic algorithm****.

** Not in the scope of CS3243.*

Building *Local* Search Problems

Intuition: we no longer care about cost and the path, but care about how to get closer to the goal.

1. **What should a node contain?** Determine state representation s_i.
2. **How to find a node's neighbours?** Identify the rules and define the valid actions A that (potentially) get you closer to the goal.
3. **Where should we start?** Generate a random initial state s_0 that **satisfies the rules**.
4. **How to judge if a neighbour is good?** Define an objective function $f(s_i) :=$ how close s_i is to the **global best state**.

Problem 1

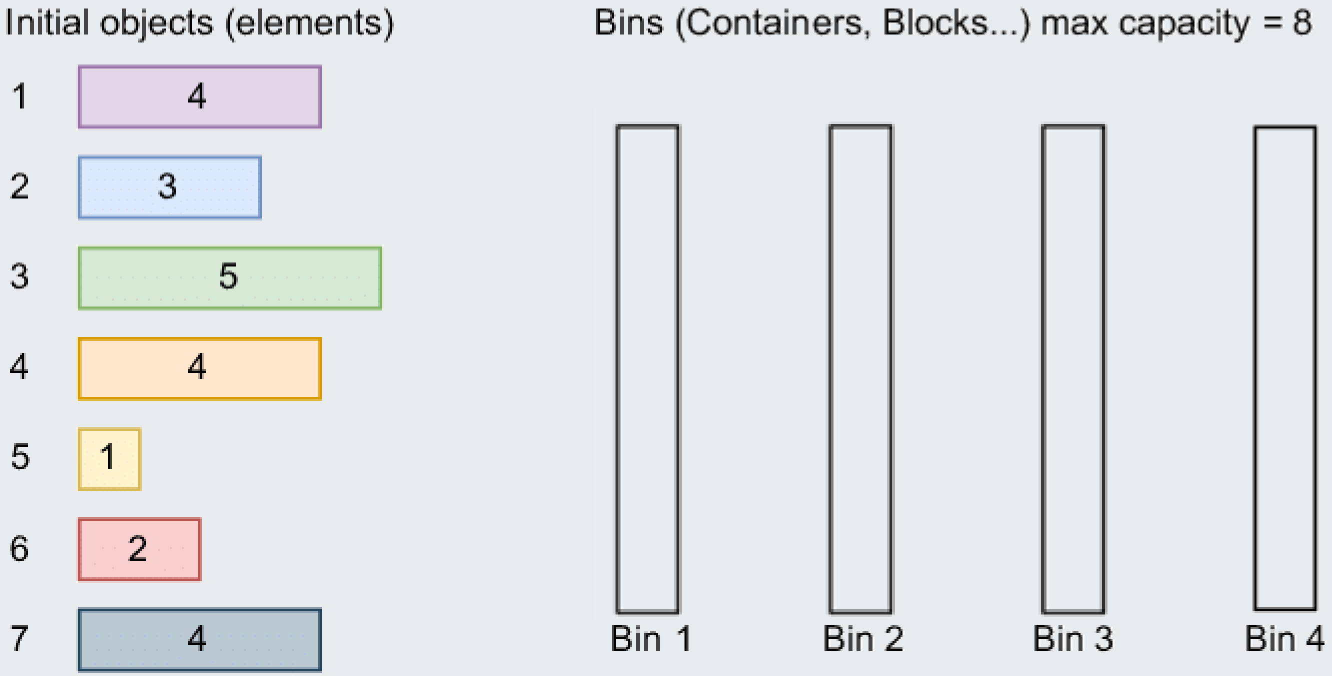


Figure. The classical "bin packing" problem.

Source: An efficient cloudlet scheduling via bin packing in cloud computing

Given:

- n items $a_1, \ldots, a_n$, each with size $s(a_i) > 0$.
- m boxes $b_1, \ldots, b_m$, each with capacity $c(b_i) > 0$.

Goal:

Pack all items into as few boxes as possible.

Problem 1

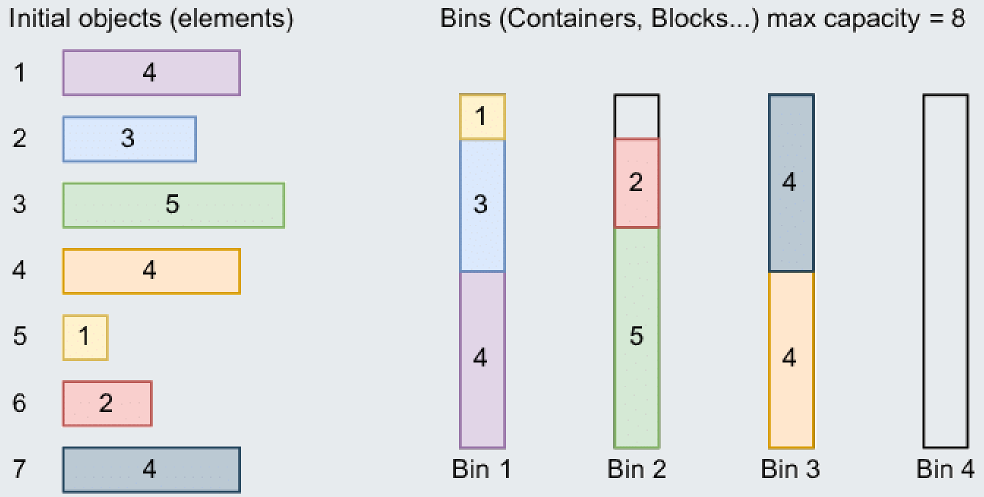


Figure. The classical "bin packing" problem.

State: Boolean matrix,

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ is in box } b_j \\ 0 & \text{otherwise} \end{cases}$$

Rules:

- Every item is in a box: $\sum_{j=1}^{m} x_{i,j} = 1$ for all i.
- Capacity is not exceeded: $\sum_{i=1}^{n} x_{i,j} s(a_i) \leq c(b_j)$ for all j.

Objective: Number of used boxes minimized, i.e. maximize #unused boxes.

Problem 1

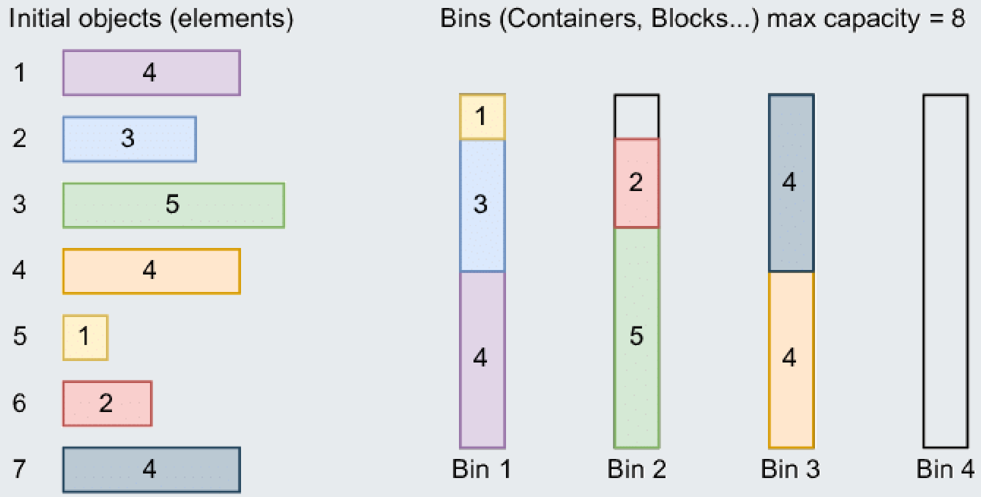


Figure. The classical "bin packing" problem.

State: Boolean matrix,

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ is in box } b_j \\ 0 & \text{otherwise} \end{cases}$$

Initial state:

random approach

- randomly assign items to boxes, or
- the *first-fit strategy*: sort the box in decreasing capacity, assign each item to the first box that fits it.

Greedy approach

Problem 1

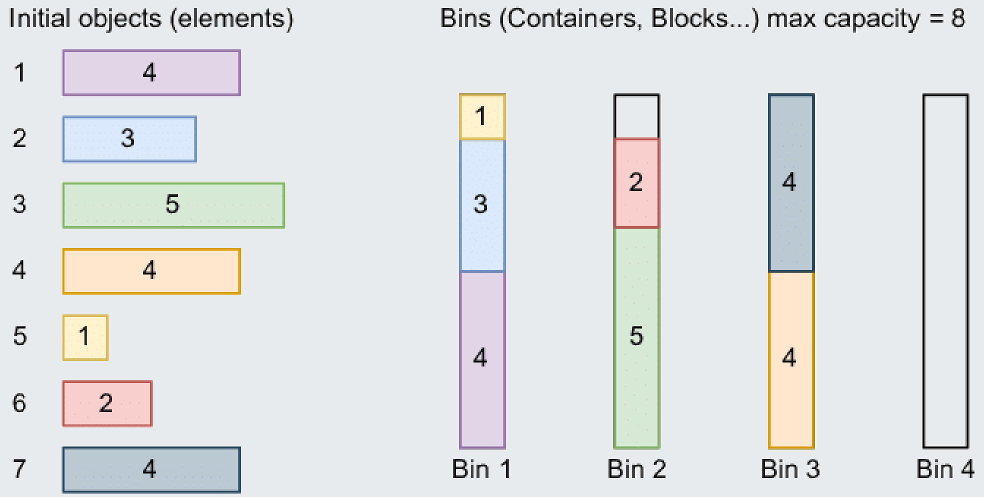


Figure. The classical "bin packing" problem.

Question: which one would you choose?

State: Boolean matrix,

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ is in box } b_j \\ 0 & \text{otherwise} \end{cases}$$

Action:

- Move an assigned item a_i from box b_j to $b_{j'}$.
- Swap two assigned items a_i, a_j.
- Remove all items in box b_j and fill the items into other non-empty boxes using *first-fit strategy*.

Problem 1

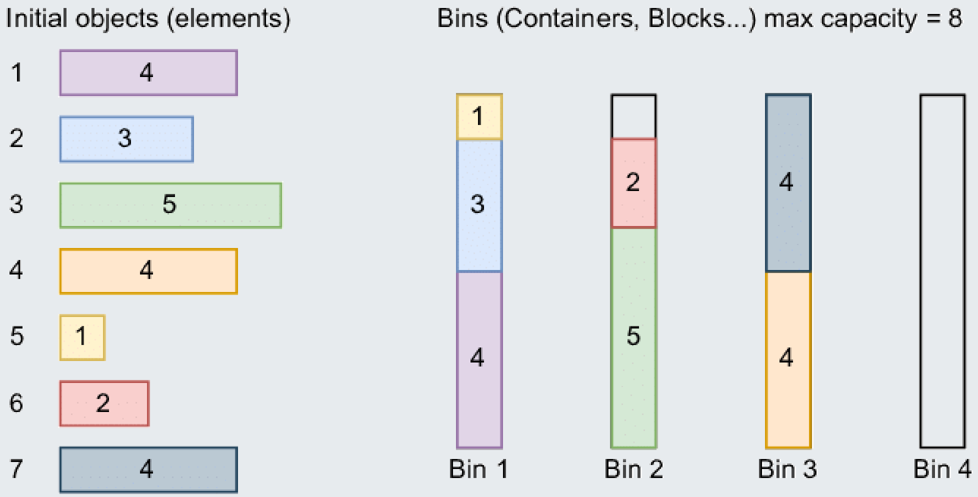


Figure. The classical "bin packing" problem.

State: Boolean matrix,

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ is in box } b_j \\ 0 & \text{otherwise} \end{cases}$$

Action: Remove all items in box b_j and fill the items into other non-empty boxes using *first-fit strategy*.

Objective function: $f(n) = \#\text{unused boxes}$.

Question: what is the potential problem of this f?

Usually, $f(n') = f(n) - 1$ for all valid successor n' of n.

Problem 1

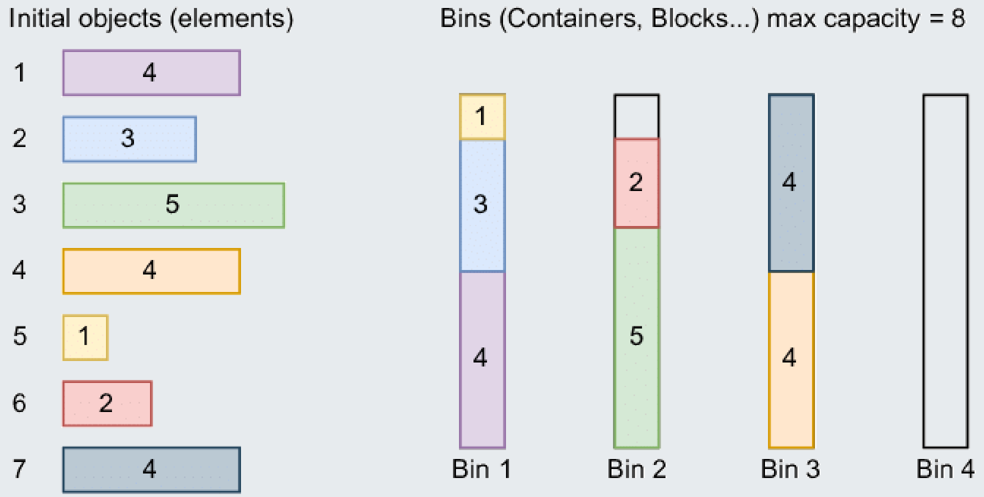


Figure. The classical "bin packing" problem.

State: Boolean matrix,

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ is in box } b_j \\ 0 & \text{otherwise} \end{cases}$$

Action: Remove all items in box b_j and fill the items into other non-empty boxes using *first-fit strategy*.

Objective function: $f(n) =$ how full the used boxes are.

Maximize $f(n) \Rightarrow$ make full use of boxes $\Rightarrow$ minimize number of boxes used.

Problem 1

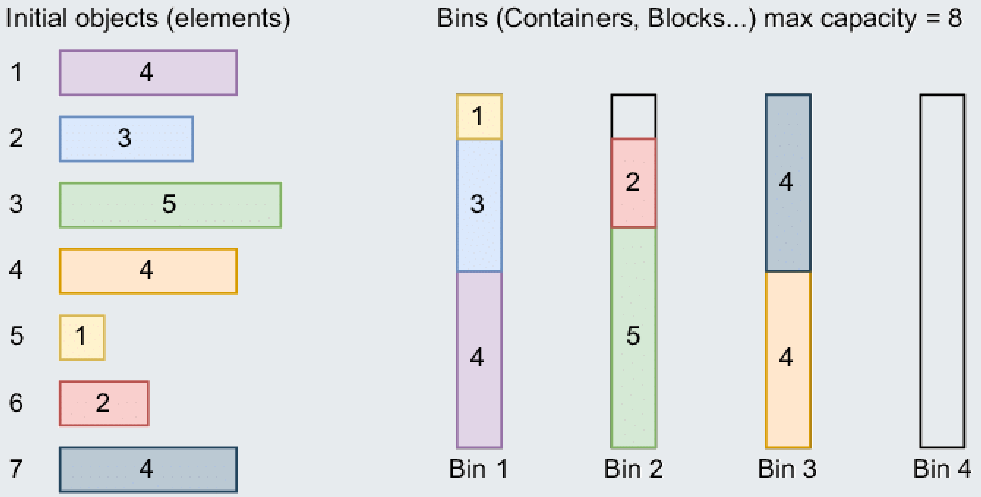


Figure. The classical "bin packing" problem.

State: Boolean matrix,

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ is in box } b_j \\ 0 & \text{otherwise} \end{cases}$$

Action: Remove all items in box b_j and fill the items into other non-empty boxes using *first-fit strategy*.

Objective function: $f(n) = \sum_j \left(\frac{\sum_i x_{i,j} s(a_i)}{c(b_j)} \right)^2$. [Hyde et. al., '09]

More full boxes, higher $f(n)$.

Problem 2

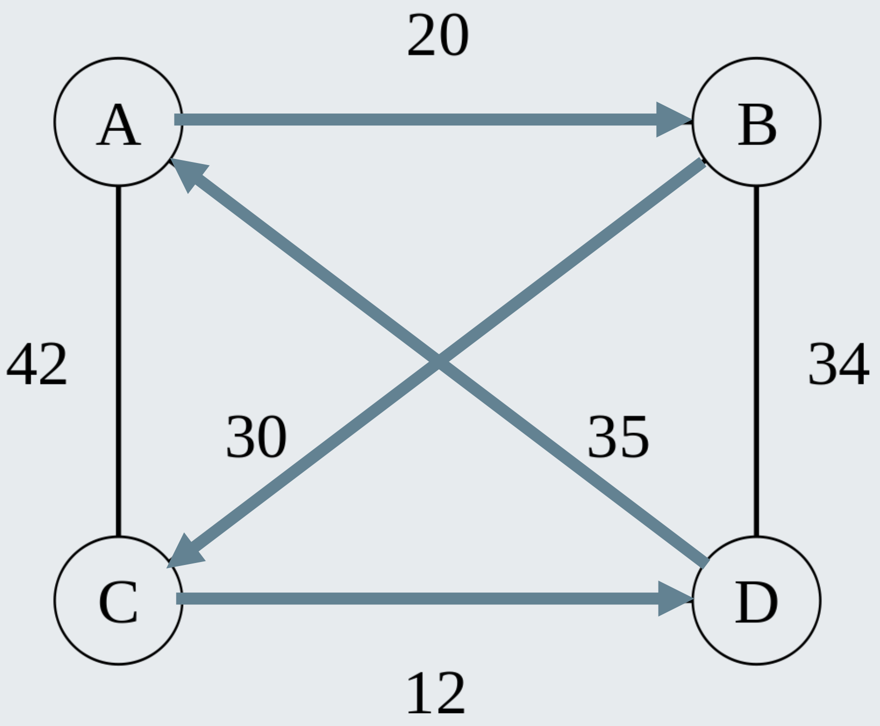


Figure. The "traveling salesman" problem (TSP).
Source: Wikipedia.

Given:

An undirected, weighted and complete graph.

Goal:

Find the shortest possible route that visits each vertex exactly once and returns to the starting vertex.

Problem 2.a, b

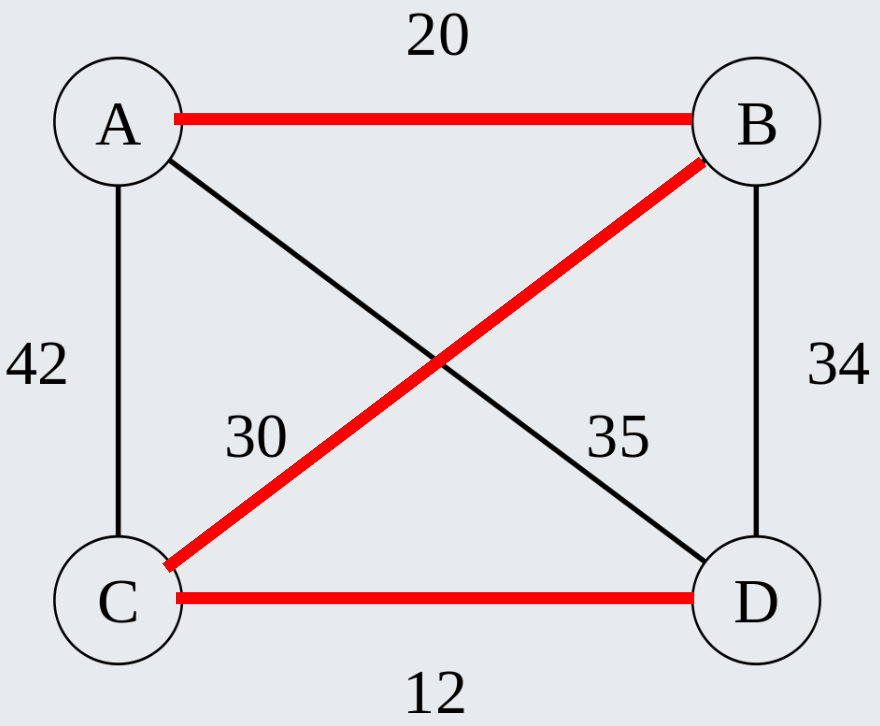


Figure. The minimum spanning tree (MST) of the graph. Source: Wikipedia.

Given:

An undirected, weighted and complete graph.

Goal: Find a set of edges that

- ~~is a cycle~~.
- connects all vertices.
- ~~visits each vertex only once.~~
- has the smallest total edge weights.

$h(n) :=$ cost of MST!

Problem 2.c

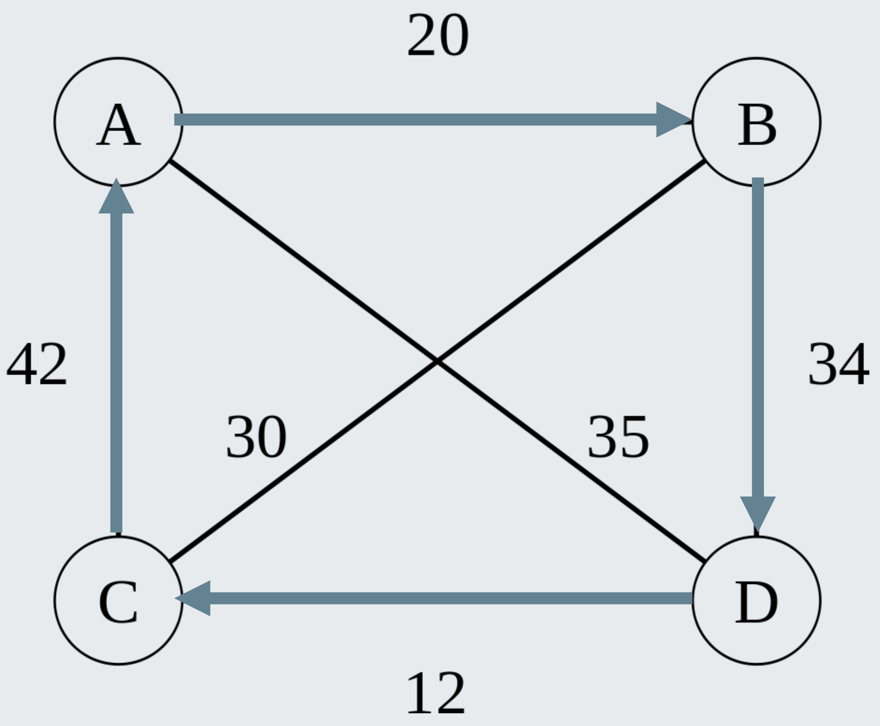


Figure. The "traveling salesman" problem (TSP).
Source: Wikipedia.

State: A path in the graph.

Rules:

- The path is a cycle that starts at A and ends at A, and

The choice is arbitrary.

- visits each node once.

Objective:

- Minimize the total cost of the path.

Problem 2.c

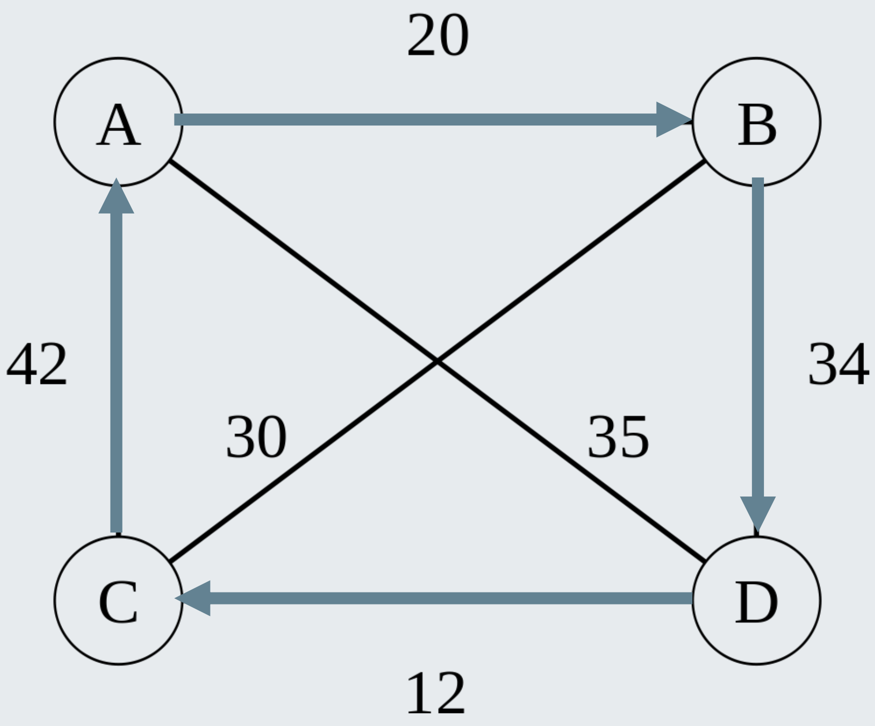


Figure. The "traveling salesman" problem (TSP).
Source: Wikipedia.

State: A path in the graph.

Initial state:

- Randomly choose a cycle in graph, or

e.g. $A \to B \to D \to C \to A$.

Problem 2.c

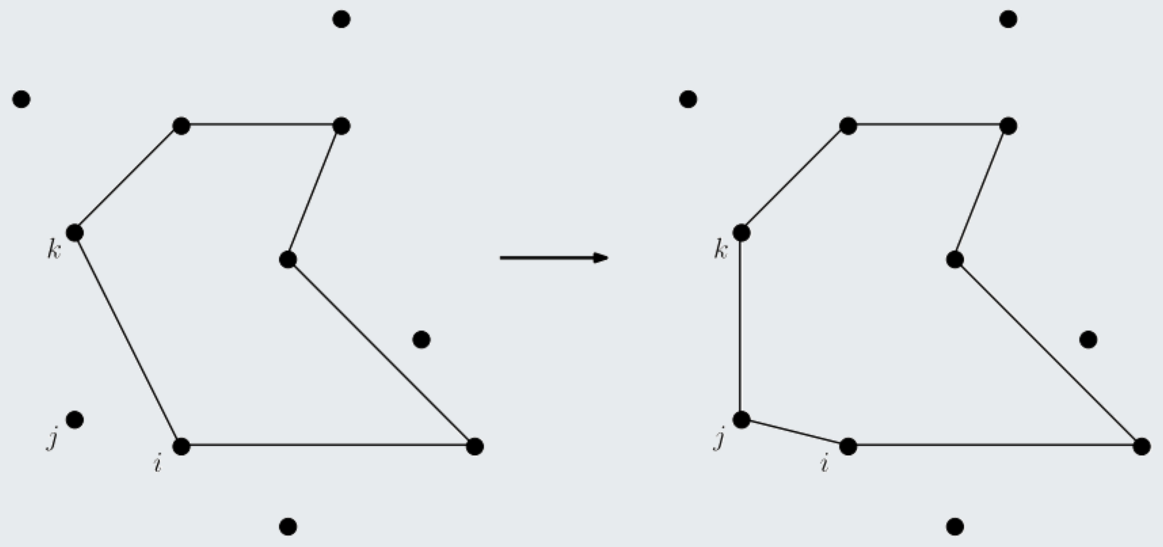


Figure. Greedy way of constructing solution to TSP.
Source: The Design of Approximation Algorithms.

State: A path in the graph.

Initial state:

- Randomly choose a cycle in graph, or
- Construct the cycle* by
    - starting with a pair of closest vertices,
    - adding other vertices one by one.
      Each time we choose the vertex that is closest to a vertex in the cycle.

** Just for fun! No need to understand this :-)*

Problem 2.c

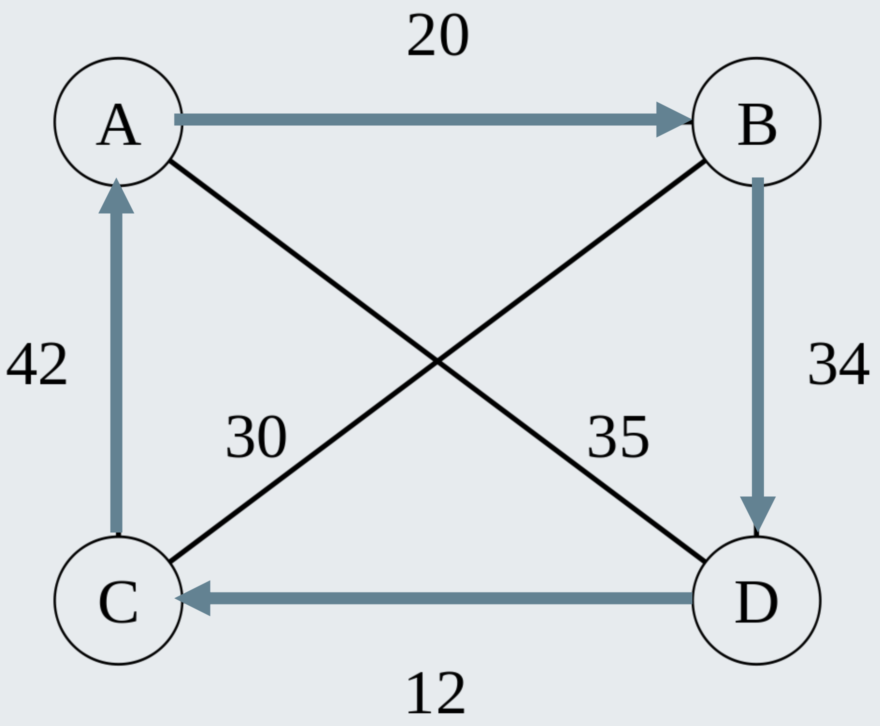


Figure. The "traveling salesman" problem (TSP).
Source: Wikipedia.

State: A path in the graph.
e.g. $A \to B \to D \to C \to A$.

Action: alter the path (by swapping the order of visiting vertices).

- swap(B, D), $f(n) = -35 - 34 - 30 - 42 = -141$
- swap(B, C), $f(n) = -42 - 12 - 34 - 20 = -108$
- swap(D, C). $f(n) = -20 - 30 - 12 - 35 = -97$ ✓

Objective function:

$f(n) = -$ total path cost.

Problem 3

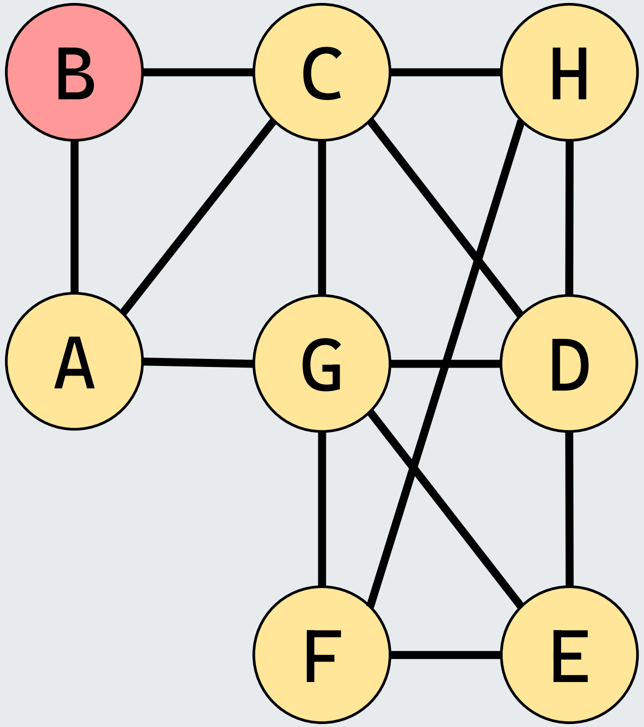


Figure. Initial state.

State: color assignment of each node.

Rules: each node is assigned a color in {RED, YELLOW, BLUE}.

Goal: No two adjacent vertices are assigned the same color.

Problem 3.a

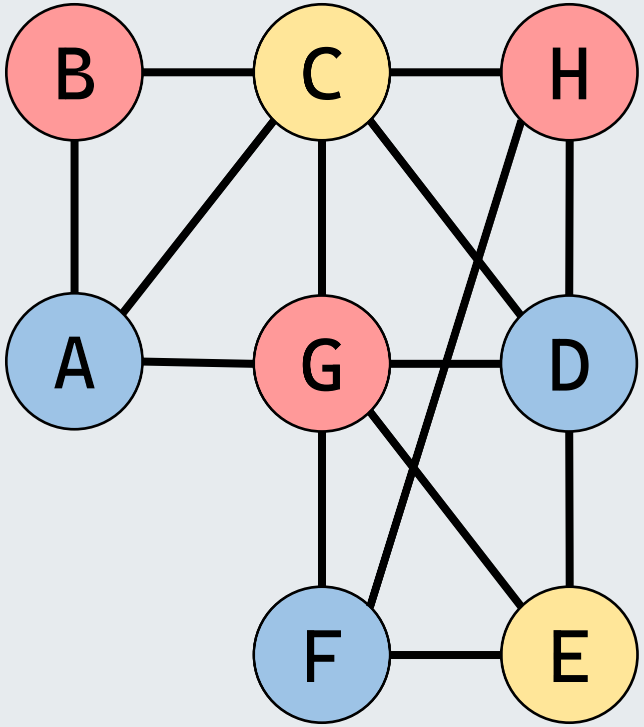


Figure. A goal state.

State: color assignment of each node.

Rules: each node is assigned a color in {RED, YELLOW, BLUE}.

Goal: No two adjacent vertices are assigned the same color.

Problem 3.b

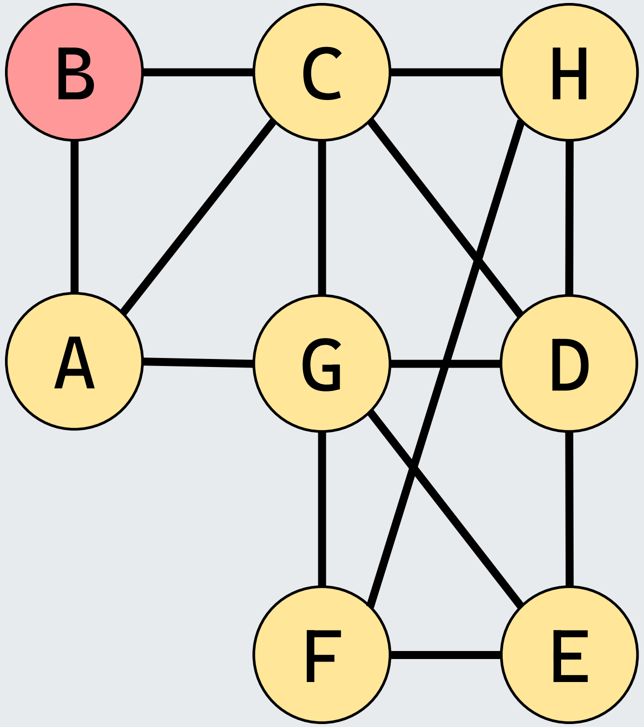


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: $f(n) = \#$pairs of adjacent vertices that have different color.

	$f(n)$		$f(n)$
$B \rightarrow$ YELLOW		$B \rightarrow$ BLUE	
$C \rightarrow$ RED		$C \rightarrow$ BLUE	
$H \rightarrow$ RED		$H \rightarrow$ BLUE	
$A \rightarrow$ RED		$A \rightarrow$ BLUE	
$G \rightarrow$ RED		$G \rightarrow$ BLUE	
$D \rightarrow$ RED		$D \rightarrow$ BLUE	
$F \rightarrow$ RED		$F \rightarrow$ BLUE	
$E \rightarrow$ RED		$E \rightarrow$ BLUE	

Problem 3.b

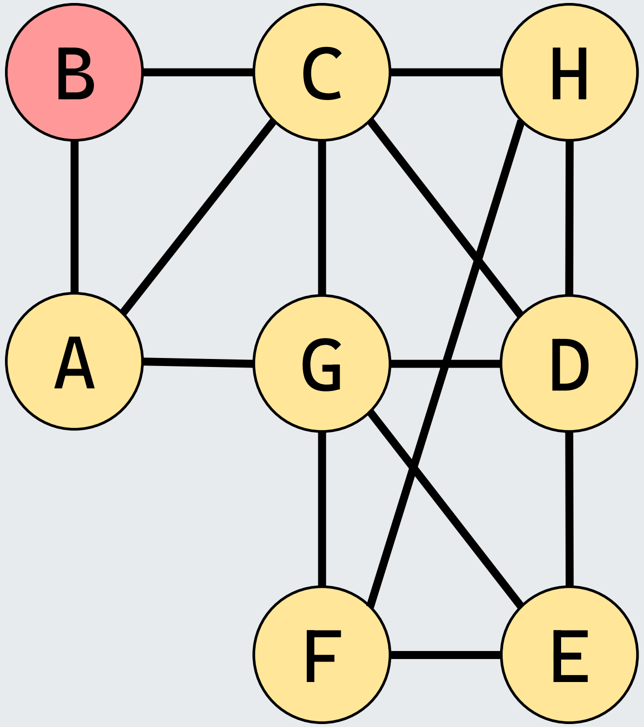


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: $f(n) = \#$pairs of adjacent vertices that have different color.

	$f(n)$		$f(n)$
$B \rightarrow$ YELLOW	0	$B \rightarrow$ BLUE	2
$C \rightarrow$ RED	5	$C \rightarrow$ BLUE	6
$H \rightarrow$ RED	5	$H \rightarrow$ BLUE	5
$A \rightarrow$ RED	3	$A \rightarrow$ BLUE	4
$G \rightarrow$ RED	7	$G \rightarrow$ BLUE	7
$D \rightarrow$ RED	6	$D \rightarrow$ BLUE	6
$F \rightarrow$ RED	5	$F \rightarrow$ BLUE	5
$E \rightarrow$ RED	5	$E \rightarrow$ BLUE	5

Problem 3.b

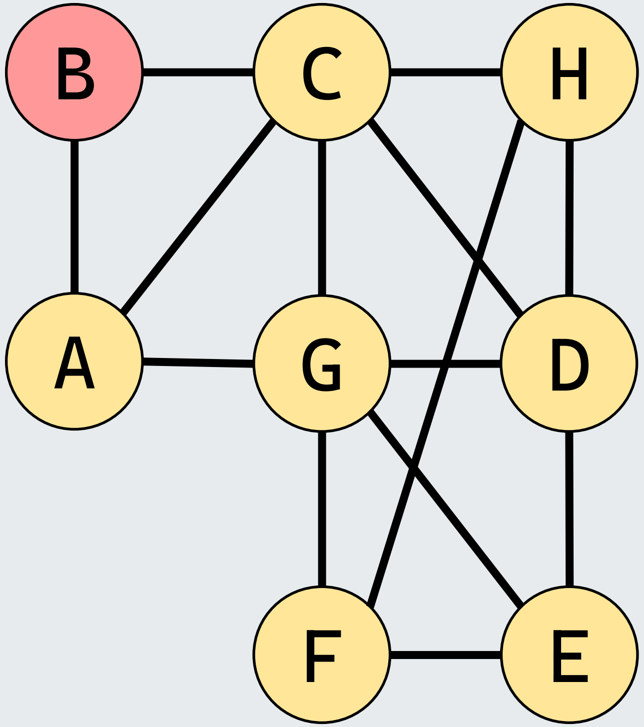


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: $f(n) = \#$pairs of adjacent vertices that have different color.

- Among all actions, find the one that resolves the most collisions!
- Changing G to RED or BLUE increases $f(n)$ by 5!

Problem 3.b

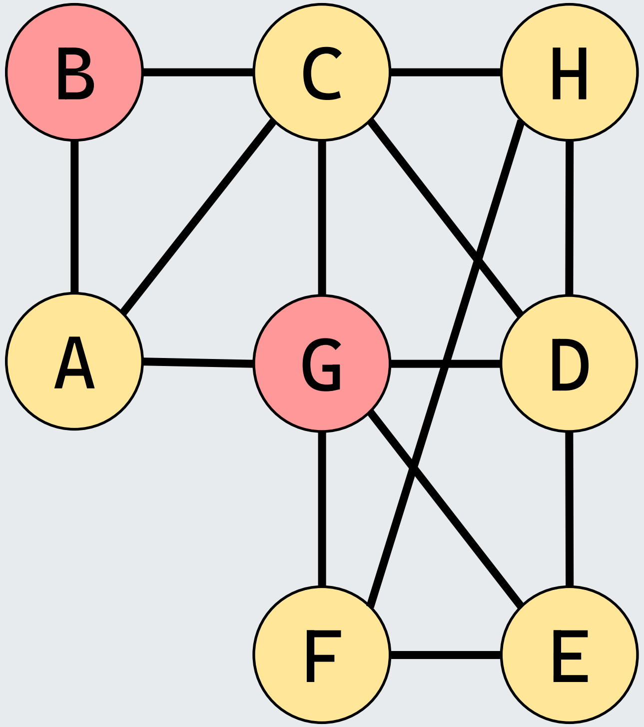


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: #pairs of adjacent vertices that have different color (collision).

- Among all actions, find the one that resolves the most collisions!
- Changing A to BLUE increases $f(n)$ by 3!

Problem 3.b

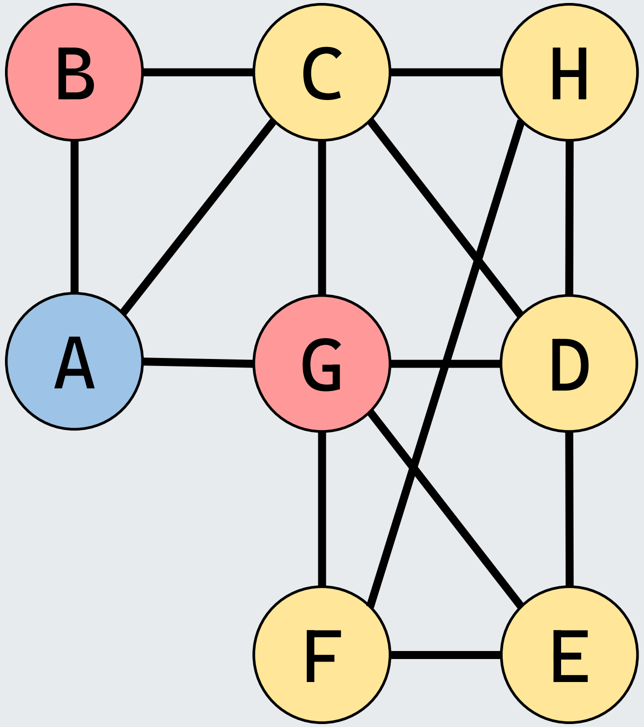


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: #pairs of adjacent vertices that have different color (collision).

- Among all actions, find the one that resolves the most collisions!
- Changing D to BLUE increases $f(n)$ by 3!

Problem 3.b

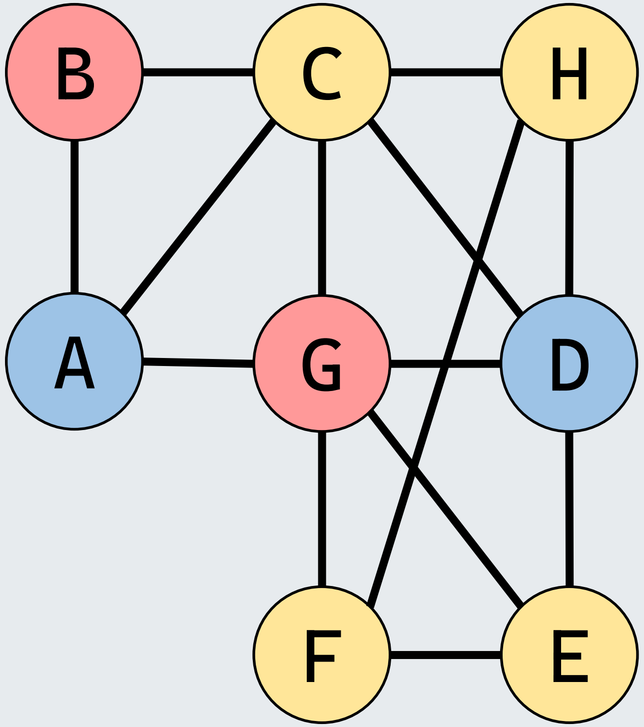


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: #pairs of adjacent vertices that have different color (collision).

- Among all actions, find the one that resolves the most collisions!
- Changing H to RED increases $f(n)$ by 2!

Problem 3.b

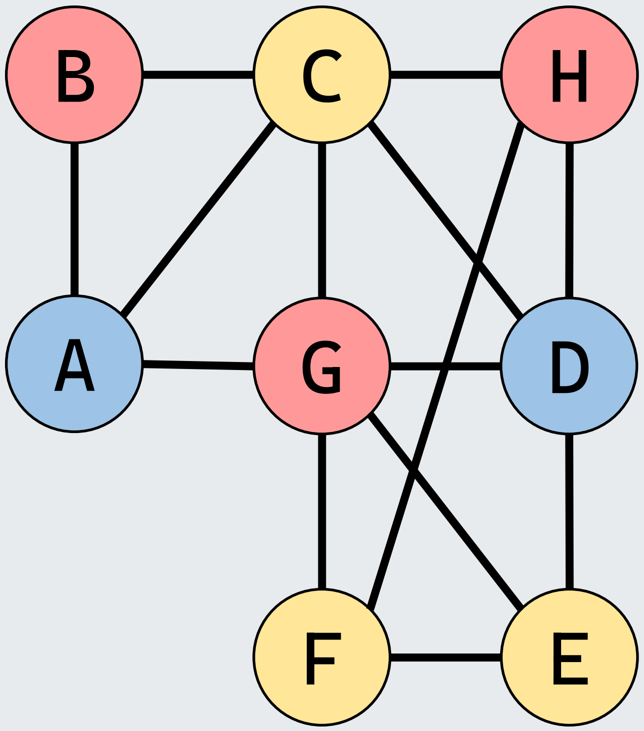


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: #pairs of adjacent vertices that have different color (collision).

- Among all actions, find the one that resolves the most collisions!
- Changing F to BLUE increases $f(n)$ by 1!

Problem 3.b

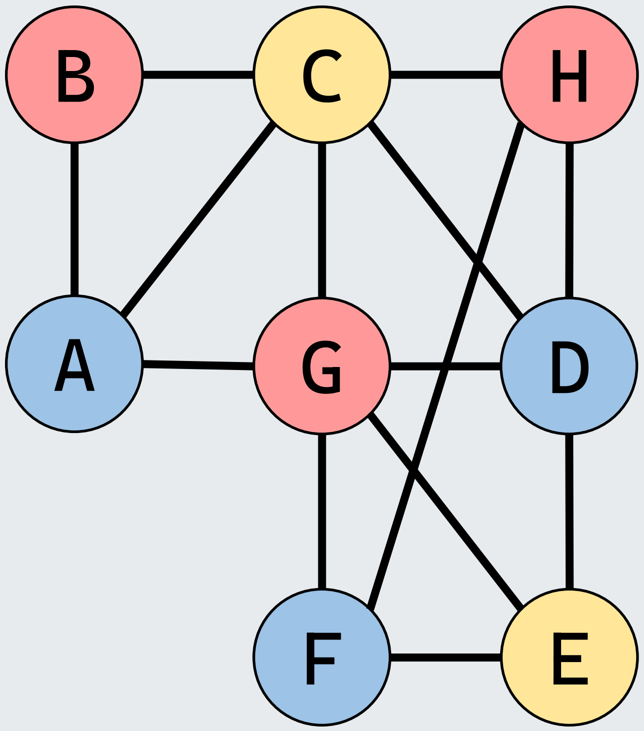


Figure. A goal state.

State: color assignment of each node.

Action: Change the color of a single vertex.

Objective function: #pairs of adjacent vertices that have different color (collision).

- Among all actions, find the one that resolves the most collisions!

All collisions resolved! :-D

End of File

Thank you very much for your attention!

References

- D. Ler, “Local Search: Goal Versus Path Search”, 2023. [Online].
- Matthew Hyde, Gabriela Ochoa, T Curtois, and JA Vazquez-Rodrguez. “A hyflex module for the one dimensional bin-packing problem”. School of Computer Science, University of Nottingham, Technical Report, 2009
- S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," 3rd ed., Prentice Hall, 2010.