



Contents

Classical search: uninformed search and informed search

Search with non-determinism/partial observation

Local search

Constrained satisfaction problem

Adversarial search for games

Lecture 09: Finding Optimal Configurations

CSCI 404 Artificial Intelligence, Spring 2018



Hua Wang, Ph.D.

Department of Computer Science
Colorado School of Mines

February 08, 2018



Beyond Classical Search

Problems/Restrictions in Classical Searches

The classical search algorithms studied so far explore the state spaces systematically, which thereby use too much memory!

To address these problems/restrictions, we are going to study:

Local search



Beyond Classical Search

Problems/Restrictions in Classical Searches

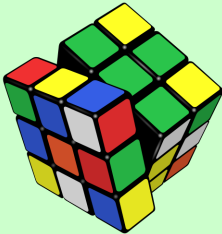
The classical search algorithms studied so far explore the state spaces systematically, which thereby use too much memory!

To address these problems/restrictions, we are going to study:

Local search

Two types of search problems

Rubik's Cube

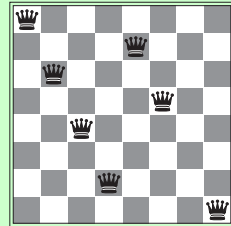


Start state is given

Goal state is known ahead of time

Solution path matters

N-Queens Problem



No specific start state

Goal state is unknown (only have goal test)

Solution path does not matter



Local search algorithms

If the path to the goal does not matter, we may consider local search algorithms:

Operates using a single current node (rather than multiple paths), and generally move to only neighbors of that node.

Typically the paths followed by the search are **not** retained.

Advantages of local search

Local search algorithms are not systematic, but

Local search uses very little memory — usually a constant amount of memory;

they can find reasonably solutions in large or even infinite (continuous) state spaces, for which systematic algorithms are unsuitable.



Local search algorithms

If the path to the goal does not matter, we may consider local search algorithms:

Operates using a single current node (rather than multiple paths), and generally move to only neighbors of that node.

Typically the paths followed by the search are **not** retained.

Advantages of local search

Local search algorithms are not systematic, but

Local search uses very little memory — usually a constant amount of memory;

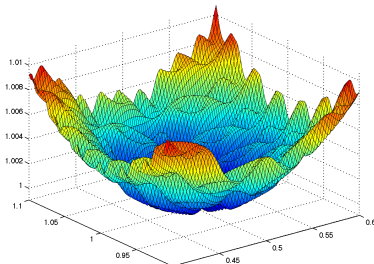
they can find reasonably solutions in large or even infinite (continuous) state spaces, for which systematic algorithms are unsuitable.

Local search algorithms

Some types of search problems can be formulated in terms of optimization

We don't have a start state, don't care about the path to a solution, can move around state space arbitrarily.

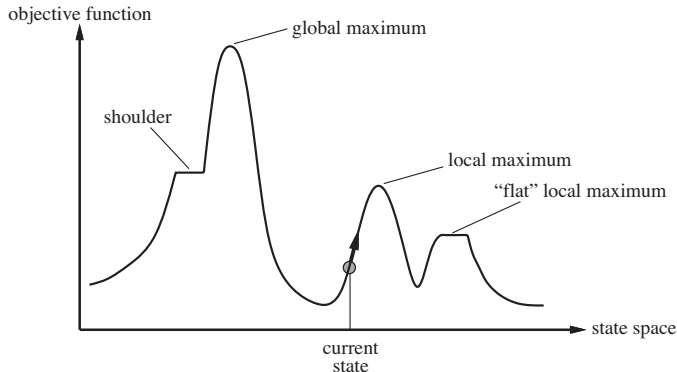
We have an objective function that tells us about the quality of a state (possible solution), and we want to find a good solution by minimizing or maximizing the value of this function.



State-space landscape

If the evaluation corresponds to cost, we aim to find the lowest value — a global minimum.

If the evaluation corresponds to objective function, we aim to find the highest peak — a global maximum.



Example: n -queens problem

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

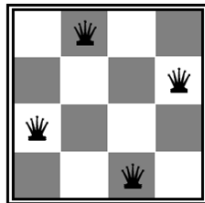
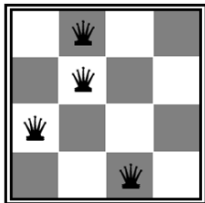
State space: all possible n -queen configurations

How big is the state space when $n = 8$?

What's the **objective function**?

Number of pairwise conflicts

Ideal case to be 0 conflict



Example: n -queens problem

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

State space: all possible n -queen configurations

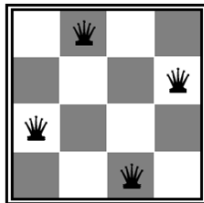
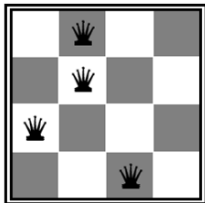
How big is the state space when $n = 8$?

$$64 \times 63 \times \cdots \times 57 \approx 1.8 \times 10^{14}!$$

What's the **objective function**?

Number of pairwise conflicts

Ideal case to be 0 conflict



Example: Traveling salesman problem

Find the shortest tour connecting a given set of sites

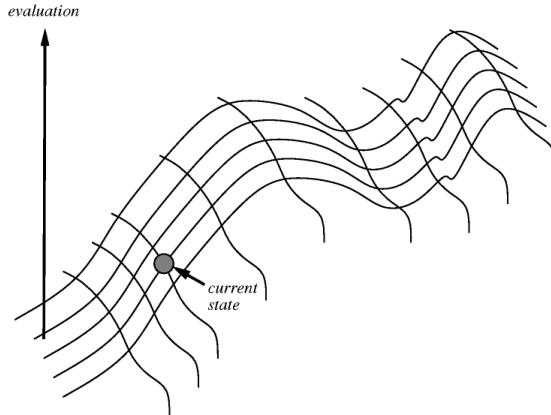
State space: all possible tours

Objective function: length of tour



Hill-climbing (greedy) search

Idea: keep a single “current” state and try to locally improve it
“like climbing mount Everest in thick fog with amnesia”

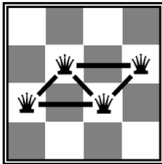


Example: n -queens problem

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal. State space: all possible n -queen configurations

Objective function: number of pairwise conflicts

What's a possible local improvement strategy?



$h = 5$

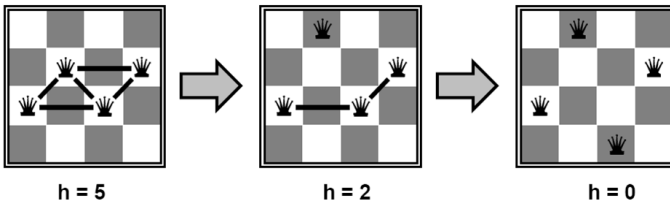
Example: n -queens problem

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal. State space: all possible n -queen configurations

Objective function: number of pairwise conflicts

What's a possible local improvement strategy?

Move one queen within its column to reduce conflicts.



Example: n -queens problem

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal. State space: all possible n -queen configurations

Objective function: number of pairwise conflicts

What's a possible local improvement strategy?

Move one queen within its column to reduce conflicts.

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♔	13	16	13	16
♔	14	17	15	♔	14	16	16
17	♔	16	18	15	♔	15	♔
18	14	♔	15	15	14	♔	16
14	14	13	17	12	14	12	18

The number of conflict is 17. We should move to the configuration along the steepest-descent direction — the one of 12.

Example: Traveling Salesman Problem

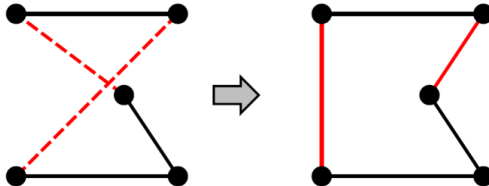
Find the shortest tour connecting n cities

State space: all possible tours

Objective function: length of tour

What's a possible local improvement strategy?

Start with any complete tour, exchange endpoints of two edges





Hill-climbing (greedy) search (maximization)

Hill-climbing (greedy) search

Initialize *current* to starting state

Loop:

Let *next* = highest-valued successor of *current*

If $\text{value}(\text{next}) < \text{value}(\text{current})$ return *current*

Else let *current* = *next*



Hill-climbing search


Is it complete/optimal?


Hill-climbing search

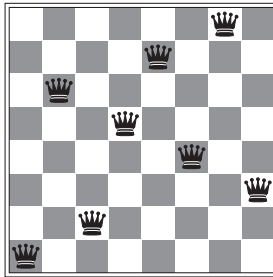
Is it complete/optimal?

No — can get stuck in local optima

Example: local optimum for the 8-queens problem

Cons : Starting from a randomly generated 8-queens state, steepest-ascent hill climbing gets stuck **86%** of the time, solving only **14%** of problem instances.

Pros : It works quickly, taking just **4** steps on average when it succeeds and **3** when it gets stuck — not bad for a state space with $8^8 \approx 17$ **million** states.



$h = 1$: conflicting at $\langle 4, 7 \rangle$

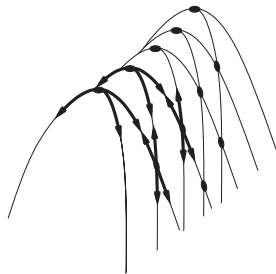
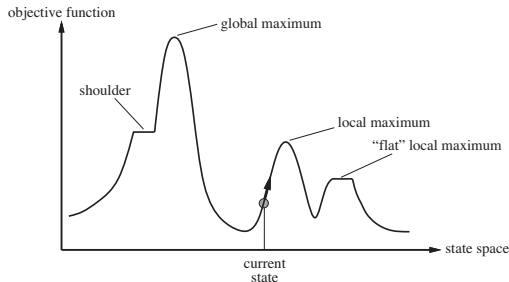
State-space landscape

Hill climbing often get stuck for the following reasons:

Local maxima.

Plateaux — flat local maxima, shoulder.

Ridge.



How to escape from getting stuck?

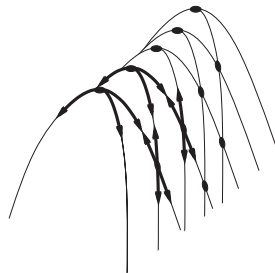
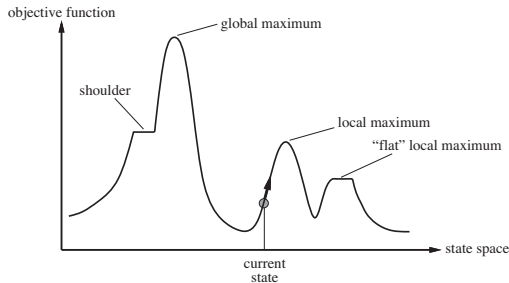
State-space landscape

Hill climbing often get stuck for the following reasons:

Local maxima.

Plateaux — flat local maxima, shoulder.


Ridge.




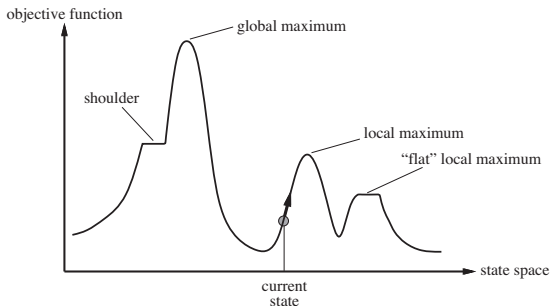
How to escape from getting stuck?

Remedy 1: Sideways move for hill-climbing algorithm

For plateaux, sideways move could help, in the hope that the plateau is a shoulder.

Pros : This raises the percentage of problem instances solved by hill climbing from **14%** to **94%** — Success comes at a cost: the algorithm averages roughly **21** steps for each successful instance and **64** for each failure.

Cons : If we always allow sideways moves when there are no uphill moves, an infinite loop will occur whenever the algorithm reaches a flat local maximum that is not a shoulder.



Remedy 2: stochastic hill climbing

Stochastic hill climbing chooses at random from among the uphill moves

the probability of selection can vary with the steepness of the uphill move.



This usually converges more slowly than steepest ascent,



but in some state landscapes, it finds better solutions.

Remedy 3: first-choice hill climbing

First-choice hill climbing implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state.



This is a good strategy when a state has many (e.g., thousands) of successors.



Remedy 4: random-restart hill climbing

Random-restart hill climbing adopts the well-known adage, “If at first you don’t succeed, try, try again.”

It conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.

It is trivially complete with probability approaching 1, because it will eventually generate a goal state as the initial state.



Copyright statement

This slide set is designed for the teaching purpose only in the course of CSCI-404 Artificial Intelligence at the Department of Computer Science of Colorado School of Mines in Spring 2018.

Some contents in this slide set are obtained from Internet and maybe copyright sensitive. Copyright and all rights therein are retained by the respective authors or by other copyright holders. Distributing or reposting the whole or part of this slide set not for the teaching purpose is **HIGHLY** prohibited, unless explicit permissions from all copyright holders are granted.