COIT20277 Introduction to Artificial Intelligence

Week 5 - Lecture

- Heuristic Search Techniques

BE WHAT YOU WANT TO BE
cqu.edu.au

Acknowledgement of Country

I respectfully acknowledge the Traditional Custodians of the land on which we live, work and learn. I pay my respects to the First Nations people and their Elders, past, present and future

Acknowledgment

The contents of this lecture have been adopted from the following references:

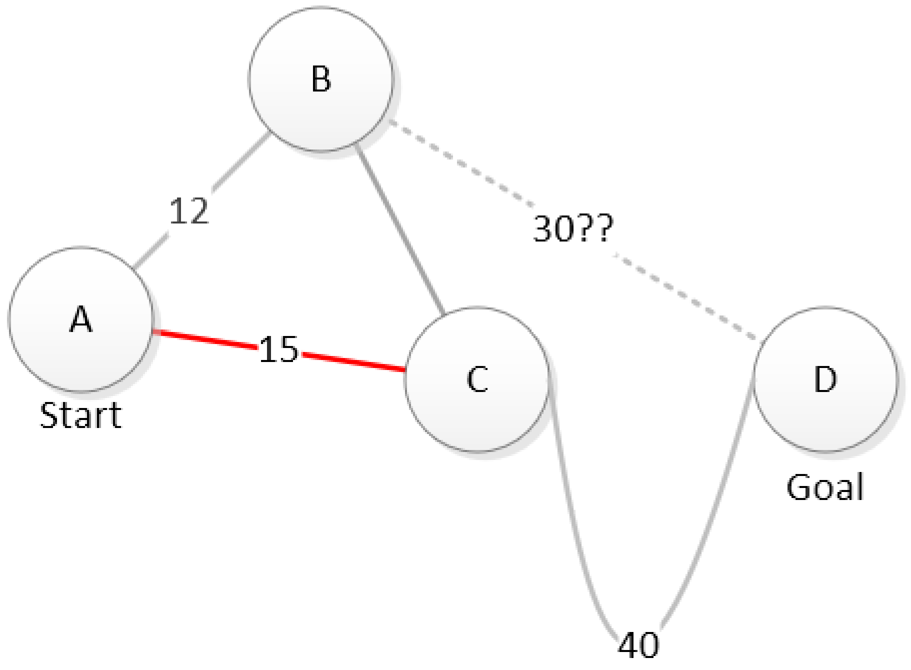
- Artificial Intelligence with Python (2nd edition), *Artasanchez and Joshi*, ISBN 978-1-83921-953-5
    - Chapter 10

Topics

- What is Heuristic Search?
- Uninformed vs. Informed Search
- Uninformed Search
    - Breadth-First Search
    - Depth-First Search
- Informed (or Heuristic) Search
    - A* (or A-star) Search
- Pros and Cons of Heuristic Search
- Constraint Satisfaction Problem (CSP)
- Local Search
    - Hill Climbing
    - Simulated Annealing

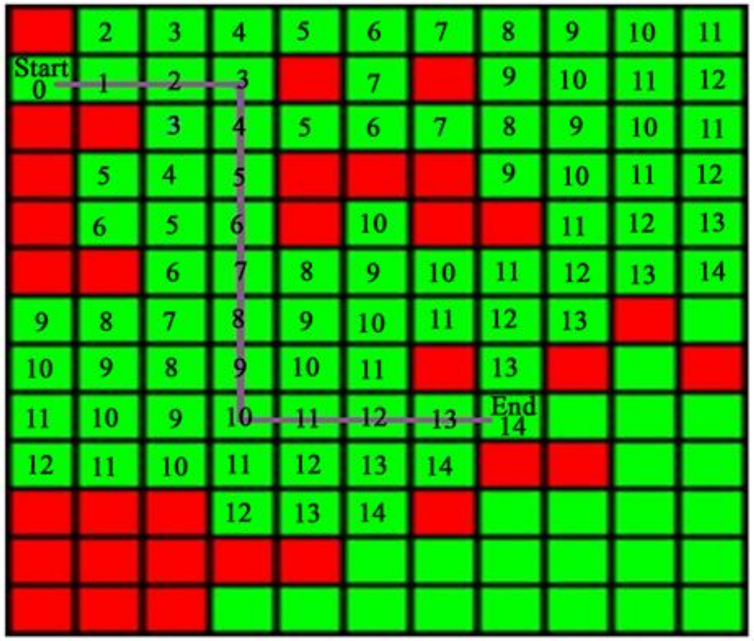
Heuristic Search Techniques

- What is heuristic search and why is it important?
- What are the main types of heuristic search techniques?
- What are some examples of problems that can be solved by heuristic search?



What is Heuristic Search?

- Heuristic search is a method of finding solutions in a large and complex solution space.
- Heuristic search uses a rule of thumb (*heuristic*) to guide the search algorithm and eliminate or prioritize some options.
- Heuristic search can speed up the process and find good solutions, but not necessarily the best ones.

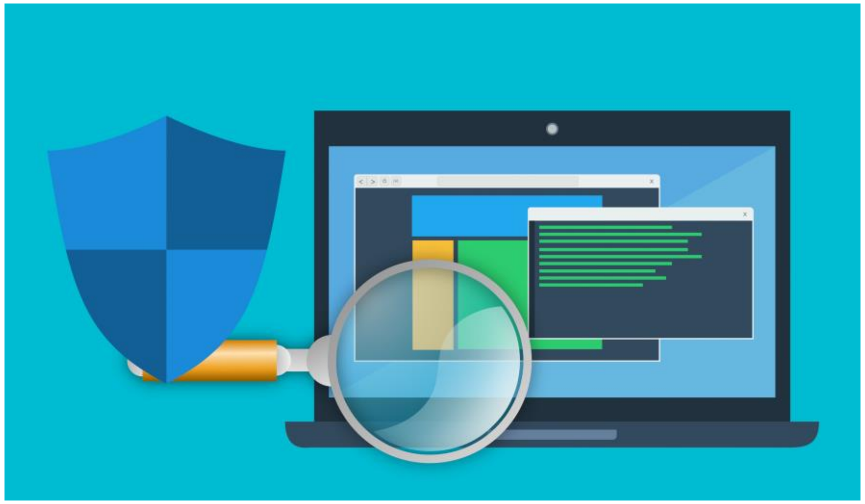


Uninformed vs. Informed Search

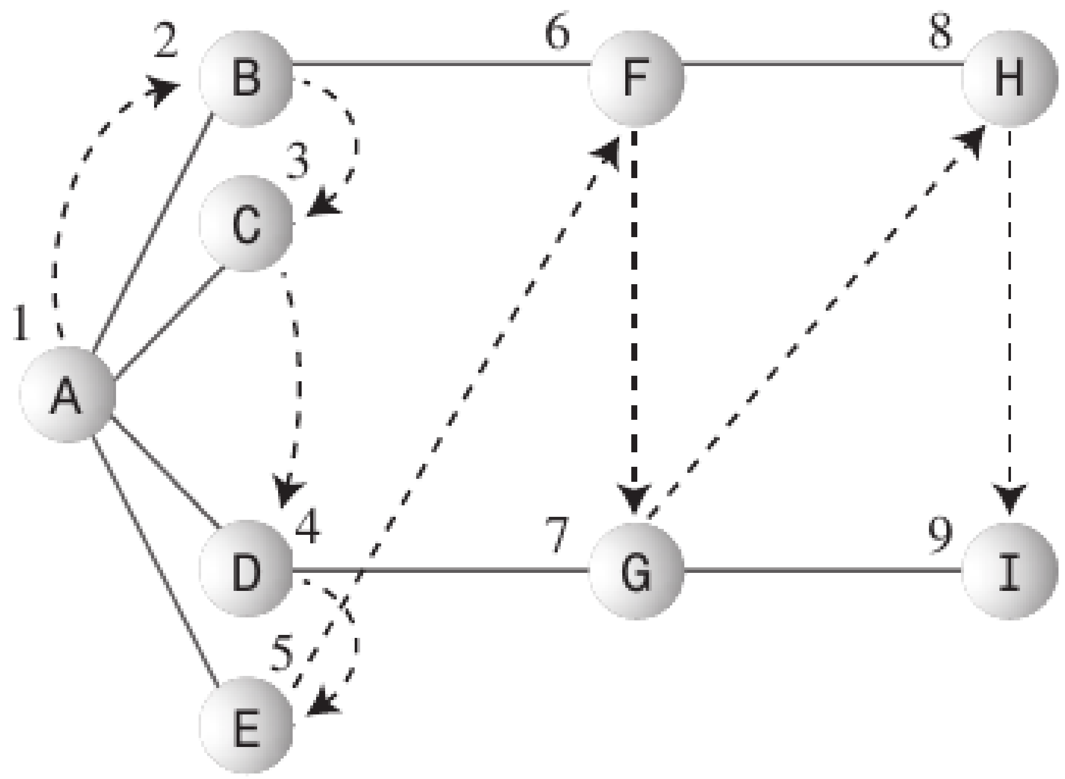
- ***Uninformed search*** algorithms do not use any domain-specific knowledge or heuristics.
- Uninformed search algorithms explore the solution space blindly and uniformly.
- ***Informed search*** algorithms use domain-specific knowledge or heuristics to direct the search.
- Informed search algorithms explore the solution space selectively and efficiently.

Uninformed Searches

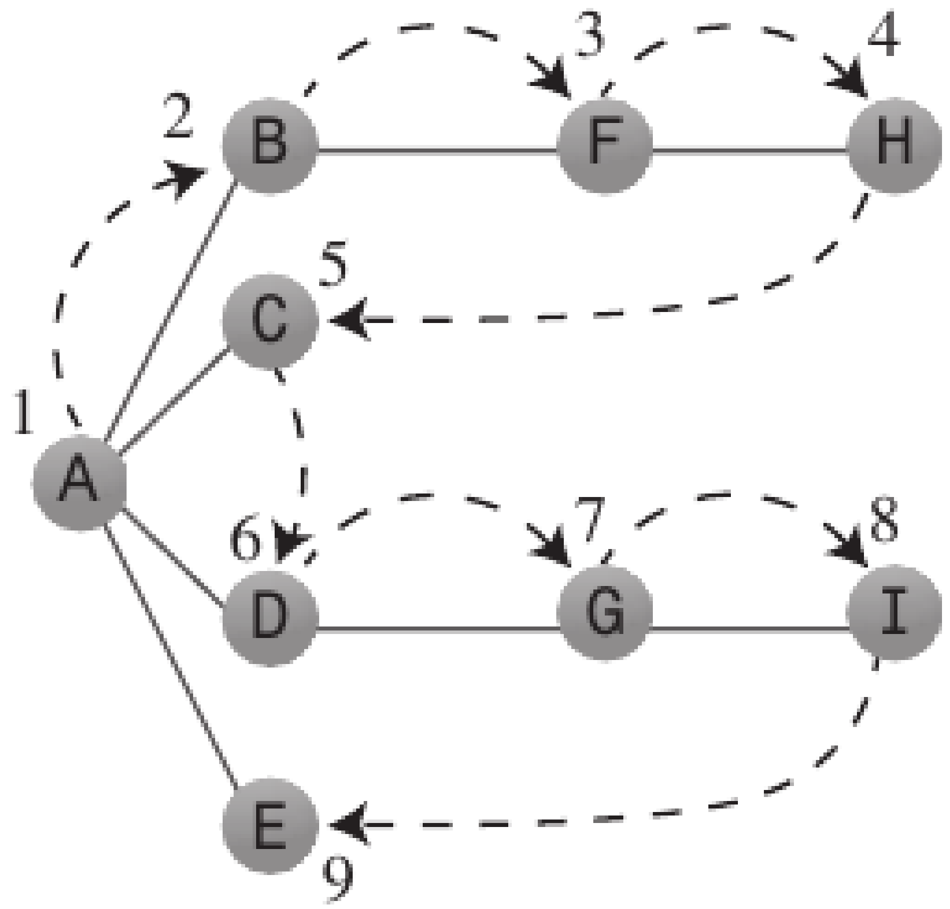
- ***Uninformed searches*** do not use any prior information or rules to eliminate some paths.
- They check all the plausible paths and pick the optimal one. They are simple to implement but can be inefficient for large graphs.
- Widely used Uninformed Searches:
  - Depth First Search (DFS): Explores all nodes along a branch before moving to the next branch.
  - Breadth First Search (BFS): Explores all nodes at a given level before moving to the next level.



Breadth-First Search

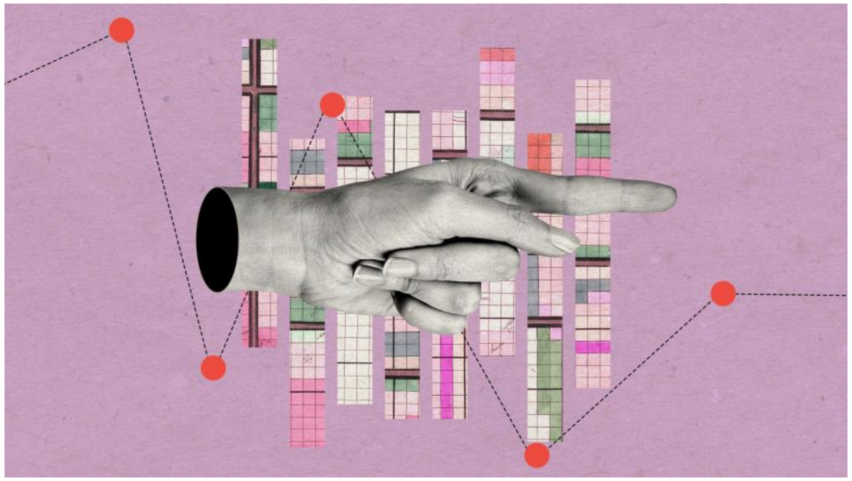


Depth-First Search

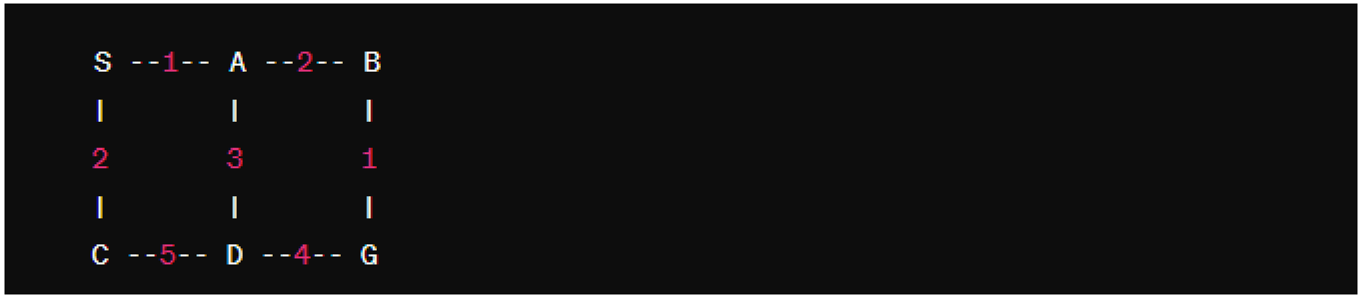


Informed (or Heuristic) Searches

- Uses prior information or rules to guide the search by eliminating unnecessary paths, rendering them more efficient than uninformed searches.
- Defines a heuristic function that estimates the cost of the path from the current node to the goal.
- May not always find the most optimal solution but guarantees a good solution in a reasonable time.
- Widely used informed search include A* search, which combines the strengths of BFS and DFS, using both exploration and informed guidance.



A* Search



```
S --1-- A --2-- B
|       |       |
2       3       1
|       |       |
C --5-- D --4-- G
```

- Let's consider a simple example of finding the shortest path from the start node 'S' to the goal node 'G' in a graph.
- Each letter represents a node in the graph.
- The numbers on the edges represent the cost to traverse between nodes.
- Let's assume the heuristic function estimates the straight-line distance between each node and the goal node 'G'. In this case, we'll use the following heuristic values:
  - $h(S) = 5$
  - $h(A) = 4$
  - $h(B) = 3$
  - $h(C) = 3$
  - $h(D) = 2$
  - $h(G) = 0$

A* Search (cont…)

- Now, let's run the A* search algorithm to find the shortest path from 'S' to ‘G’:

1. Start at node ‘S’, with a total cost of 0 (since it’s the starting node) and an empty path.
2. Expand node ‘S’ and add its neighbours ‘A’ and ‘C’ to the open list.
3. Calculate the total estimated cost $f(n)$ for ‘A’ and ‘C’:
    - For ‘A’: $f(A) = g(A) + h(A) = 1 + 4 = 5$
    - For ‘C’: $f(C) = g(C) + h(C) = 2 + 3 = 5$
4. Expand node ‘A’ next since it has the lowest $f(n)$ value.
5. Add ‘B’ and ‘D’ to the open list.
6. Calculate the total estimated cost $f(n)$ for ‘B’ and ‘D’:
    1. For ‘B’: $f(B) = g(B) + h(B) = 3 + 3 = 6$
    2. For ‘D’: $f(D) = g(D) + h(D) = 4 + 2 = 6$
7. Expand node ’B’ next since it has the lowest $f(n)$ value.
8. Add ’G’ to the open list.
9. Expand node ’G’, which is the goal node. The algorithm terminates.
10. The shortest path from ’S’ to ’G’ is found: $S \rightarrow A \rightarrow B \rightarrow G$ with a total cost of 4.

- The A* search algorithm efficiently found the shortest path from 'S' to 'G' by considering both the actual cost from the start node to each node and the heuristic estimates of the remaining distance to the goal node.

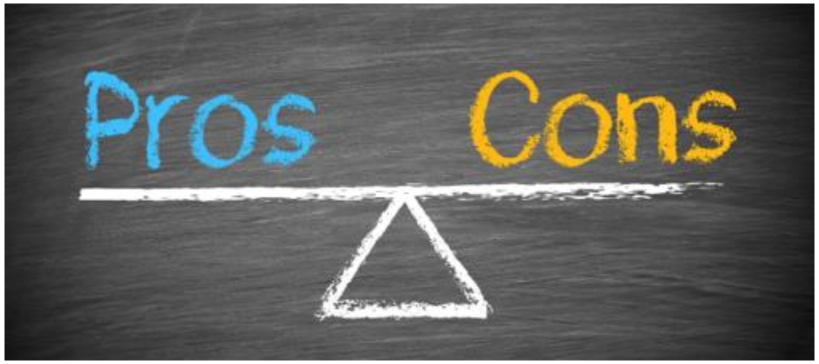
Pros and Cons of Heuristic Searches

Pros

- Heuristic searches offer significant speed advantages over uninformed searches, especially for complex problems like searching large graphs.
- Can be used in cases where uninformed searches are impractical.
- Provide good solutions in a reasonable amount of time.

Cons

- The trade-off of potentially missing the absolute best solution.
- Effectiveness depends heavily on the quality of the chosen heuristic function.
- Can be more complex to implement than uninformed searches.

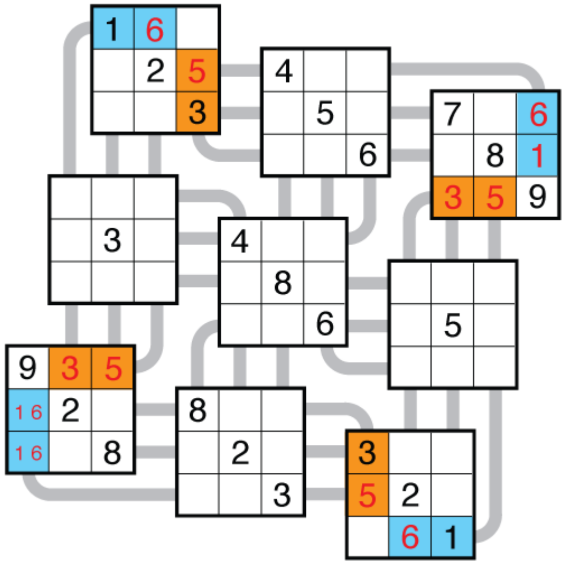


Constraint Satisfaction Problems (CSPs)

- CSPs arise in various scenarios where we need to find solutions that adhere to specific rules or limitations.
- One can use heuristic search techniques to solve CSPs efficiently.
- What are constraints?
  - They are limitations or rules that guide the solution process.
  - They dictate what options are permissible and which ones are forbidden.
  - Imagine playing chess – certain moves are legal based on the piece you're controlling and the positions of other pieces on the board.

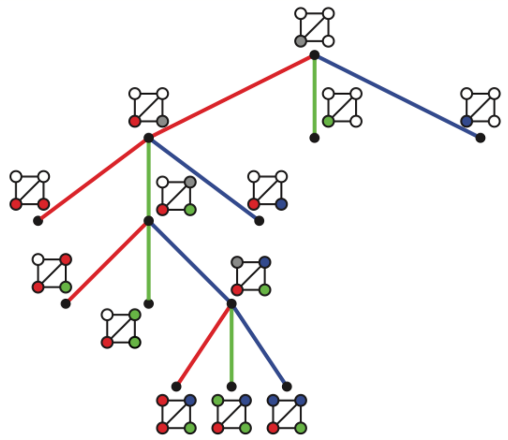
Sudoku: A CSP Example

- Sudoku is a classic puzzle game where you fill a 9x9 grid with digits 1-9, ensuring no row, column, or 3x3 subgrid contains the same digit twice.
- These rules translate into constraints for a CSP solver.
- Each square represents a variable, and the possible values (digits 1-9) are its domain.
- Constraints dictate that no two variables within the same row, column, or sub-grid can share the same value.
- By satisfying these constraints, we reach a valid Sudoku solution.



Brute Force vs. CSP Approach

- Brute force: Testing all possible combinations exhaustively, like trying every number in each Sudoku square.
- Imagine solving a 100x100 Sudoku using brute force – it would take an eternity! That's where CSP shines.
- CSP uses constraints to eliminate invalid combinations early on, reducing the search space.
- Like knowing beforehand that a number can't be placed in a square due to existing constraints in its row, column, or sub-grid.



CSP Formulation

- CSP is defined by:
  - Variables: Entities whose values need to be determined.
  - Domains: Sets of possible values for each variable.
  - Constraints: Rules that restrict combinations of variable values.
- Imagine a scheduling or timetabling problem:
  - Variables: Tasks, timeslots.
  - Domains: Available time slots for each task.
  - Constraints: Two tasks requiring the same resource can't be assigned the same time.
- Involves clearly defining the variables, their allowed values (domains), and the constraints that govern their interactions.
- It's like setting the stage for the search algorithm to find a solution where all the actors (variables) play their roles (values) while respecting the stage directions (constraints).

Local Search for Constraint Satisfaction Problems

- Local search is a powerful technique for solving Constraint Satisfaction Problems (CSPs).
- It works by iteratively improving a solution by updating variables until all constraints are satisfied.
- Each update moves the solution closer to the goal (minimum constraint violation or optimal state).
- Hence the name "local search" – it explores solutions in the vicinity of the current one.
- **Example:** Imagine navigating a maze. Local search takes small steps, exploring neighboring paths, always choosing the one that leads closer to the exit.
- Similarly, when solving a CSP, it makes changes to variable values, ensuring each change brings the solution closer to satisfying all constraints.

Local Search Algorithms: Guiding the Search

- Local search algorithms use heuristic functions to evaluate solutions.
- These functions estimate the "cost" of a solution (constraint violations, distance to goal).
- The algorithm aims to minimize the cost at each step.
- Think of a heuristic function as a guidepost in the maze. It estimates how promising each neighboring path is, helping the algorithm choose the direction that likely leads to the exit faster.
- In CSPs, the heuristic measures how well a solution adheres to the constraints, guiding the search towards an optimal state.

Hill Climbing: Climbing the Cost Landscape

- ***Hill climbing***, a popular local search technique, iteratively improves the solution.
- It compares the current solution to its neighbors and chooses the one with the lowest cost (constraint violation or distance to goal).
- It "climbs" the cost landscape, seeking the peak, which represents the optimal solution.
- However, it can get stuck on local optima (smaller peaks), not reaching the global optimum (highest peak).

Simulated Annealing: Escaping Local Optima

- Simulated annealing is a powerful local search technique that helps us avoid getting stuck in local optima.
- It draws inspiration from the annealing process in *metallurgy*.
- Metals are heated, allowing atoms to rearrange, and then cooled slowly to achieve a desired structure.
- Simulated annealing guides the search towards better solutions by allowing occasional "uphill" moves, mimicking the heat phase, and then gradually restricting these moves as the "cooling" progresses.
- This analogy translates to searching for optimal solutions.

The Objective Function: Guiding the Search

- An ***objective function*** serves as the compass, guiding the search towards better solutions.
- It evaluates the "goodness" of each state, helping the algorithm make informed decisions.
- Think of the objective function as a scorecard. It assigns a score to each candidate solution, indicating its "fitness" for the problem.
- Simulated annealing uses this score to compare states, favoring those with lower scores (deeper valleys) and occasionally accepting higher scores (uphill moves) with a controlled probability.

Cooling Down: The Annealing Schedule

- The annealing schedule controls how quickly the system "cools down," impacting the search process.
- A *slow cooling* schedule allows for more exploration and potential escape from local optima.
- A *fast cooling* schedule converges quickly but might miss better solutions trapped in deeper valleys.
- Just like the cooling rate determines the final structure of metal, the annealing schedule in simulated annealing plays a crucial role.
- A well-designed schedule balances exploration and exploitation, allowing the search to escape local traps while efficiently converging towards the optimal solution.

THANK YOU

TIME FOR DISCUSSION & QUESTIONS