APPLIED ALGORITHMS

SOLVING PROBLEM BY SEARCHING

Lecture 5: Local Search

T.S Ha Minh Hoang
Th.S Nguyen Minh Anh

Phenikaa University

Last Update: 6th March 2023

Recap

- CSP
- Backtracking
- Forward Checking and Heuristics

Backtracking search in the worst case performs an exhaustive DFS of the entire search tree, which can take a very very long time. How do we avoid this?

Outline

Search for the optimal configuration

Constraint satisfaction problem

- **Objective**: find a configuration that satisfies all constraints

Optimization problem

- **Objective**: find the best configuration that optimize (min/max) an **objective function**
- The **objective function**: reflects our preference towards each configuration (or state)

The search space

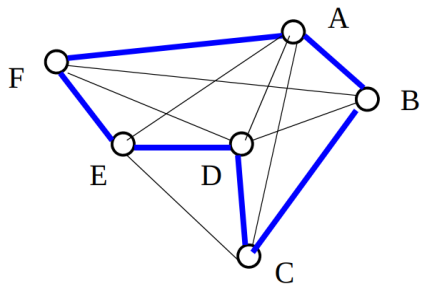
If the search space of configurations is

- **Discrete** or **finite**
  - then it is a **combinatorial optimization problem**
- **Continuous**
  - then it is a **parametric/continous optimization problem**

Example: Traveling salesman problem

Problem:

- A graph with distances
- Find tour - a path that visits every city once and returns to the start (e.g. ABCDEF)
- Goal: the shortest tour



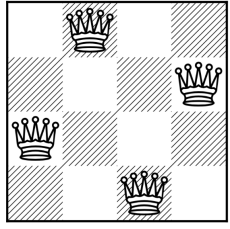
Example: Knapsack Problem

Problem:

- Given precious n items, each associated with a value
- Pick the items into your bag which has a capacity limitation
- Goal: maximizes the overall value of items in your bag

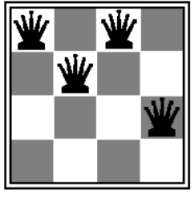
The N-Queens

- A CSP problem
- Is it possible to formulate the problem as an **optimization problem**?

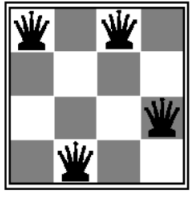


The N-Queens

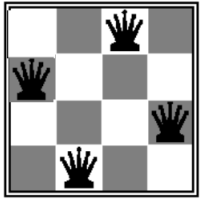
- A CSP problem
- Is it possible to formulate the problem as an **optimization problem**? **Yes**
- The **quality of a configuration in a CSP** can be measured by **the number of violated constraints**
- **Solving**: **minimize** the *number of constraint violations*



# of violations =3



# of violations =1



# of violations =0

Outline

Motivation

Some issues of the search algorithms we have discussed so far:

- The search algorithms we have seen so far include systematic search (breadth-first, depth-first, iterative deepening, etc.) where we look at the entire search space in a systematic manner till we have found a goal (or all goals, if we have to).
- We also have seen heuristic search (best-first, A*-search) where we compute an evaluation function for each candidate node and choose the one that has the best heuristic value (the lowest heuristic value is usually the best).
- In both systematic search and heuristic search, we have to keep track of what's called the frontier (some books call it open or agenda) data structure that keeps track of all the candidate nodes for traversal in the next move.
- We make a choice from this frontier data structure to choose the next node in the search space we go to.
- We usually have to construct the path from the start node to the goal node to solve a problem.

Motivation

- In many optimization problems, we need to find an optimal path to the goal from a start state (e.g., find the shortest path from a start city to an end city), but in many other optimization problems, the path is irrelevant.
- The goal state itself is the solution.
- Here, it is not important how we get to the goal state, we just need to know the goal state.
- **Idea:** iterative improvement algorithms: keep a single "current" state, and try to improve it.

Backtracking vs Local Search

Backtracking: extend partial assignments

Local search: modify complete assignments

Local Search Algorithms

- State space = set of "complete" configurations
- Keep a single "current" state, try to improve it.
- Very memory efficient: only remember current state

Example: N-queens

Generate possible neighbors for the following state: 1 relocate

Example: N-queens

Generate possible neighbors for the following state: 2 relocate

Components of a local search

Some issues of the search algorithms we have discussed so far

Properties of a local search

Some issues of the search algorithms we have discussed so far

Outline

Hill climbing (or gradient ascent/descent)

- This is a very simple algorithm for finding a local maximum
- Iteratively maximize "value" of current state, by replacing it by successor state that has highest value, as long as possible.
- In other words, we start from an initial position (possibly random), the move along the direction of steepest ascent/gradient (descent/negative of gradient, for minimization), go along the direction for a little while; and repeat the process.

Some ideas on making Hill-climbing better

- Allowing sideways moves
  - When stuck on a ridge or plateau (i.e., all successors have the same value), allow it to move anyway hoping it is a shoulder and after some time, there will be a way up.
- How many sideways moves?
  - If we always allow sideways moves when there are no uphill moves, an infinite loop may occur if it's not a shoulder.
- Solution
  - Put a limit on the number of consecutive sideways moves allowed
- Experience with a real problem
  - The authors allowed 100 consecutive sideways moves in the 8-queens problem.
  - This raises the percentage of problems solved from 14% to 94%
  - However, now it takes on average 21 steps when successful and 64 steps when it fails.

Variants of Hill Climbing

Outline

Outline