COL333/671: Introduction to AI

Semester I, 2024-25

Local Search Algorithms

Rohan Paul

Outline

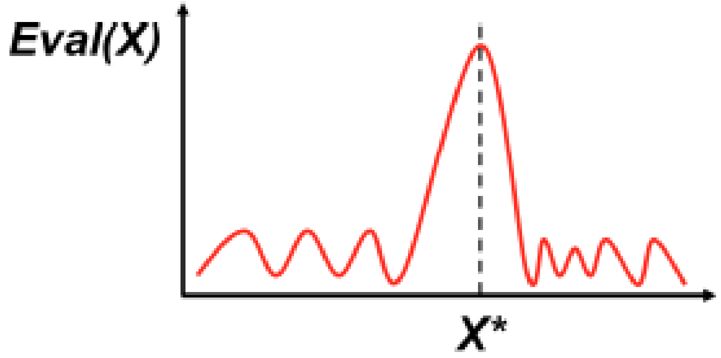
- Last Class
  - Informed Search
- This Class
  - Local Search Algorithms
- Reference Material
  - AIMA Ch. 4.1

Acknowledgement

These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Doina Precup, Dorsa Sadigh, Percy Liang, Mausam, Dan Klein, Nicholas Roy and others.

Search Methods for Discrete Optimization

- **Setting**
  - A set of discrete states, *X*.
  - An objective/evaluation function assigns a "goodness" value to a state, *Eval(X)*
  - Problem is to search the state space for the state, *X** that maximizes the objective.



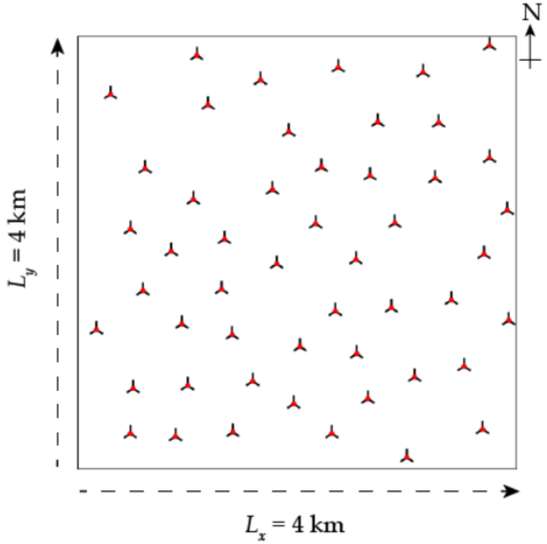
- **Searching for the optimal solution can be challenging. Why?**
  - The number of states is very large.
    - Cannot simply enumerate all states and find the optimal.
  - We can only evaluate the function.
    - Cannot write it down analytically and optimize it directly.

Key Idea
- **Searching for "the optimal" solution is very difficult.**
- **Question is whether we can search for a reasonably good solution.**

Example – Windmill Placements

Problem: Optimizing the locations of windmills in a wind farm

- An area to place windmills.
- Location of windmills affects the others. Reduced efficiency for those in the wake of others.
- Grid the area into bins.
- A large number of configurations of windmills possible.
- Given a configuration we can evaluate the total efficiency of the farm.
- Can neither enumerate all configurations nor optimize the power efficiency function analytically.
- **Goal is to <u>search</u> for the <u>configuration</u> that maximizes the efficiency.**



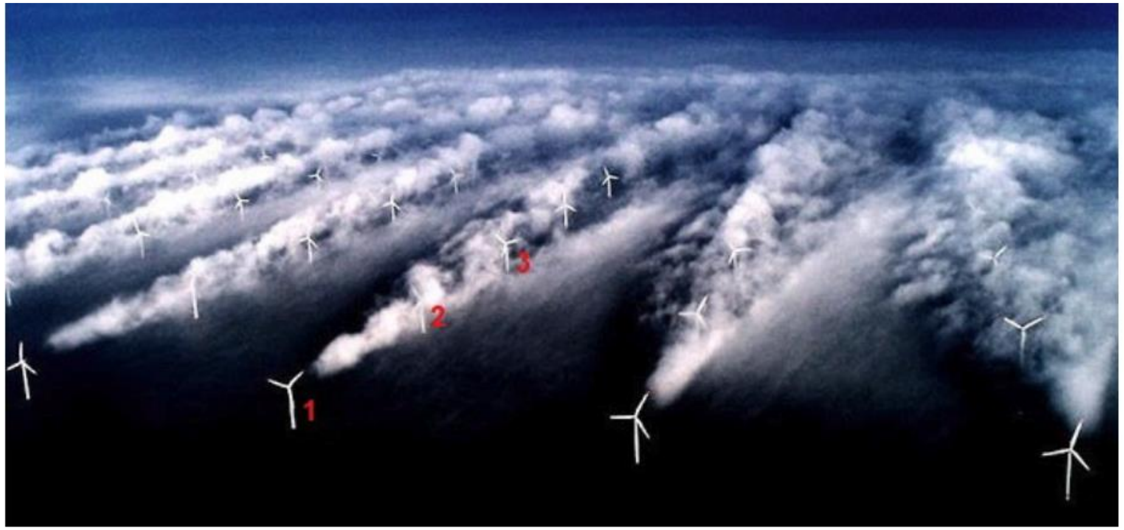
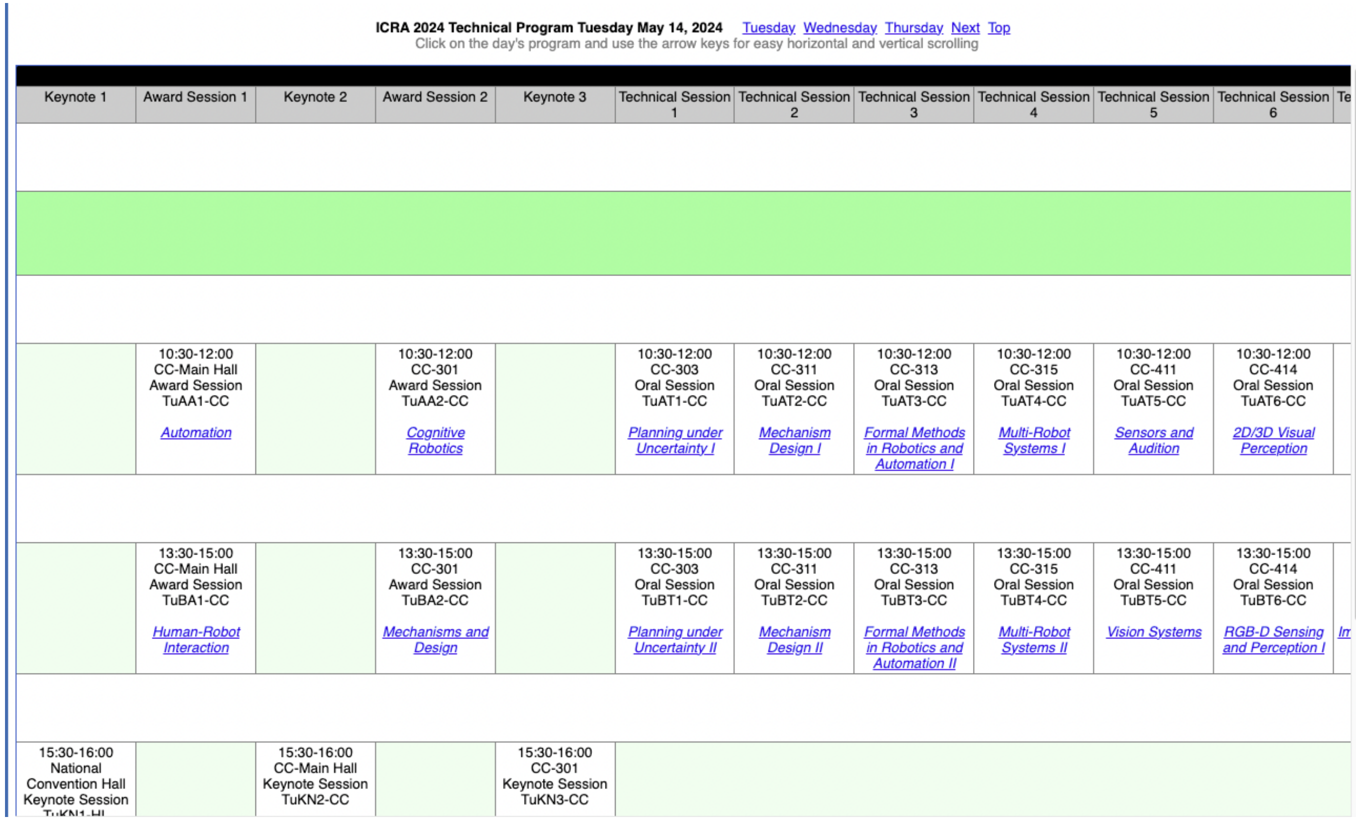


Figure 5: Turbines experiencing multiple wakes. As an example, turbine 3 is experiencing wake effects from both turbine 1 and 2. Image adopted from [4].

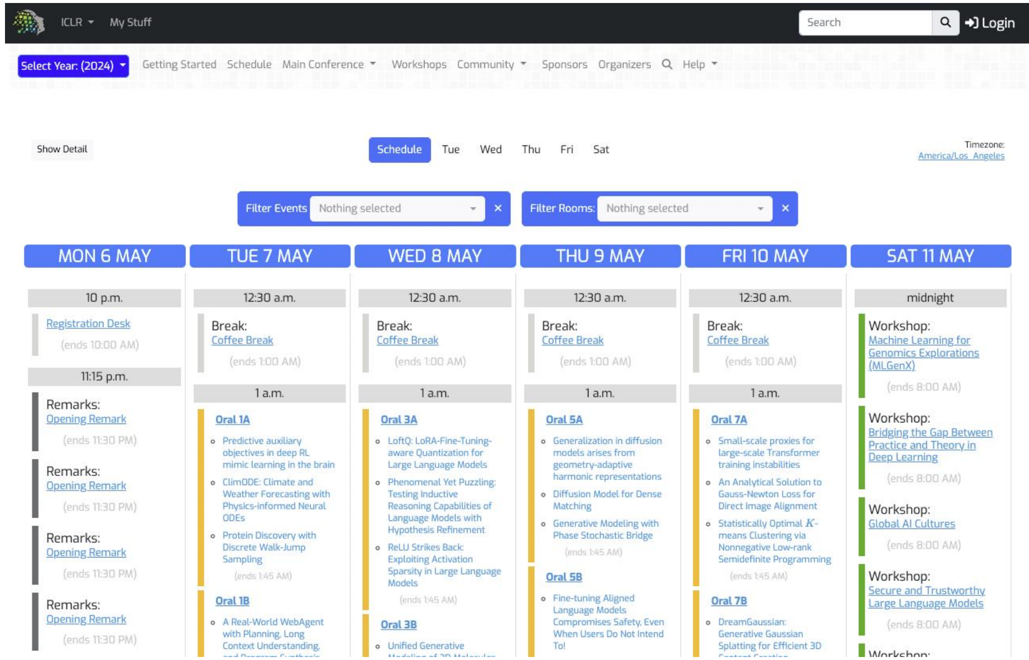
Inspired from this example: https://www.shell.in/energy-and-innovation/ai-hackathon/_jcr_content/par/textimage_1834506119_1619963074.stream/1612943059963/4b0a86b7cc0fe7179148284ffed9ef33524c2816/windfarm-layout-optimisation-challenge.pdf

Example: Conference Scheduling



ICRA 2024 Technical Program Tuesday May 14, 2024 Tuesday Wednesday Thursday Next Top
Click on the day's program and use the arrow keys for easy horizontal and vertical scrolling

Keynote 1	Award Session 1	Keynote 2	Award Session 2	Keynote 3	Technical Session 1	Technical Session 2	Technical Session 3	Technical Session 4	Technical Session 5	Technical Session 6	Te
	10:30-12:00 CC-Main Hall Award Session TuAA1-CC *Automation*		10:30-12:00 CC-301 Award Session TuAA2-CC *Cognitive Robotics*		10:30-12:00 CC-303 Oral Session TuAT1-CC *Planning under Uncertainty I*	10:30-12:00 CC-311 Oral Session TuAT2-CC *Mechanism Design I*	10:30-12:00 CC-313 Oral Session TuAT3-CC *Formal Methods in Robotics and Automation I*	10:30-12:00 CC-315 Oral Session TuAT4-CC *Multi-Robot Systems I*	10:30-12:00 CC-411 Oral Session TuAT5-CC *Sensors and Audition*	10:30-12:00 CC-414 Oral Session TuAT6-CC *2D/3D Visual Perception*	
	13:30-15:00 CC-Main Hall Award Session TuBA1-CC *Human-Robot Interaction*		13:30-15:00 CC-301 Award Session TuBA2-CC *Mechanisms and Design*		13:30-15:00 CC-303 Oral Session TuBT1-CC *Planning under Uncertainty II*	13:30-15:00 CC-311 Oral Session TuBT2-CC *Mechanism Design II*	13:30-15:00 CC-313 Oral Session TuBT3-CC *Formal Methods in Robotics and Automation II*	13:30-15:00 CC-315 Oral Session TuBT4-CC *Multi-Robot Systems II*	13:30-15:00 CC-411 Oral Session TuBT5-CC *Vision Systems*	13:30-15:00 CC-414 Oral Session TuBT6-CC *RGB-D Sensing and Perception I*	*In*
15:30-16:00 National Convention Hall Keynote Session		15:30-16:00 CC-Main Hall Keynote Session TuKN2-CC		15:30-16:00 CC-301 Keynote Session TuKN3-CC							

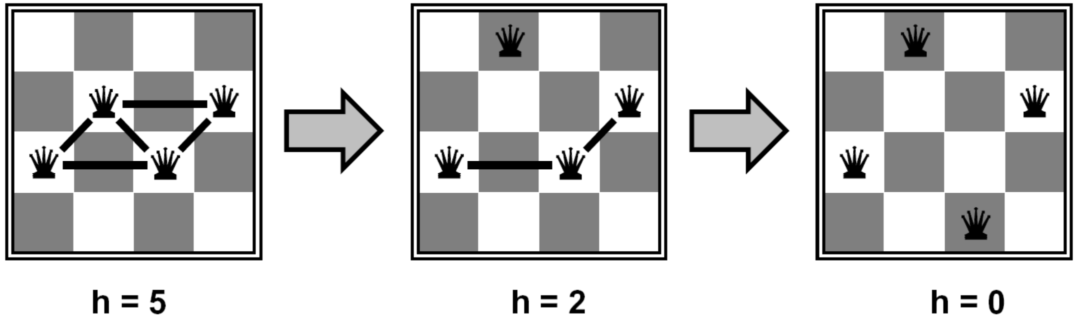


Assign papers that are similar in a session. Avoid conflicts between sessions.

Example

4-Queens Problem

- Discrete set of states: 4 queens in 4 columns (4^4 = 256 states)
- Goal is to find a configuration such that there are no attacks.
  - Moving a piece will change the configuration.
- Any configuration can be evaluated using a function
  - h(x) = number of attacks (number of violated binary constraints)
- Search for the configuration that is optimal such that h = 0.



Example

Formally

Variables: x_0, x_1, x_2, x_3 where x_i is the row position of the queen in column i, where $i \in \{0, 1, 2, 3\}$. Assume that there is one queen per column.

Domains: $x_i \in \{0, 1, 2, 3\}\ \forall i$.

Initial state: 4 queens on the board in random row positions.

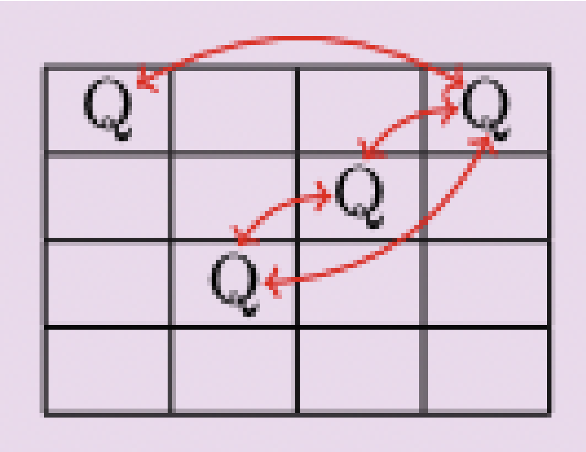
Goal state: 4 queens on the board with no pair of queens attacking each other.

Neighbour relation:

- Version A: move a single queen to a different row in the same column.
- Version B: swap the row positions of two queens.

Cost function: the number of pairs of queens attacking each other, directly or indirectly.

Number of attacks are 4.



Local Search Methods

- **Keep track of a single "current" state**
  - We need a principled way to search/explore the state space hoping to find the state with the optimal evaluation.
  - Do not maintain a search tree as we need the solution not the path that led to the solution.
  - Only maintain a single current state.

- **Perform local improvements**
  - Look for alternatives in the vicinity of that solution
  - Try to move towards more better solutions.

Hill-climbing Search

Let S be the start node and let G be the goal node.

Let h(c) be a heuristic function giving the value of a node

Let c be the start node

Loop

    Let c' = the highest valued neighbor of c
    If h(c) ≥ h(c') then return c
    c = c'

Start at a configuration. Evaluate the neighbors. Move to the highest valued neighbor if its value is higher than the current state. Else stay.

Hill climbing

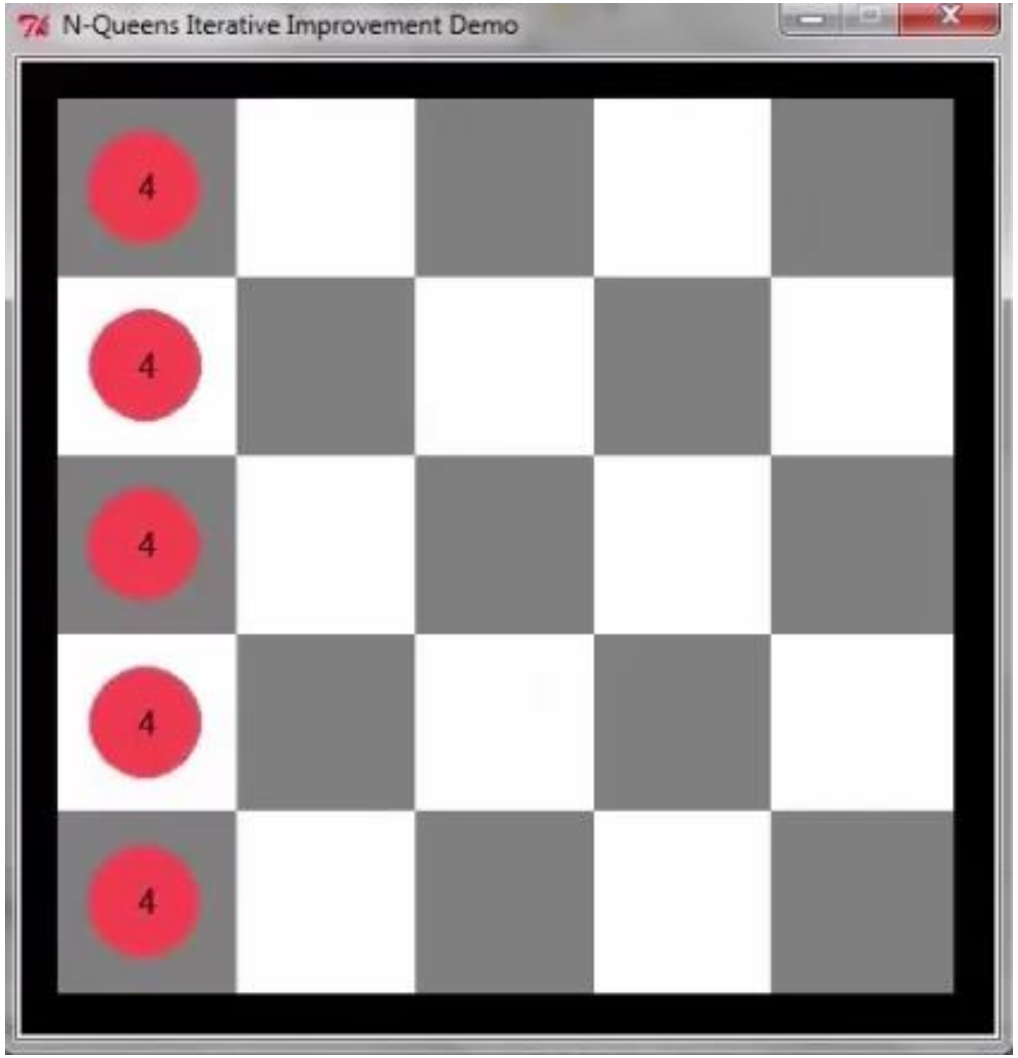
"Like climbing Everest in thick fog with amnesia"

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

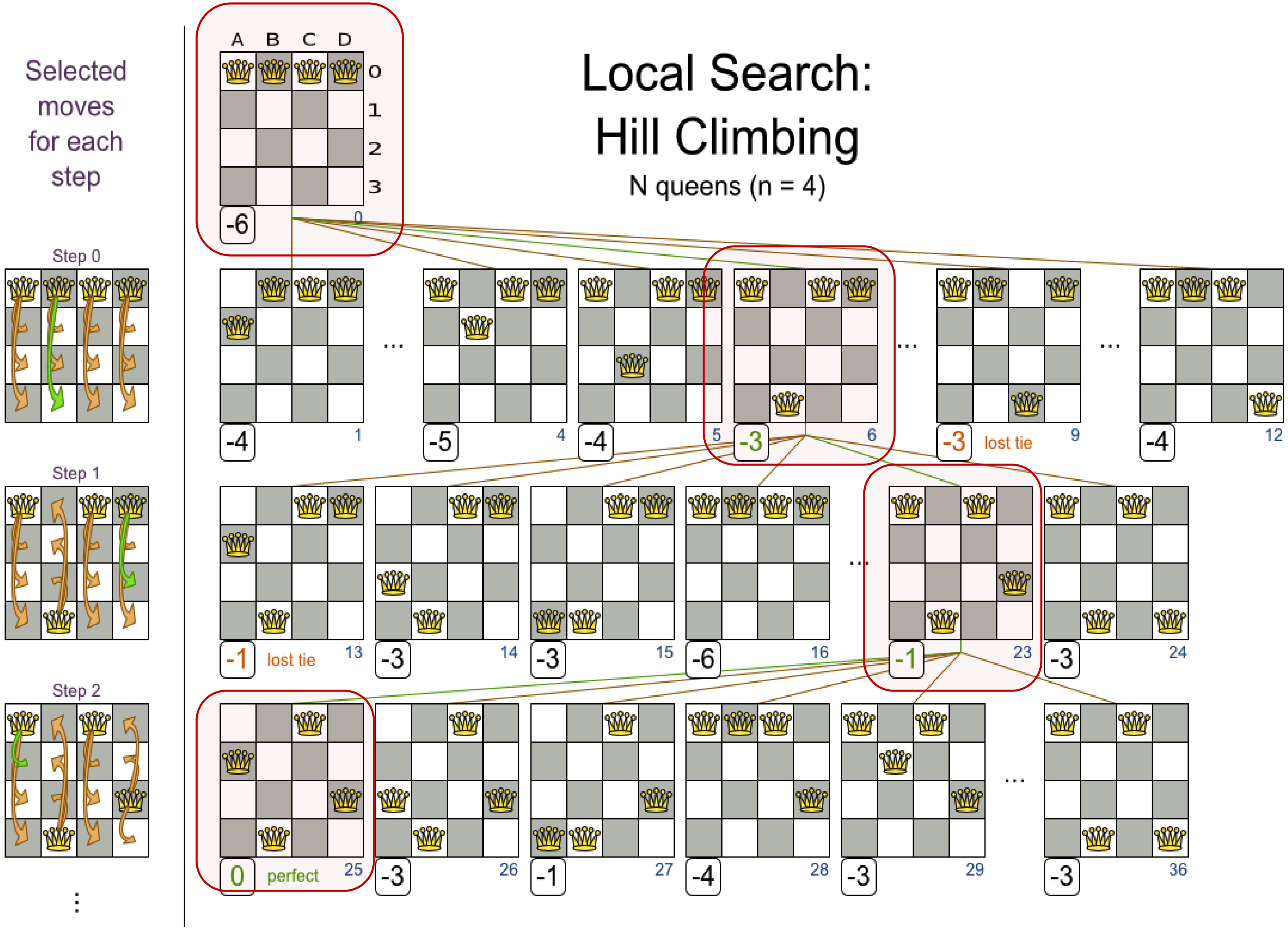
Hill climbing for 4 -queens

- Select a column and move the queen to the square with the fewest conflicts.
- Perform local modifications to the state by changing the position of one piece till the evaluation is minimum.
- Evaluate the possibilities from a state and then jump to that state.



Example

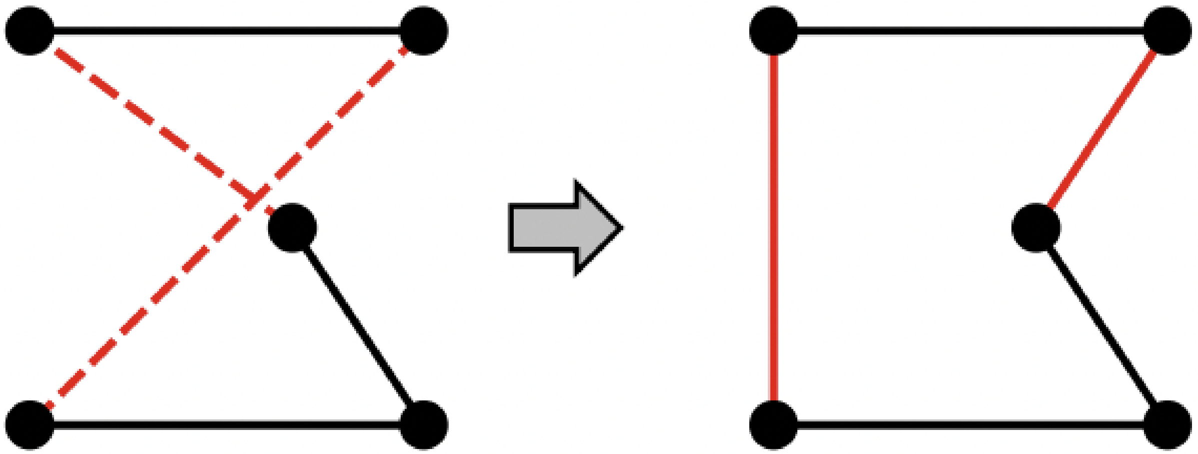
- Local search looks at a state and its <u>local</u> neighborhood.
- <u>Not</u> constructing the entire search tree.
- Consider local modifications to the state. Immediately jump to the next promising neighbor state. Then start again.
- *Highly scalable.*



Example: Idea of local improvements

Locally improving a solution for a Travelling Salesperson Problem.

Start with any complete tour, perform pairwise exchanges

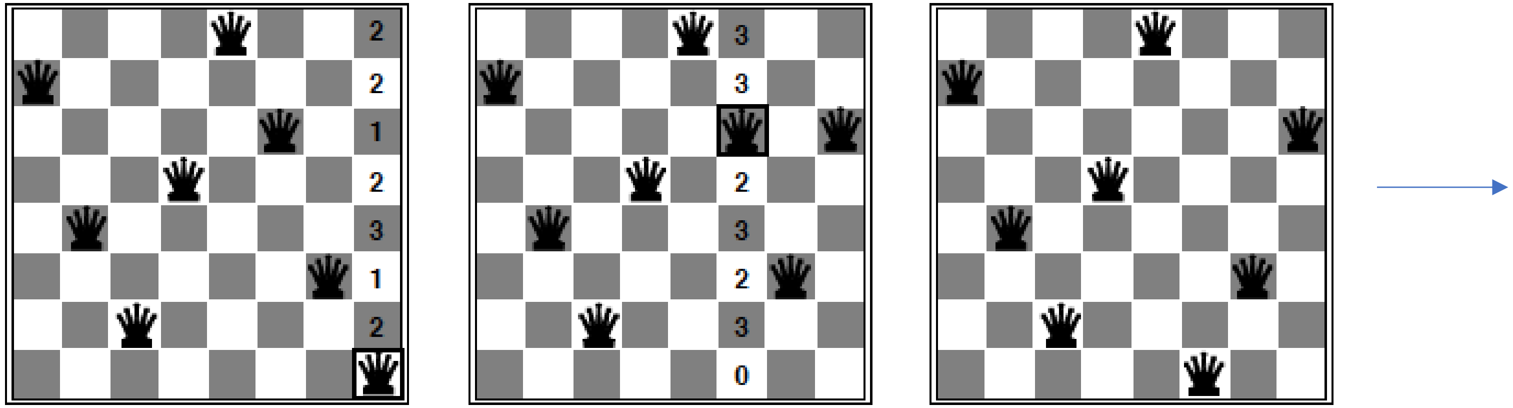


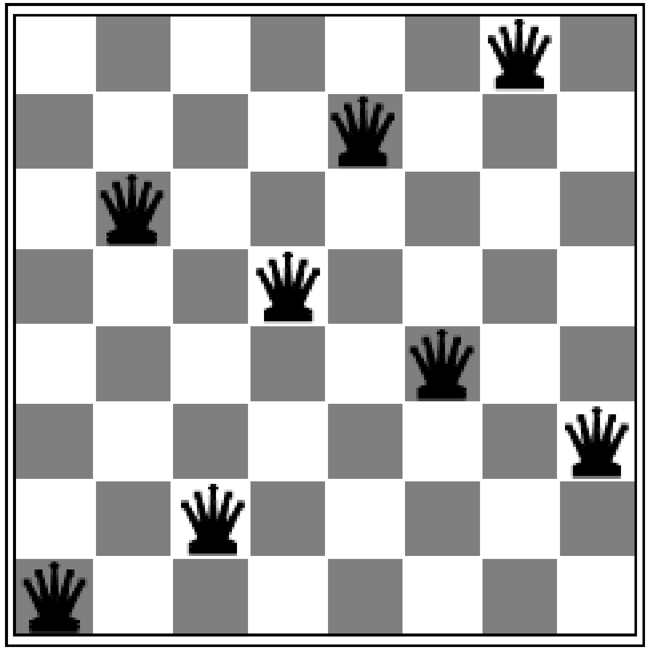
Variants of this approach get within 1% of optimal very quickly with thousands of cities

The idea of making local improvements to a candidate solution is a general and widely applicable technique.

8-Queens Problem

Is this an optimal state?





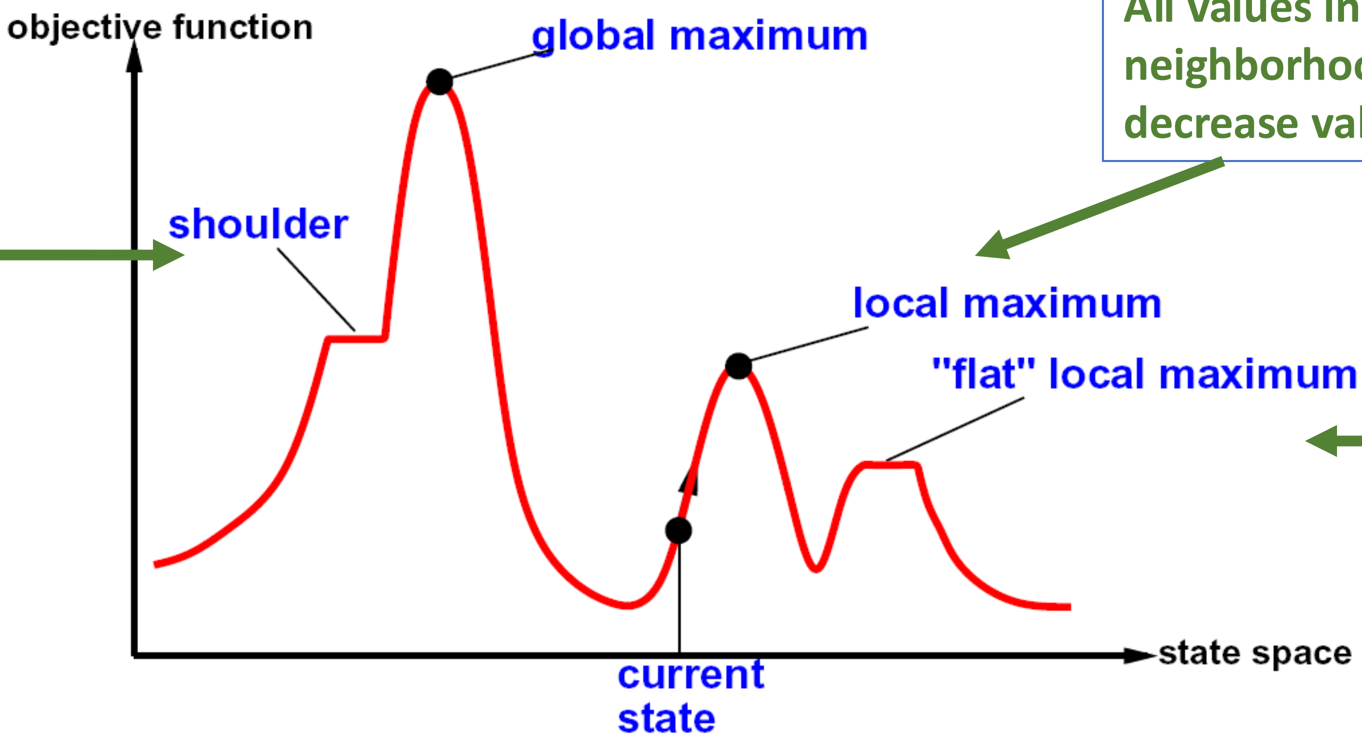
Issue: search reaches a solution where it cannot be improved - a local minimum.

Local minima (h = 1). Every successor has a higher cost.

Core Problem in Local Search

All values in the neighborhood decrease values

Neighborhood values are equivalent, need to search for a longer distance to find a path to a higher value



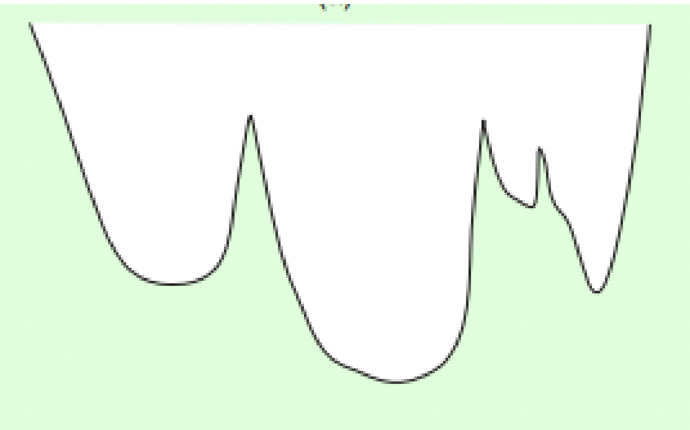
Neighborhood values are equivalent beyond which values only decrease

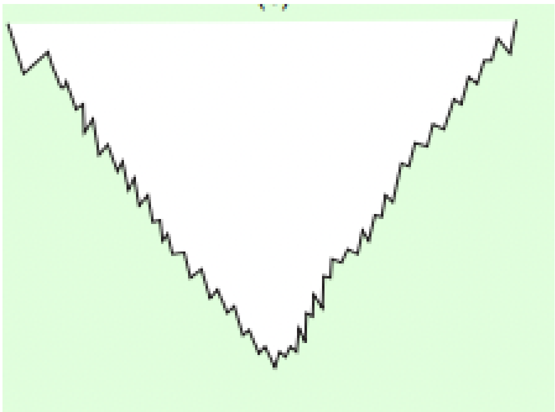
- Hill climbing prone to local maxima. Neighbors may not be of higher value. Search will stop at a sub-optimal solution
- Locally optimal actions may not lead to the globally optimal solution

Escaping local minima: Adding randomness

- **Random Re-starts**
  - A series of searches from randomly generated initial states.
- **Random Walk**
  - Pick ”any” candidate move (whether improves the solution or not).

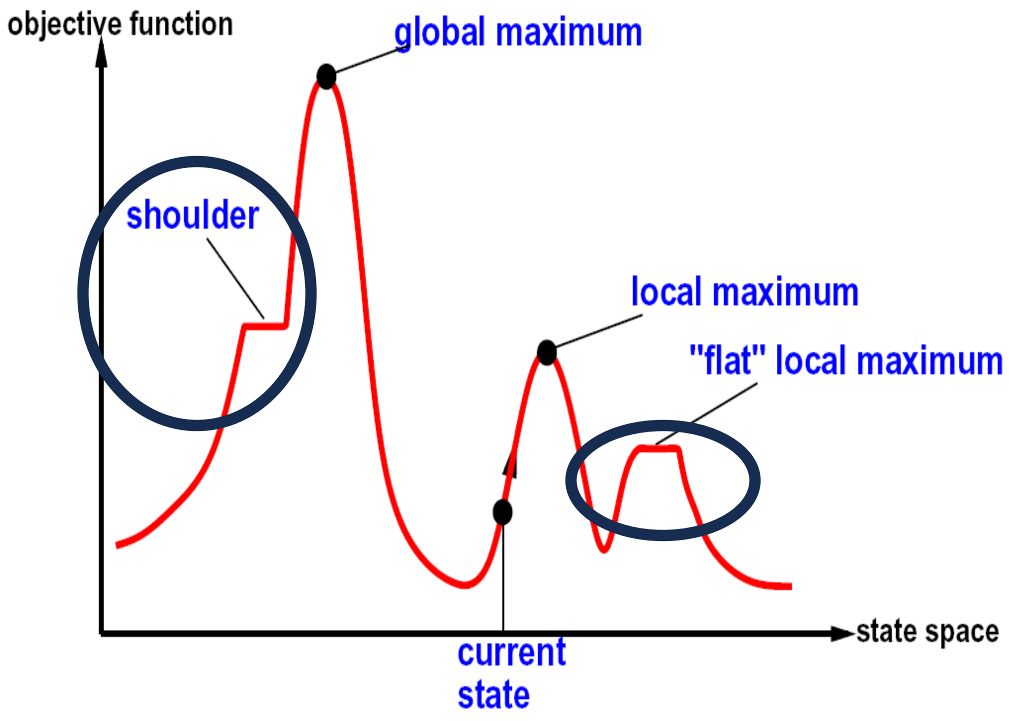
Q: Which method to use for the following cost surfaces? Random re-starts or random walk?





Escaping local minima: Adding randomness

- **Escaping flat local minima (shoulders)**
  - When local search reaches a flat area, that is, when all the neighbours have the same cost as the current state, it terminates right away
  - Keep moving - strategy
    - Make sideways moves for a few steps.



- ***Stochastic* Hill Climbing**
  - Instead of picking the *best move, pick any* move that produces an *improvement*.

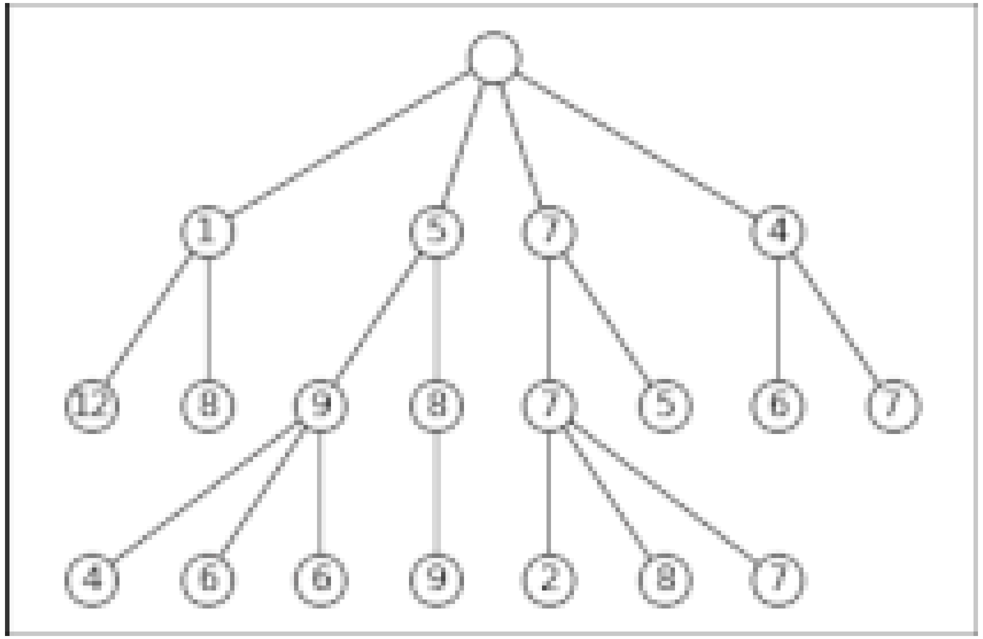
Looking for Solution from Multiple Points

- **Local Beam Search**
- Algorithm
  - Track k states (rather than 1).
  - Begin with k randomly sampled states.
  - Loop
    - Generate successors of each of the k-states
    - If anyone has the goal, the algorithm halts
    - Otherwise, select only the k-best successors from the list and repeat.

- Note:
  - Each run is <u>not</u> independent, information is passed between parallel search threads.
  - Promising states are propagated. Less promising states are not propagated.
  - Problem: states become concentrated in a small region of space.

Beam Search is a General Search Technique

- Beam search is a general idea (see right figure).
- Instead of considering all solutions at a level, consider only the top-k.
- Note: usually our memory is finite in size, there is an upper bound on the number of states that can be kept.
- In general, it is an approximate search method.

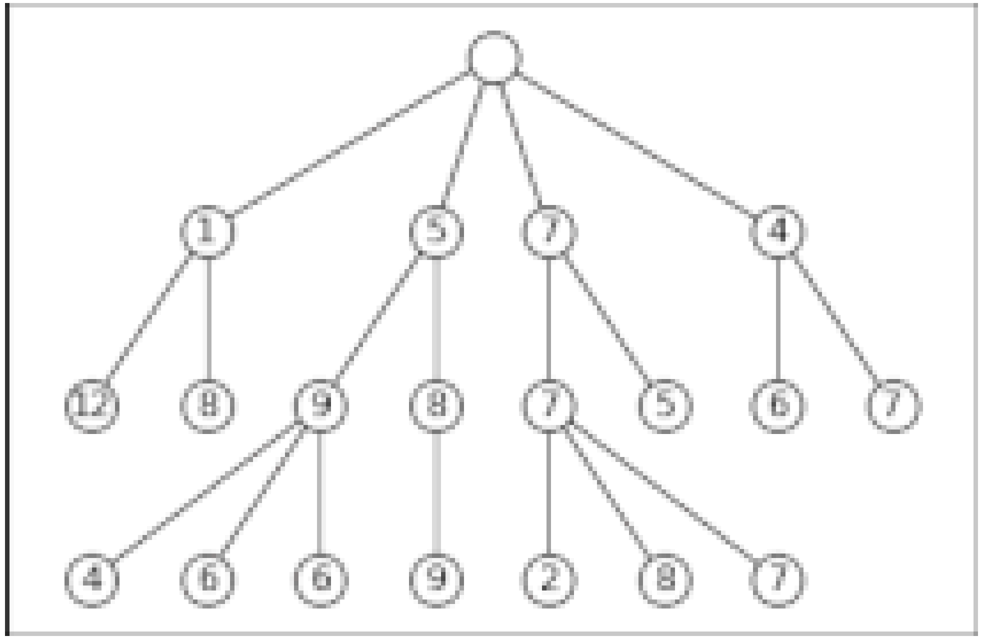


Beam search is a general idea. Here, shown in the context of a tree search. Beam size is 3. For local search we don't construct the full tree.

Source: wikipedia

"Stochastic" Beam Search

- **Local beam search**
  - Problem: states become concentrated in a small region of space
  - Search degenerates to hill climbing
- ***Stochastic* beam search**
  - Instead of taking the best k states
  - Sample k states from a distribution
  - Probability of selecting a state *increases* as the *value* of the state.



Instead of top k, sample k given a probability.

Source: wikipedia

Simulated Annealing

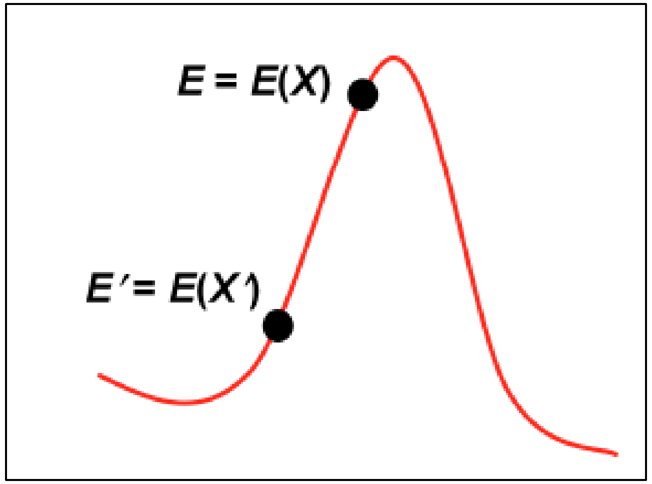
- In case of an improving move – move there.
- But allow some apparently **bad moves** - to escape local maxima.
- **Decrease** the size and the frequency of bad moves over time.

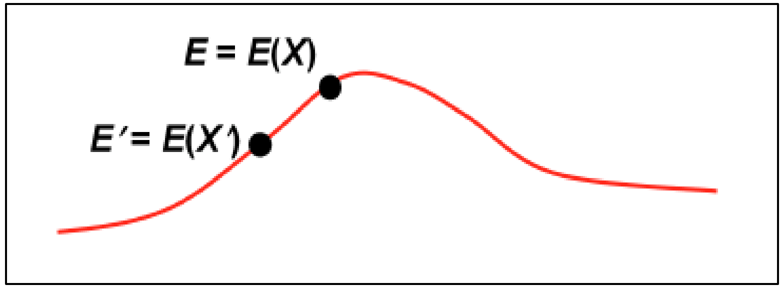
- Algorithm sketch
  1. Start at initial configuration X of value E (high is good)
  2. Repeat:
     (a) Let X_i be a *random neighbor* of X and E_i be its value
     (b) If $E < E_i$ then let $X \leftarrow X_i$ and $E \leftarrow E_i$
     (c) Else, *with some probability* p, still accept the move: $X \leftarrow X_i$ and $E \leftarrow E_i$
- Best solution ever found is always remembered

A form of Monte-Carlo Search. Move around the environment to explore it instead of systematically sweeping. Powerful technique for large domains.

Simulated Annealing: How to decide *p*?

- Considering a move from state of value E to a lower valued state of E'. *That is considering a sub-optimal move (E is higher than E').*
- If (E – E') is large:
  - Likely to be close to a promising maximum.
  - Less inclined to to go downhill.
- If (E – E') is small:
  - The closest maximum may be shallow
  - More inclined to go downhill is not as bad.





Simulated Annealing: Selecting Moves

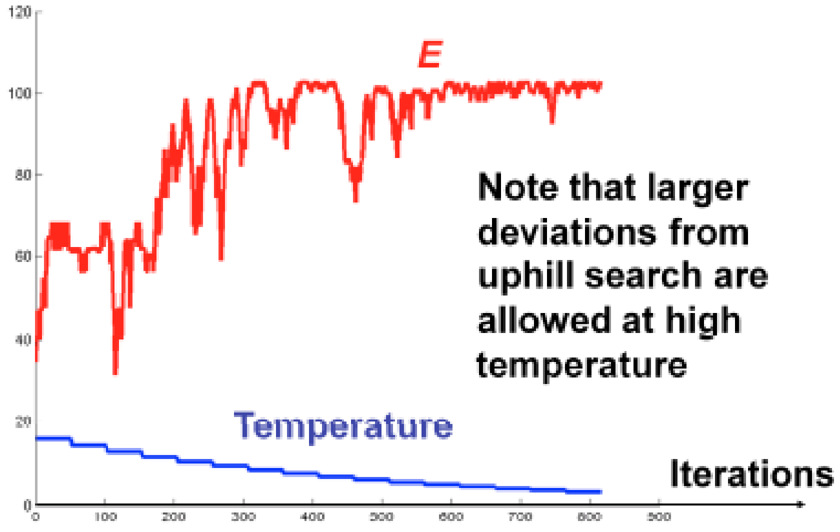
- If the new value E_i is ***better*** than the old value E, move to X_i
- If the new value is ***worse*** (E_i < E) then move to the neighboring solution as per *Boltzmann* distribution.

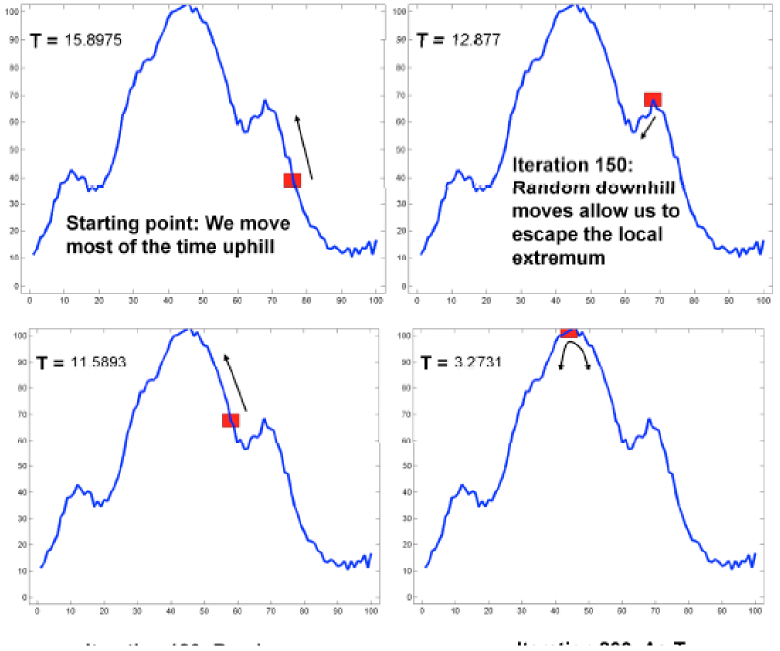
$$\exp\left(-\frac{E-E_i}{T}\right)$$

- Temperature (T>0)
  - **T is high**, exp is ~0, acceptance probability is ~1, high probability of acceptance of a worse solution.
  - **T is low**, the probability of moving to a worse solution is ~ 0, low probability of acceptance of a worse solution.
  - Schedule T to reduce over time.

Simulated Annealing

- **T is high**
  - The algorithm is in an *exploratory* phase
  - Even bad moves have a high chance of being picked
- **T is low**
  - The algorithm is in an *exploitation* phase
  - The “bad” moves have very low probability
- **If T is decreased slowly enough**
  - Simulated annealing is guaranteed to reach the best solution in the limit.



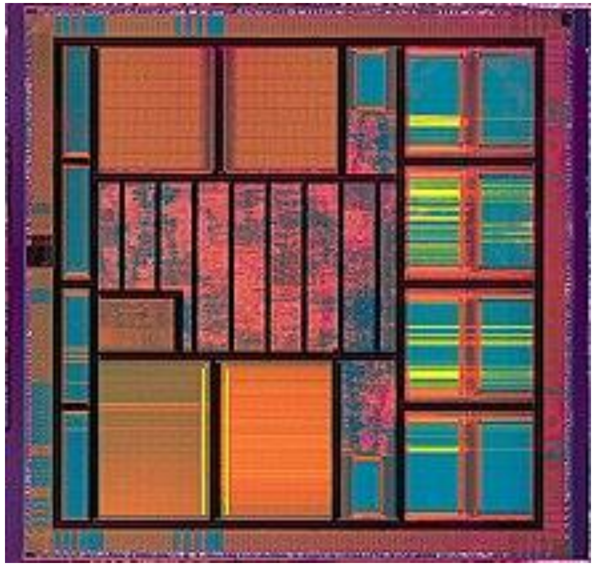


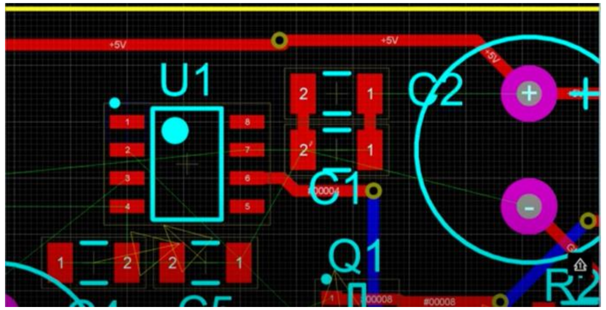
Able to escape local maxima.

Adding (some) memory: Tabu Search

- Local search loses track of the global cost landscape.
  - May frequently come back to the same state
- Introduce "memory" to prevent re-visits.
  - Maintain a finite-sized "tabu" list which remembers recently visited states so that one does not go towards them.
  - If a state proposed in the neighbourhood is in the tabu list do not go.

Motivating example: PCB layout with lower wire overlaps.





Search with Memory

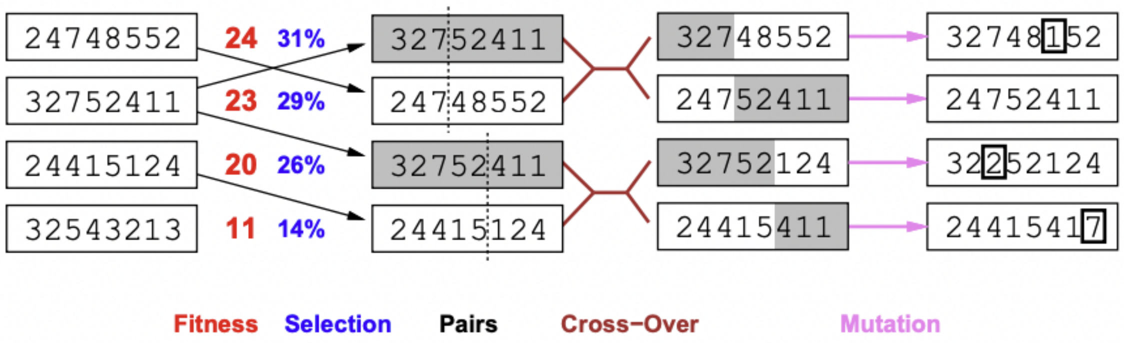
- Tabu Search
  - Maintain a tabu list of the k last assignments.
  - Don't allow an assignment that is already on the tabu list.
  - If k = 1, we don't allow an assignment of to the same value to the variable chosen.
  - Maintain a finite-sized tabu list (a form of local memory) which remembers recently visited states so that one does not go towards them.
  - Note: Tabu search allows for sub-optimal moves.
- Types of memory rules
  - Short-term: immediate states visited in the past.
  - Longer-term: guide the search towards certain regions of the search – where all have we explored in the past.
- Generalise to – searching locally by growing a tree for a short horizon and then picking a move (combining local and tree search).

Genetic Algorithms

- Idea
  - Variant of stochastic beam search: progression is by modifying a state.
  - Combine two states to generate the successor.
  - A mechanism to propose next moves in a different way.

- Ingredients
  - Coding of a solution into a string of symbols or bit-string
  - A fitness function to judge the worth of a state (or configuration)
  - A population of states (or configurations)

= stochastic local beam search + generate successors from **pairs** of states



Many variations:

how selection will be applied, what type of cross-over operators will be used, etc.

Selection of individuals according to a fitness function and pairing

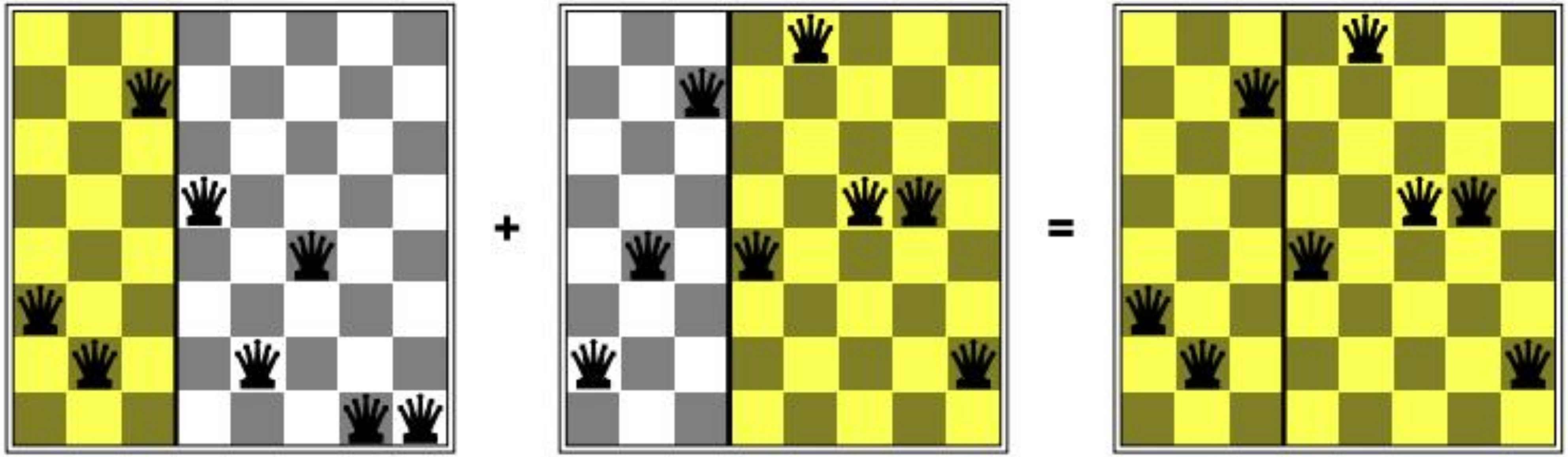
Calculation of the breaking points and recombination

According to a given probability elements in the string are modified.

Genetic Algorithms

https://rednuht.org/genetic_cars_2/

View as a way to propose moves – in an evolutionary way.



Advantage: ability to combine large blocks that evolved independently, impact the granularity of search.