# CS-540: Intro to Artificial Intelligence

*Informed Search II + Local Search*

Lecturer: Erin Winter

# A-Search

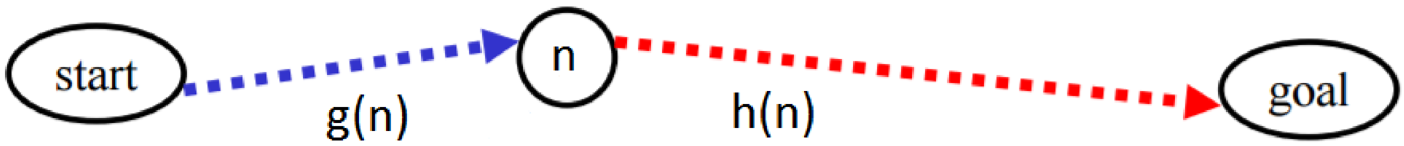
- Use evaluation function

$$f(n) = g(n) + h(n)$$

where ***g*(*n*)** is minimum cost path from start to current node *n* (as defined in Dijkstra's)

- Nodes in *the frontier* are sorted by increasing **f(n)** value

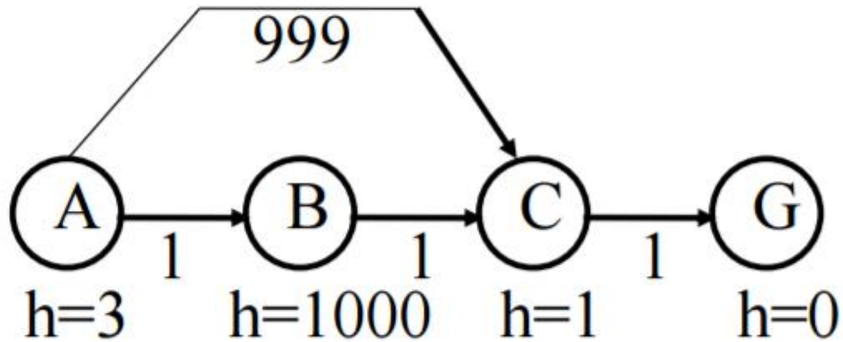
- Think of it as a "first half" g(n) + "second half" h(n) computation



# A-Search alone doesn't suffice

- With an arbitrarily bad heuristic function, A-Search isn't **optimal** or **complete**.

A-Search: A,C,G
Cost: 100
Optimal: A,B,C,G
Cost: 3




- We can do better by constraining the heuristic to be **admissible.**

- An admissible heuristic h(n) for all nodes n will never **overestimate** the *true minimum cost path from n to the goal.*

A-Search + Admissible (or Consistent) Heuristic =

# A* Search

effective and used in practice

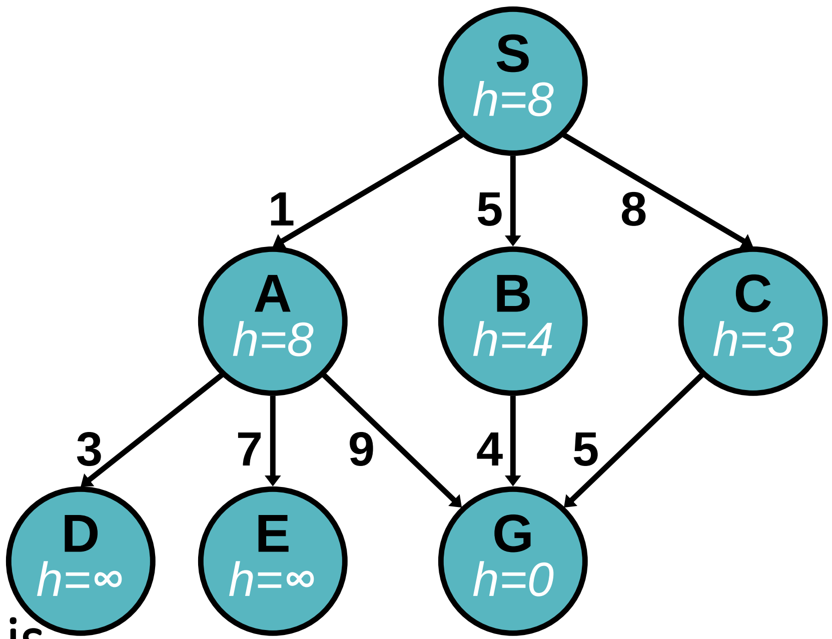
# A* Search Practice

Is *h* is admissible and/or consistent? It is consistent.

$f(n) = g(n) + h(n)$

\# of nodes tested: 0, expanded: 0

| expnd. node | Frontier |
|---|---|
| | {S:0+8} |




To test for admissibility, check that h(n) never overestimates the true cost to the goal.

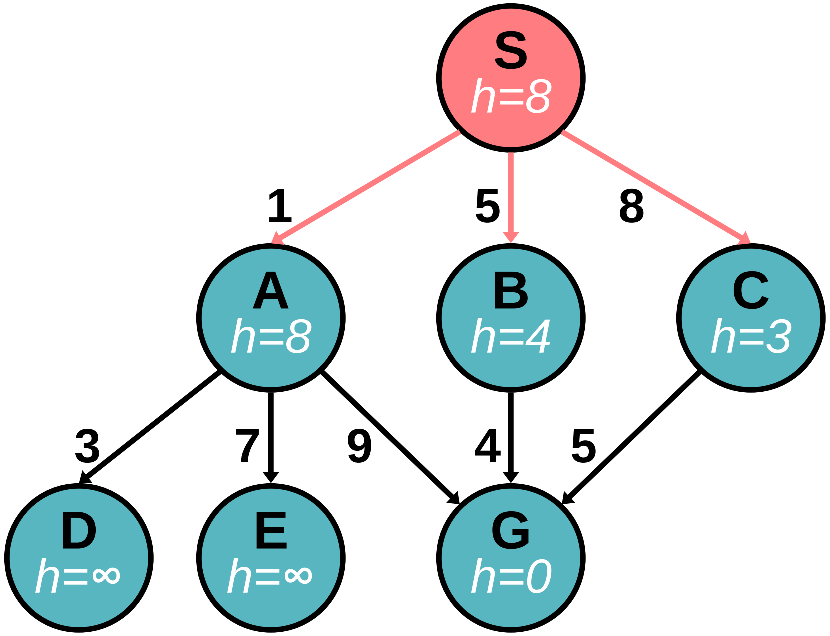
To check for consistency, **check h(n) is consistent AND h(*n*) ≤ *c*(*n*, *n*′) + *h*(*n*′) for all successor nodes n′.**

# A* Search

$$f(n) = g(n) + h(n)$$

# of nodes tested: 1, expanded: 1

| expnd. node | Frontier |
|---|---|
| | {S:8} |
| S not goal | {A:1+8,B:5+4,C:8+3} |

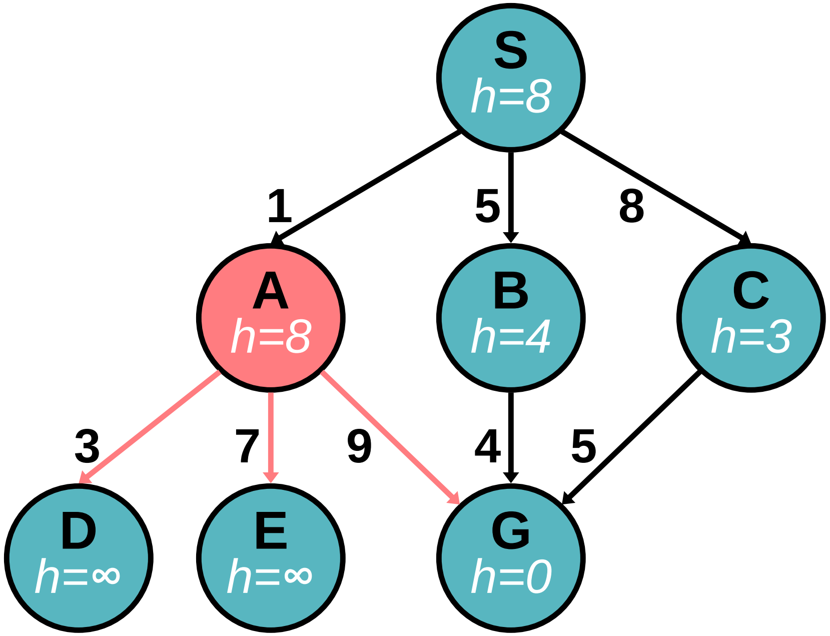


# A* Search

$f(n) = g(n) + h(n)$

# of nodes tested: 2, expanded: 2

| **expnd. node** | **Frontier** |
|---|---|
| | {S:8} |
| S | {A:9,B:9,C:11} |
| A not goal | {B:9,G:1+9+0,C:11, D:1+3+∞,E:1+7+∞} |

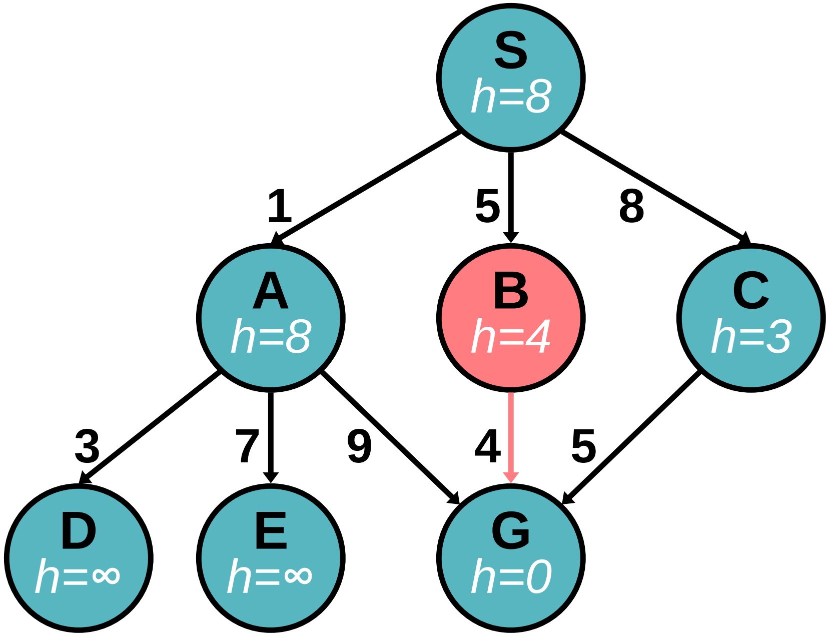


# A* Search

$$f(n) = g(n) + h(n)$$

# of nodes tested: 3, expanded: 3

| expnd. node | Frontier |
|---|---|
| | {S:8} |
| S | {A:9,B:9,C:11} |
| A | {B:9,G:10,C:11,D:∞,E:∞} |
| B not goal | {G:5+4+0,~~G:10~~,C:11, D:∞,E:∞} replace |

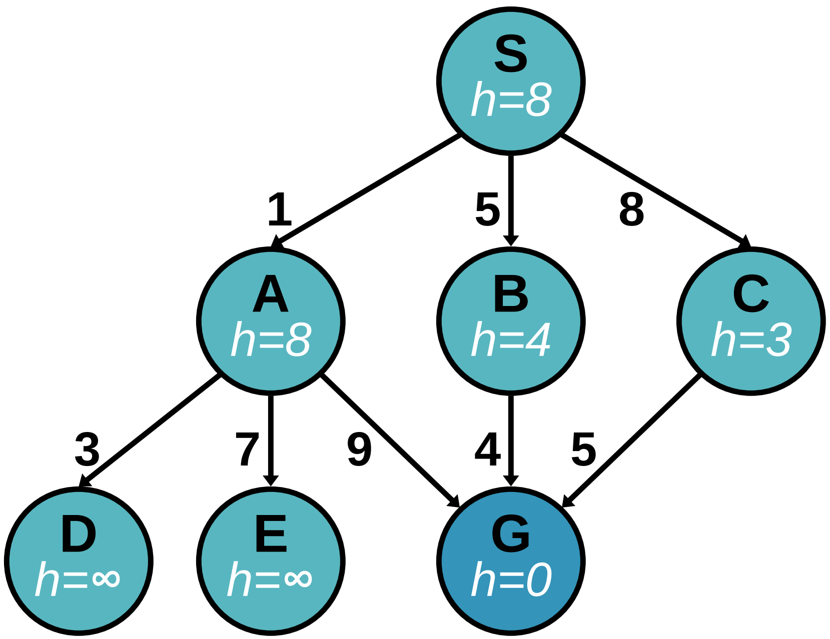


# A* Search

$f(n) = g(n) + h(n)$

# of nodes tested: 4, expanded: 3

| expnd. node | Frontier |
|---|---|
| | {S:8} |
| S | {A:9,B:9,C:11} |
| A | {B:9,G:10,C:11,D:∞,E:∞} |
| B | {G:9,C:11,D:∞,E:∞} |
| G goal | {C:11,D:∞,E:∞} not expanded |

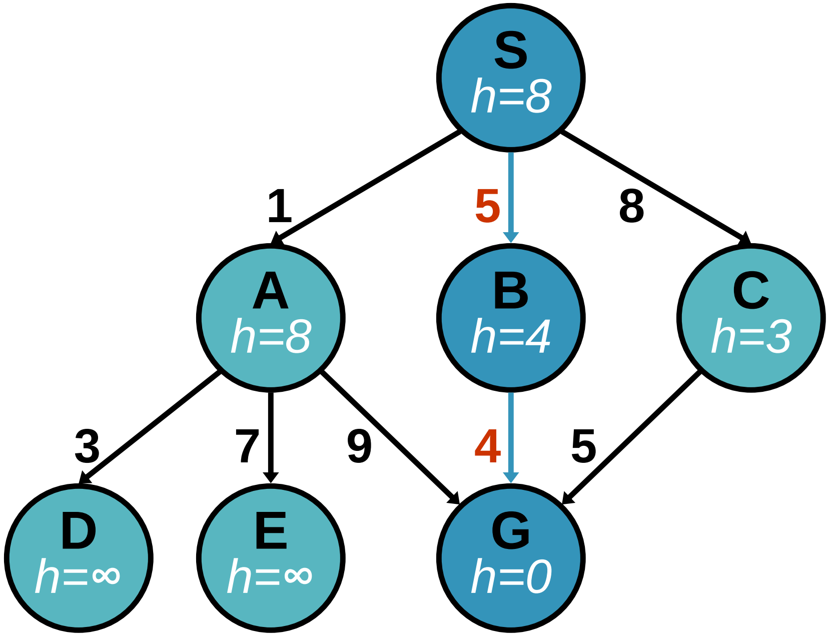


# A* Search

$f(n) = g(n) + h(n)$

\# of nodes tested: 4, expanded: 3

| expnd. node | Frontier |
|---|---|
| | {S:8} |
| S | {A:9,B:9,C:11} |
| A | {B:9,G:10,C:11,D:∞,E:∞} |
| B | {G:9,C:11,D:∞,E:∞} |
| G | {C:11,D:∞,E:∞} |




*Fast, optimal, complete under same graph conditions as Dijkstra's*

**path: S,B,G**
**cost: 9**

# A* performance in Practice

[from Russell and Norvig, Fig 3.29]

Example State

| | | |
|---|---|---|
| 1 | | 5 |
| 2 | 6 | 3 |
| 7 | 4 | 8 |

Goal State

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

| | **For 8-puzzle, average number of states expanded over 100 randomly chosen problems in which optimal path is length …** | | |
|---|---|---|---|
| | **… 4 steps** | **… 8 steps** | **… 12 steps** |
| **Depth-First Iterative Deepening (IDS)** | **112** | **6,300** | **$3.6 \times 10^6$** |
| **A* search using “number of misplaced tiles” as the heuristic** | **13** | **39** | **227** |
| **A* using “Sum of City-Block distances” as the heuristic** | **12** | **25** | **73** |

# Heuristics and A* Performance

- If $h(n) = h^*(n)$ for all $n$,
  - only nodes on optimal solution path are expanded
  - no unnecessary work is performed
- If $h(n) = 0$ for all $n$,
  - the heuristic is admissible
  - A* performs exactly is Dijkstra's

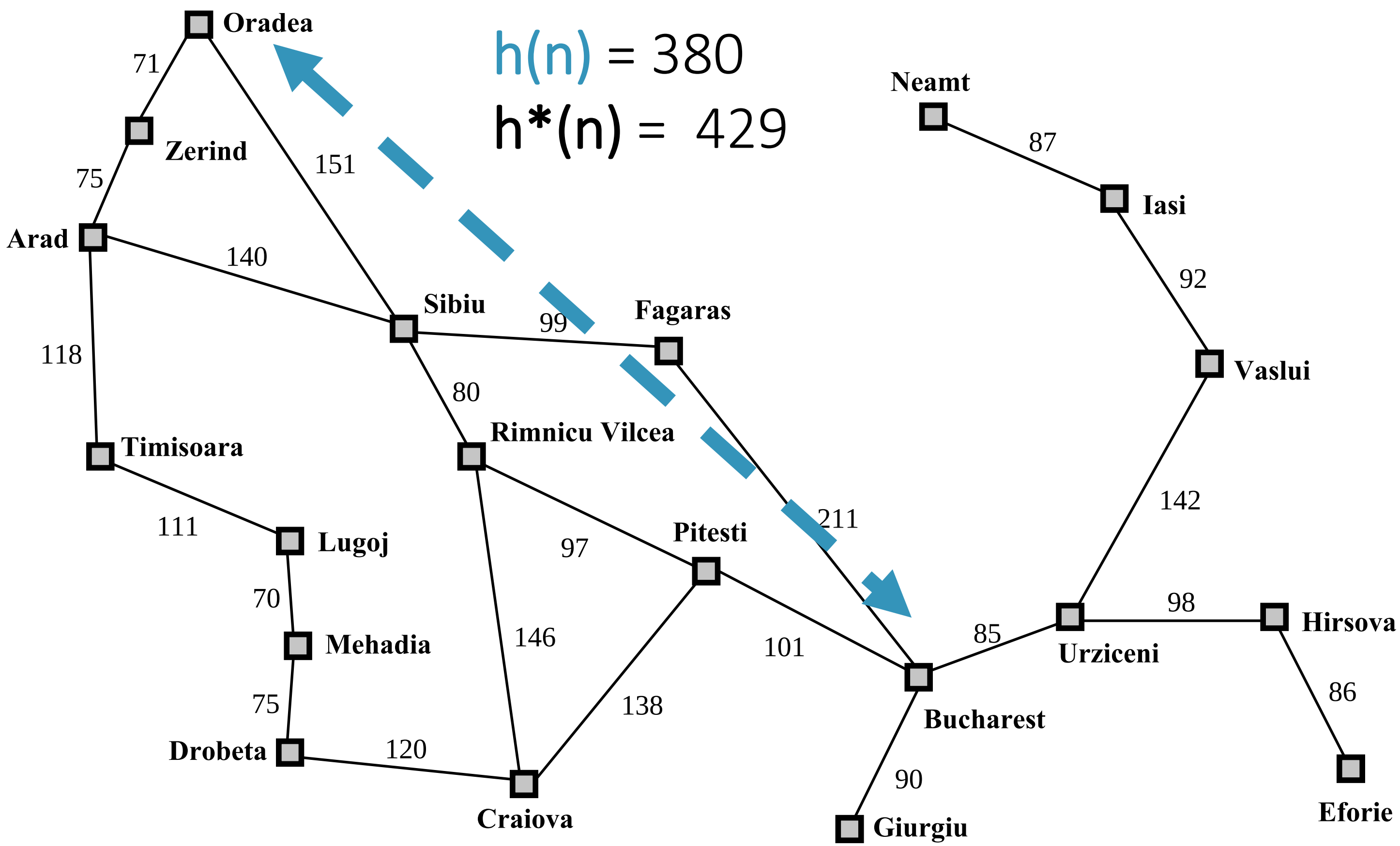
The closer $h$ is to $h^*$,
the fewer extra nodes that will be expanded

# How to come up with admissible/consistent heuristics?

Heuristics are often defined by **relaxing the problem**, i.e., computing the exact cost of a solution to a **simplified** version of problem

- remove constraints
  - **8-puzzle:** Each tile moves independently
- simplify problem
  - **8-puzzle:** A tile can move to any adjacent position ➔ Number of moves to get a tile to its goal position = City-Block distance (aka Manhattan distance or $L_1$ distance)
  - **Map navigation:** Use distance formula from n to goal state as h(n), which will always be optimistic because straight line distance is best case, reality has obstacles

An admissible heuristic is always optimistic. Ex: in this roadmap, h*(n) from Oradea to Bucharest can’t be less than the **straight line distance** from Oradest to Bucharest. Hence, h(n) = straight line distance from Bucharest is admissible.



# A* and Heuristics

**For an admissible heuristic**

- Outside of certain problem domains, $h$ is frequently very simple
- therefore search resorts to (almost) Dijkstra's through parts of the search space

- A* often suffers because it cannot venture down a single path unless it is almost continuously having success (i.e., $h$ is decreasing); any failure to decrease $h$ will cause the search to switch to another path

# The Bad News About A*

- A* uses a lot of memory, ie. O(number of states)
- For some search problems, A* will fill the available system memory before finding a solution

**How do we use less memory?**

1. Sacrifice optimality – beam search
2. Sacrifice some time – iterative deepening A*

# Beam Search

Use an evaluation function ***f*(*n*) = *h*(*n*)** as in Greedy Best-First search, ***and*** restrict the maximum size of the Frontier to a constant, ***k***

- Only keep *k* best nodes as candidates for expansion, and throw away the rest
- More space efficient than Greedy Best-First Search, but may throw away a node on a solution path. Space complexity **O(km)**.
- **Not complete**
- **Not optimal/admissible**

# Iterating Deepening A*

- Iterative-deepening A*
- Cutoff based on *f(n)* = *g* (n)+ *h(n)* value rather than depth
- At each iteration do loop-avoiding DFS, not expanding any node with *f*-value that exceeds current threshold
- At each iteration increase the *f*-value threshold by setting it to the smallest *f*-value of any node that exceeded the cutoff in the previous iteration
- Complete
- Optimal / Admissible
- Linear space required

# Informed and Uninformed Search

These methods are for search problems in the following form:

- A directed graph of states $S$
- Start state(s) $I \subseteq S$
- Goal state(s) $G \subseteq S$
- A successor function *suc*(n) that returns all neighbors of state n
- A cost function c(n, n') that returns the cost of moving from state n to state n'*

*This information can be implied through the structure and labels on a graph

# Informed and Uninformed Search

- These methods are systematic, they search for a **path** from start state to a goal state, then "execute" solution path's sequence of operators

- Effective on small search spaces
- not okay for NP-Hard problems requiring exponential time to find the (optimal) solution

# Example NP-Hard Problem: Traveling Salesman

A salesperson wants to visit a list of cities

- stopping in each city only *once*
- (sometimes also must return to the first city)
- goal is to find an ordering of cities that minimizes total distance traveled

# Example NP-Hard Problem: Traveling Salesman

A solution is a permutation of cities, called a tour

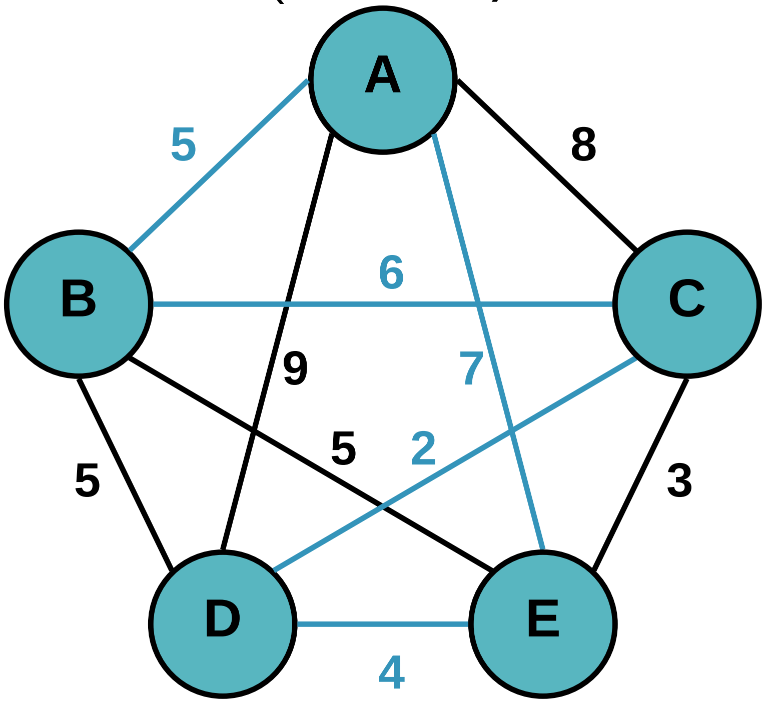
e.g. A – B – C – D – E

**assume tours can start at *any* city and do *not* return home**

| | **A** | **B** | **C** | **D** | **E** |
|---|---|---|---|---|---|
| **A** | 0 | 5 | 8 | 9 | 7 |
| **B** | 5 | 0 | 6 | 5 | 5 |
| **C** | 8 | 6 | 0 | 2 | 3 |
| **D** | 9 | 5 | 2 | 0 | 4 |
| **E** | 7 | 5 | 3 | 4 | 0 |

**5 City TSP**

**(not to scale)**



# Example NP-Hard Problem: Traveling Salesman

## Classic NP-Hard problem

How many solutions exist?

`n!` where `n` = # of cities

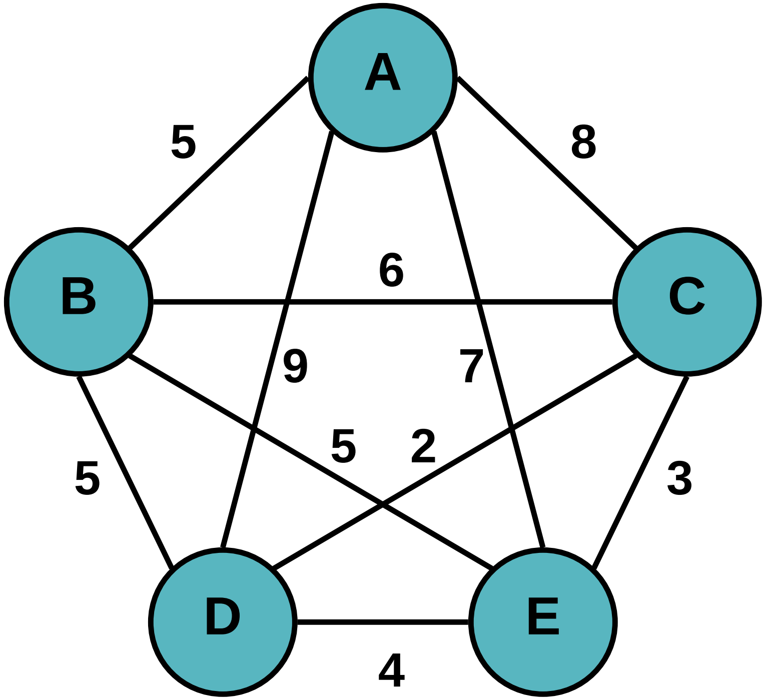
n = 5 results in 120 tours

n = 10 results in 3,628,800 tours

n = 20 results in ~$2.4*10^{18}$ tours

| | A | B | C | D | E |
|---|---|---|---|---|---|
| **A** | 0 | 5 | 8 | 9 | 7 |
| **B** | 5 | 0 | 6 | 5 | 5 |
| **C** | 8 | 6 | 0 | 2 | 3 |
| **D** | 9 | 5 | 2 | 0 | 4 |
| **E** | 7 | 5 | 3 | 4 | 0 |

**5 City TSP**
**(not to scale)**



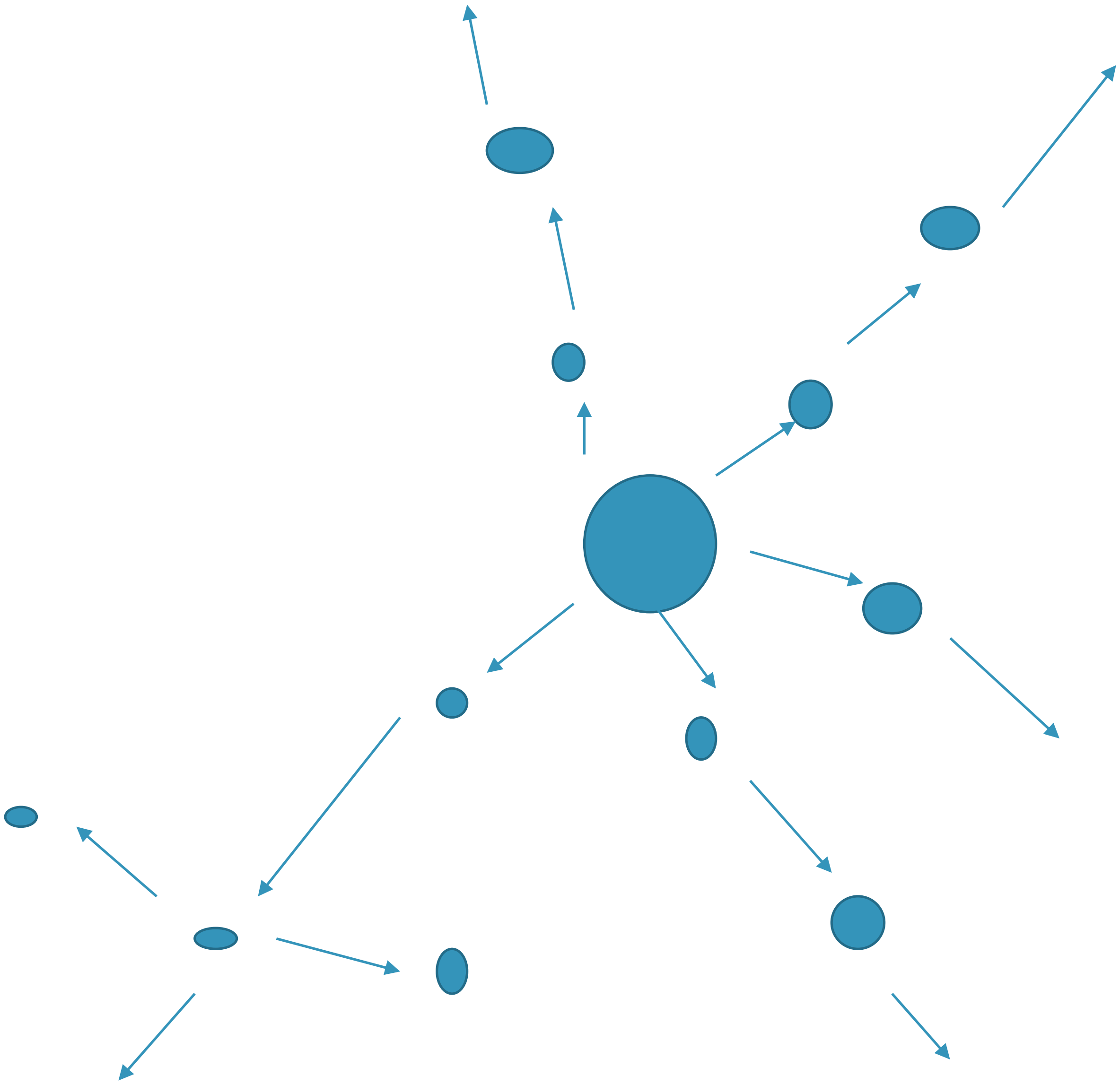
# Solving Hard Problems

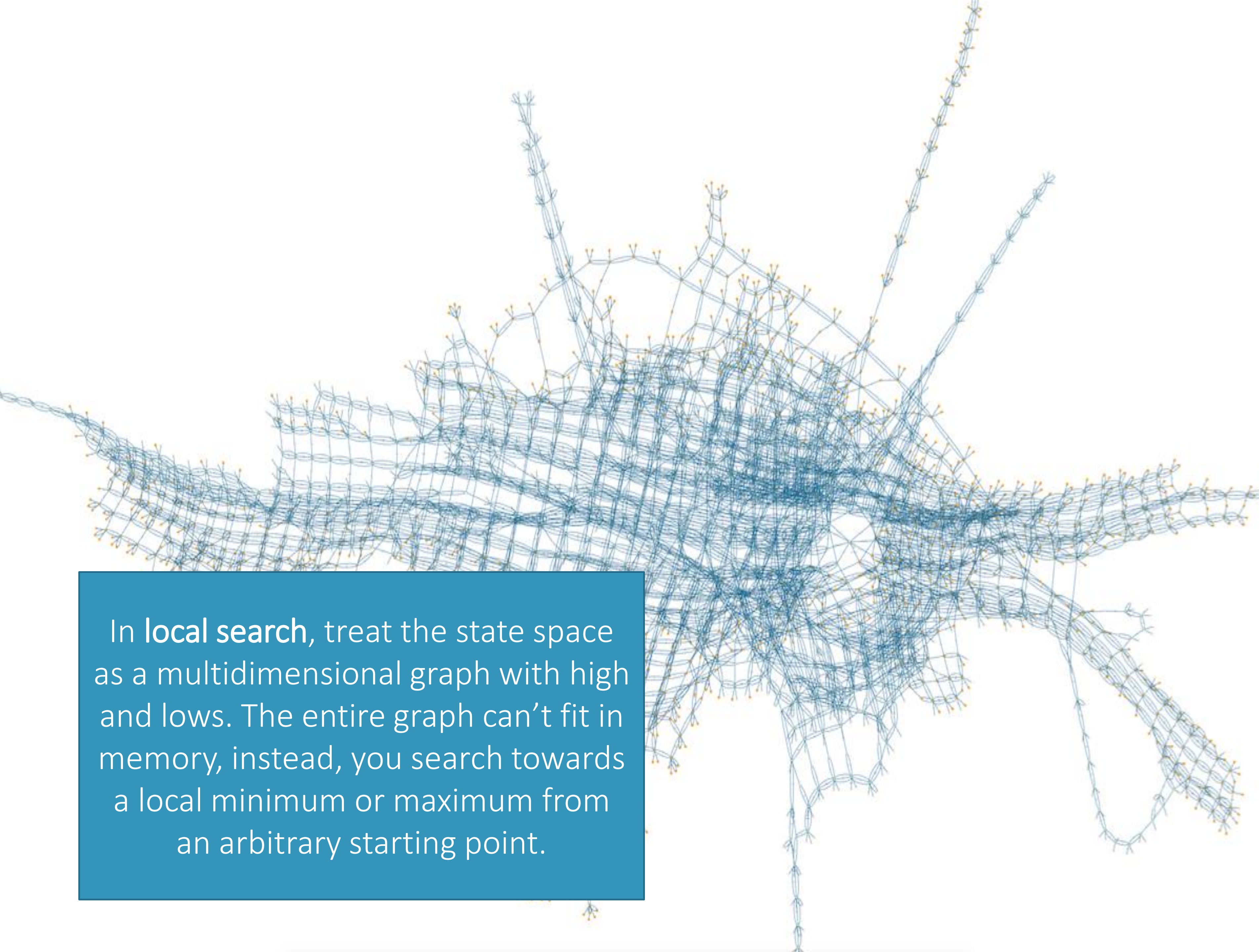
- Hard problems can be solved in polynomial time by using either an:
  - **approximate model**: find an exact solution to a simpler version of the problem
  - **approximate solution**: find a non-optimal solution
    to the original hard problem

- We'll explore means to search through a solution space by iteratively improving solutions until one is found that is optimal or near optimal

What if we don't want or need a path? What if just we want a sufficiently **good** state?

# Local Search





In **local search**, treat the state space as a multidimensional graph with high and lows. The entire graph can’t fit in memory, instead, you search towards a local minimum or maximum from an arbitrary starting point.

# Local Search

- Each state n has a **score** f(n) that we can compute
- The goal is to find the **state** (not path) with the highest score, or a reasonably high score
- This is an optimization problem
- Enumerating entire state space is intractable
- Previous search algorithms are too expensive

# Local Search

**Local searching: every node is a solution**

Operators/actions go from one solution to another

Allows you to stop at any time and have a valid solution

Goal: search is to find a *better/best* solution

- No longer searching state space for a solution path and then executing the steps of the solution path
- A* *isn't* a local search since it considers different partial solutions by looking at the estimated cost of a *solution path*

# Local Searching

An ***operator/action*** is needed to transform one solution to another

- TSP: 2-swap operator
  - take two cities and swap their positions in the tour
  - A-B-C-D-E with swap(A,D) yields D-B-C-A-E
  - possible since graph is fully connected
- TSP: 2-interchange operator (aka 2-opt swap)
  - reverse the path between two cities
  - A-B-C-D-E with interchange(A,D) yields D-C-B-A-E

# Neighbors: TSP

state: A-B-C-D-E-F-G-H-A

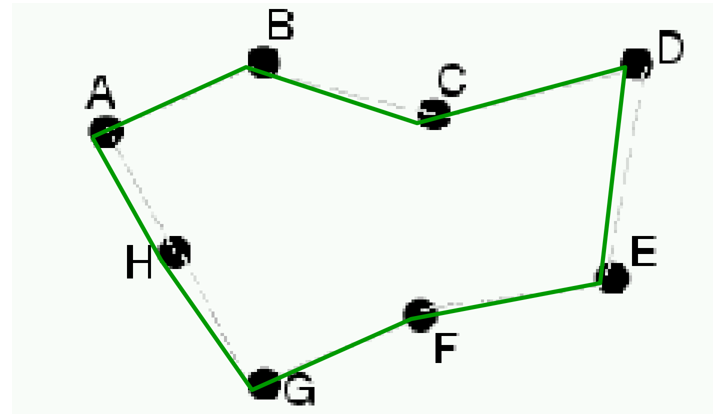
$f$ = length of tour

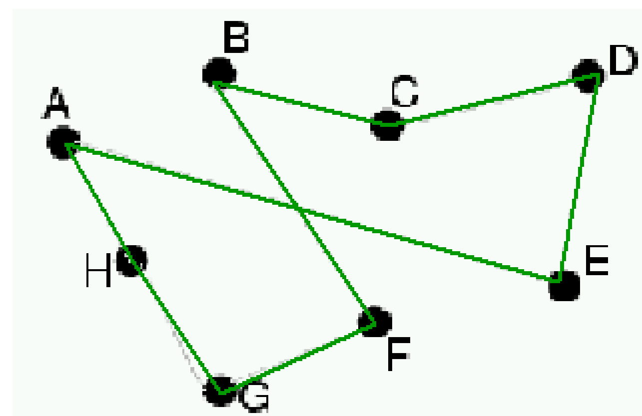
2-interchange

A-B-C-D-E-F-G-H-A

flip

A-E-D-C-B-F-G-H-A



# Local Searching

Those solutions that can be reached with one application of an operator are in the current solution's ***neighborhood*** (aka "***move set***")

**Local search** considers next only those solutions in the neighborhood

- The neighborhood should be much smaller than the size of the search space (otherwise the search degenerates)

# Examples of Neighborhoods

**N-queens**: Move queen in rightmost, most-conflicting column to a different position in that column

**SAT**: Flip the assignment of one Boolean variable

# Local Searching

An evaluation function, $f$, is used to map each solution/state to a number corresponding to the *quality/cost* of that solution

- TSP: Use the length of the tour;
  A better solution has a shorter tour length
- Maximize $f$:
  called **hill-climbing** (gradient ascent if continuous)
- Minimize $f$:
  called or **valley-finding** (gradient descent if continuous

# Hill-Climbing (HC)

**What's a neighbor?**

Problem spaces tend to have structure. A *small change* produces a neighboring state

The size of the neighborhood must be small enough for efficiency

Designing the neighborhood is critical; This is the real ingenuity – not the decision to use hill-climbing

**Pick which neighbor?** The best one (greedy)

What if no neighbor is better than the current state? **Stop**

# Hill-Climbing Algorithm

1. Pick initial state $s$
2. Pick $t$ in neighbors($s$) with the largest $f(t)$
3. **if** $f(t) \leq f(s)$ **then** stop and **return** $s$
4. $s = t$. **Goto** Step 2.

Simple

Greedy

**Stops at a *local* maximum**

# Bonus Section: Proof of A* Optimality

# Proof of A* Optimality (by Contradiction)

- Let

$G$ be the goal in the *optimal* solution

$G2$ be a *sub-optimal* goal found using A* where $f(n) = g(n) + h(n)$, and $h(n)$ is admissible

$f^*$ be the cost of the **optimal path** from Start to $G$

Hence $g(G2) > f^*$

That is, *A** found a sub-optimal path (which it shouldn't)

# Proof of A* Optimality (by Contradiction)

- Let $n$ be some node on the *optimal* path but *not* on the path to $G2$
- $f(n) \leq f^*$

  by admissibility, since $f(n)$ never overestimates the cost to the goal it must be ≤ the cost of the optimal path
- $f(G2) \leq f(n)$

  $G2$ was chosen over $n$ for the sub-optimal goal to be found
- $f(G2) \leq f^*$

  combining equations

# Proof of A* Optimality (by Contradiction)

- $f(G2) \le f^*$
- $g(G2) + h(G2) \le f^*$

  substituting the definition of $f$
- $g(G2) \le f^*$

  because $h(G2) = 0$ since $G2$ is a goal node
- This contradicts the assumption that G2 was sub-optimal, $g(G2) > f^*$

  Therefore, A* is optimal with respect to path cost; A* search never finds a sub-optimal goal