Coping With NPC

Heuristics

Class 41

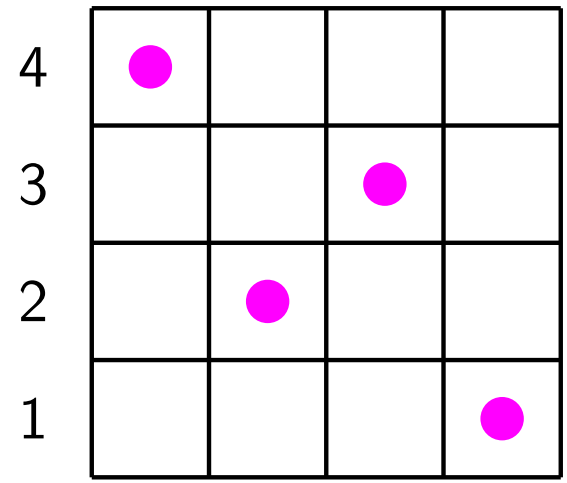
Heuristic

- from Greek *heuriskein*: to discover
- Archimedes' famous cry: "Heureka! I have found it!"

- in computer science, a non-deterministic algorithm or algorithm strategy
- goal: an optimal or near-optimal solution in polynomial time
- but because it is non-deterministic:
  1. polynomial time is not guaranteed
  2. the true optimal solution is not guaranteed

- approximation and heuristic algorithms are different
- approximation algorithms are deterministic

n-Queens



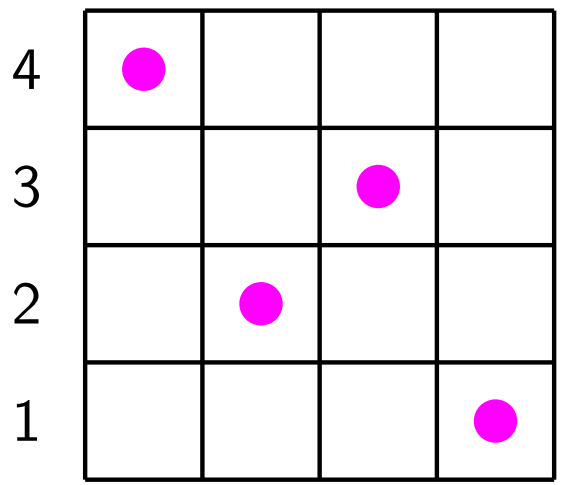
- an arrangement of queens can be represented as an n-tuple
- each number gives the row of the queen in the respective column
- this arrangement is

$$P = (4, 2, 3, 1)$$

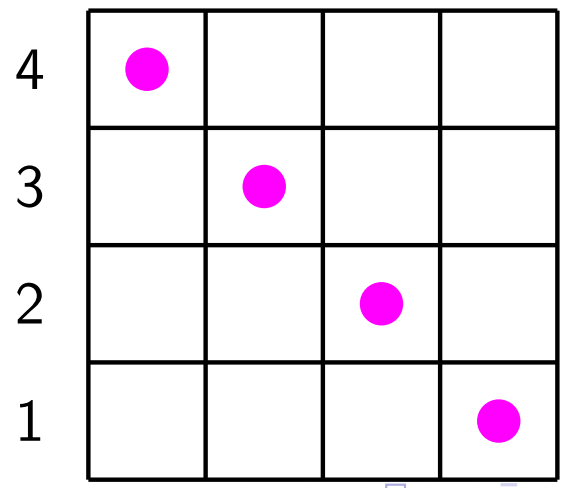
Collisions

- we can easily place queens in a way that precludes any row and column conflicts
- thus only diagonals can generate conflicts, or collisions
- n-queens is the search for an arrangement with no collisions
- we use collisions as the metric of how good a proposed solution is
- $k > 0$ queens in the same diagonal gives $\frac{k(k-1)}{2}$ collisions

2 collisions



6 collisions



Permutations

- only one queen is allowed per row
- each value in the n-tuple must appear exactly once
- thus an arrangement of n queens is represented by a permutation of the values $1, \ldots, n$
- on the previous page we have

$$P_1 = (4, 2, 3, 1)$$

and

$$P_2 = (4, 3, 2, 1)$$

Observations

- a difference between permutations represents column swaps
- swapping columns can have a huge effect on the number of collisions
- we use these observations to propose a heuristic algorithm

A Heuristic Algorithm

```
do
{
  p = a random permutation
  do
  {
    swaps = 0;
    for i in 1 .. n - 1
    {
      for j in i .. n
      {
        consider swapping i & j in p
        if collisions with swap < collisions without swap
        {
          swap(i, j)
          swaps++
        }
      }
    }
  } while swaps > 0
} while collisions > 0
```

Demonstration

run backtracking nqueens vs heuristic nqueens

General Heuristic

- generally, a heuristic algorithm is

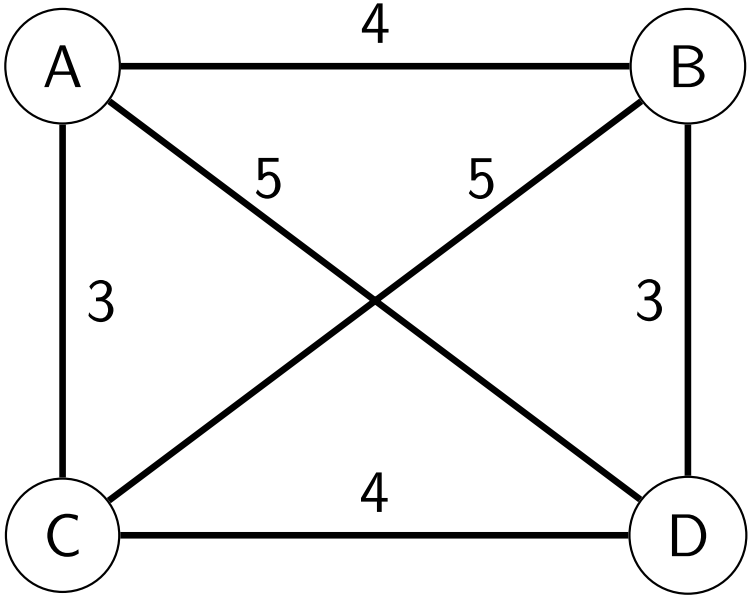
```
repeat
{
  s = random initial arrangement

  repeat
  {
    choose random s' near s
    if s' is better than s:
      replace s by s'
  } while still making progress
} until answer is close enough
```

Heuristic Algorithm for TSP

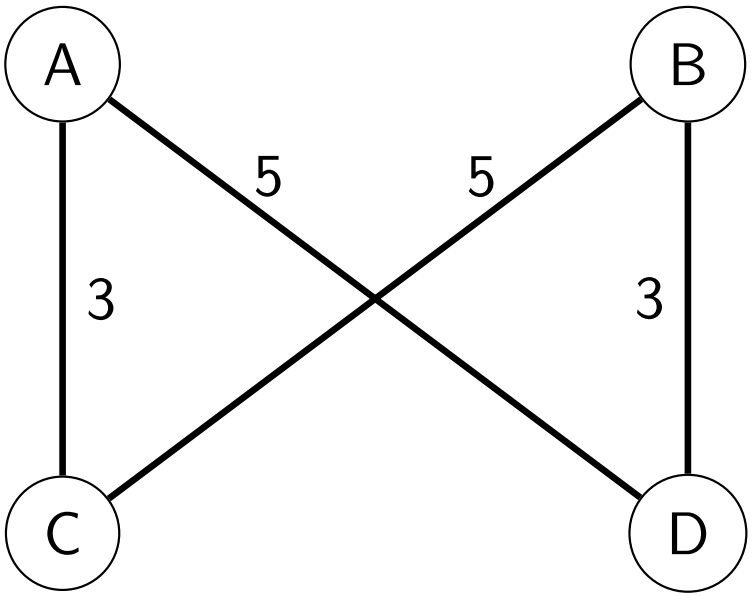
- assume a complete graph (true for many road and airline networks)
- thus there are $(n-1)!$ possible circuits
- randomly pick a circuit
- pick two edges and delete them from the circuit
- replace them with two different edges
- if the change is better, keep it; if not discard it
- repeat

TSP K_4 Example



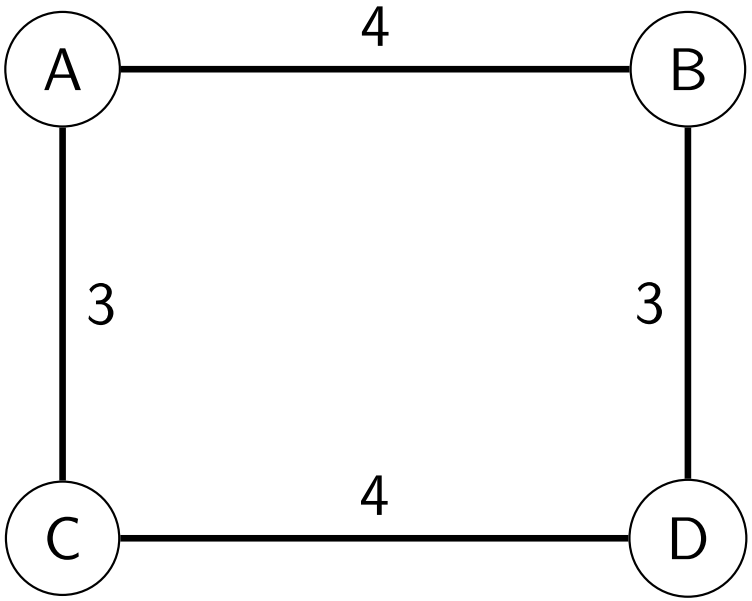
- start with K_4

TSP K_4 Example



- start with K_4
- pick a circuit at random
  $5 + 3 + 5 + 3 = 16$
  optimal? no

TSP K_4 Example



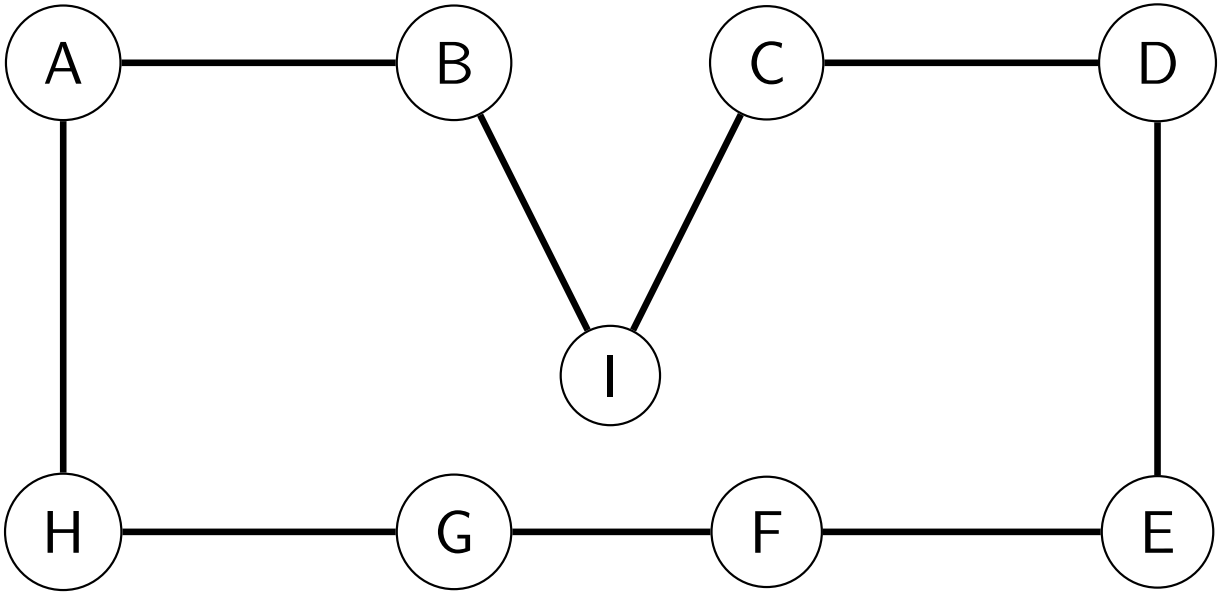
- start with K_4
- pick a circuit at random
  $5 + 3 + 5 + 3 = 16$
  optimal? no
- replace 2 edges with 2 others to make a different circuit
  $4 + 3 + 4 + 3 = 14$
  optimal? yes

Solution

- all circuits reachable from original circuit by changing 2 edges are in the 2-change neighborhood of the original
- how many circuits are in the 2-change neighborhood of K_4?
- K_4 is so simple that all possible circuits are in the 2-change neighborhood of any circuit, so the heuristic algorithm will generate an exact solution

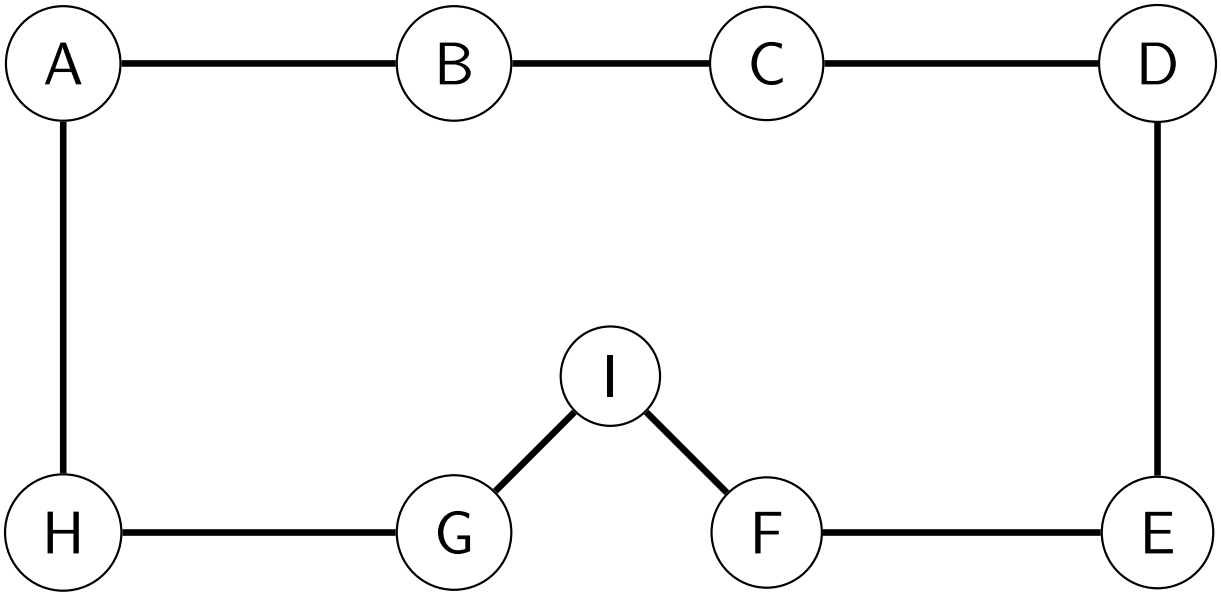
Heuristic Algorithm for TSP

- in general, an arbitrary circuit has $O(n^2)$ other circuits in its 2-change neighborhood



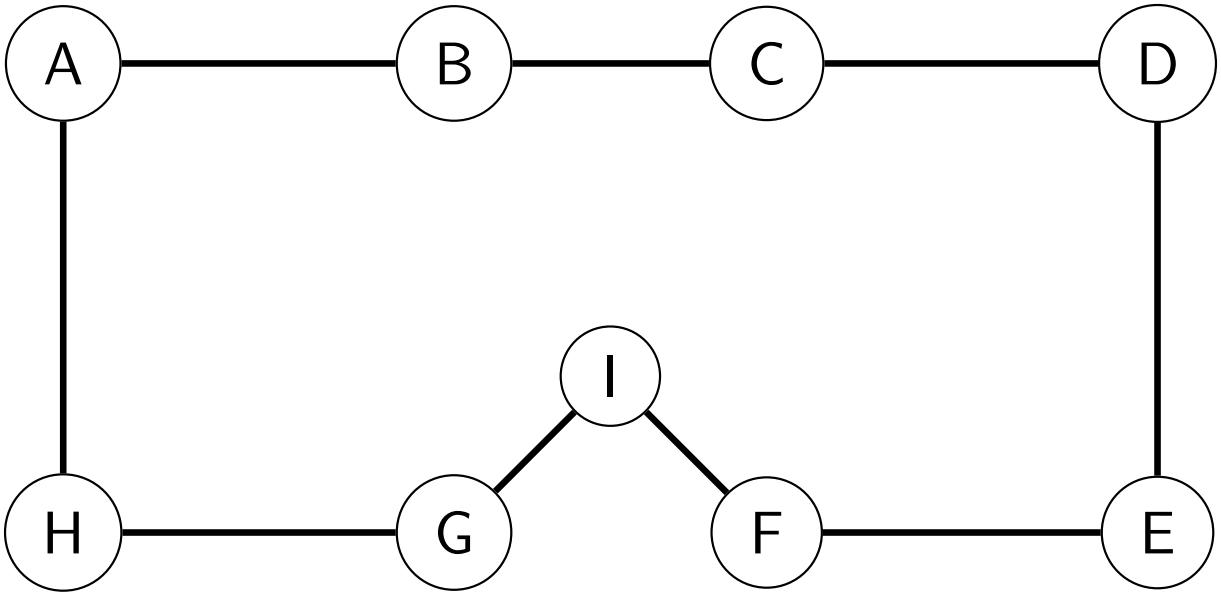
- what is the optimal TSP circuit for this graph?
- is it in the 2-change neighborhood of this circuit?

Heuristic Algorithm for TSP



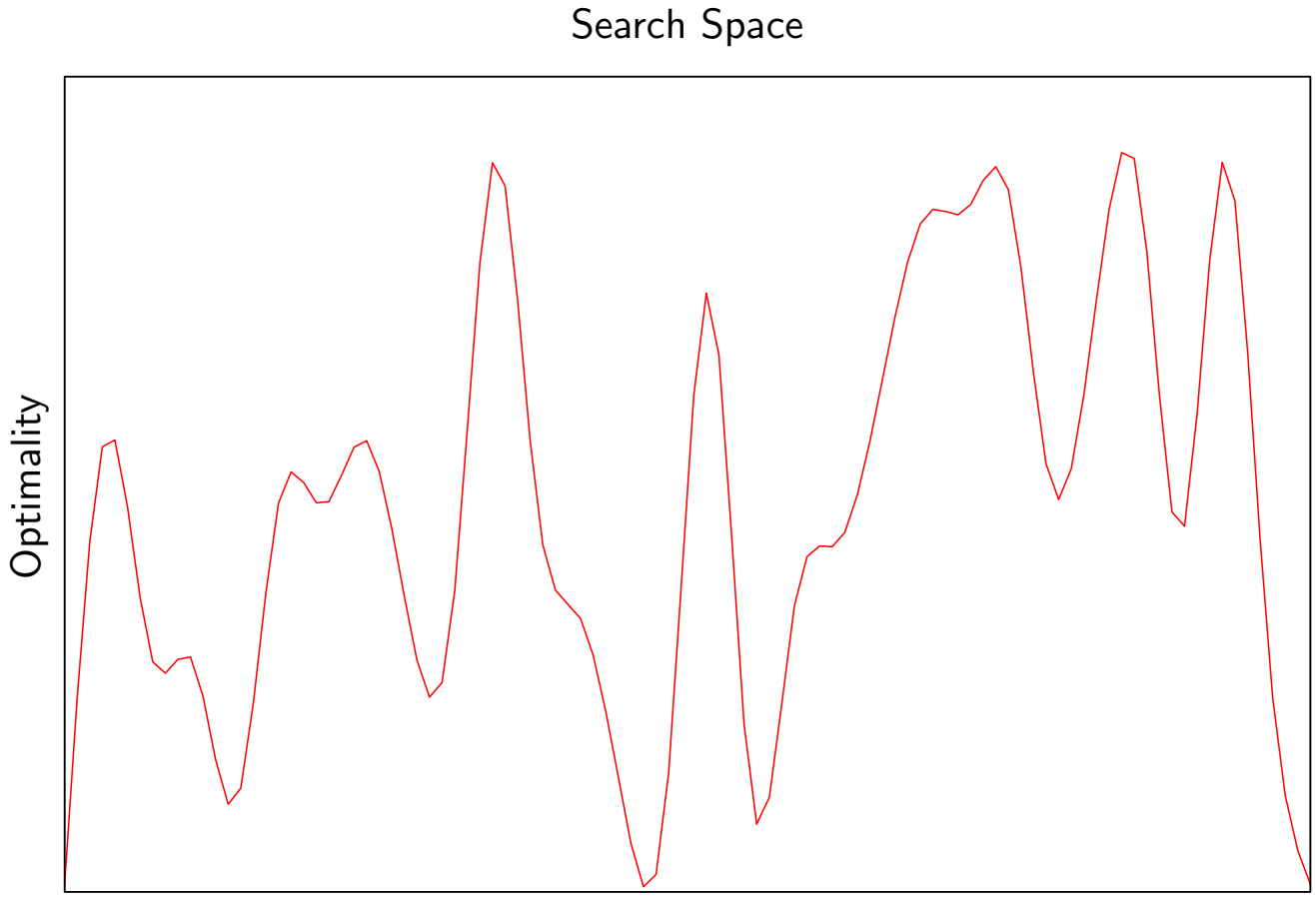
- the optimal TSP circuit is in the 3-change neighborhood of the original
- there are $O(n^3)$ circuits in a circuit's 3-change neighborhood

Heuristic Algorithm for TSP



- the optimal TSP circuit is in the 3-change neighborhood of the original
- there are $O(n^3)$ circuits in a circuit's 3-change neighborhood
- there is a trade-off between speed and quality of solution

Search Space



we can envision the search space as a landscape

Simple Heuristic

- for some shapes of the search space, the simple heuristic algorithm works well
- if the probability of reaching a good local optimum on any given iteration is p
- within $1/p$ iterations a good local optimum will be found at least 50% of the time

- however, this is not always the case
- with some problem types, as the problem size grows, the ratio of bad to good local optima grows
- sometimes ratio of bad to good becomes exponentially large

Metropolis Algorithm

- 1953
- iteratively generate a sequence of sample values
- as subsequent sample values are produced, the distribution of values more closely approximates the desired distribution
- the sequence of values is a Markov chain (dependent only on the current system state)

Metropolis Algorithm

- start with initial arrangement s and constant T

```
repeat:
  choose random s' near s
  if s' is better than s:
    replace s by s'
  else:
    replace s by s' with probability p
```

- where p is calculated from T and how much worse s' is than s
- $p = e^{-\triangle/T}$
- $\triangle =$ badness of s' compared to s

Replacement Probability

$$p = e^{-\triangle/T}$$

- what happens if the denominator T is large?
- what happens if the denominator T is small?

Replacement Probability

$$p = e^{-\triangle / T}$$

- what happens if the denominator T is large?
  the probability is close to 1
- what happens if the denominator T is small?
  the probability is close to 0

Metropolis

- you can think of T as a temperature slider
- if T is large (high temperature) the probability of accepting a worse move is high and Metropolis becomes a random walk indifferent to cost
- if T is small (low temperature) the probability of accepting a worse move is small and Metropolis becomes equivalent to simple heuristic finding the closest local optimum
- the art comes in "tuning" T to find a relatively good local optimum

Annealing

- when a liquid is cooled, crystals form
- if the temperature is dropped rapidly, crystals form quickly and irregularly, in parallel
- annealing is the process of reducing the temperature very slowly, allowing crystals to form slowly, with one serving as a template for a regular lattice configuration of very low energy

Simulated Annealing

- simulated annealing modifies the Metropolis algorithm
- T is no longer constant
- instead start with T high and lower it slowly
- change the definition of "near"
- start with initial configuration s and large T

```
while T > 0:
  decrement T slightly
  choose random s' at distance from s proportional to T
  if s' is better than s:
    replace s by s'
  else:
    replace s by s' with probability p
```

- $\triangle =$ badness of s' compared to s
- $p = e^{-\triangle/T}$

Simulated Annealing

- in simulated annealing, the slow decrease in "temperature" represents a slow decrease in the probability of accepting worse solutions
- the algorithm explores the search space widely early on, slowly converging on a very good local optimum
- the art in tuning the algorithm is
  - the cooling schedule of T
    - slower cooling gives greater likelihood of finding global maximum
    - slower cooling gives much greater running times
  - how T influences the "neighborhood" from which s' is chosen