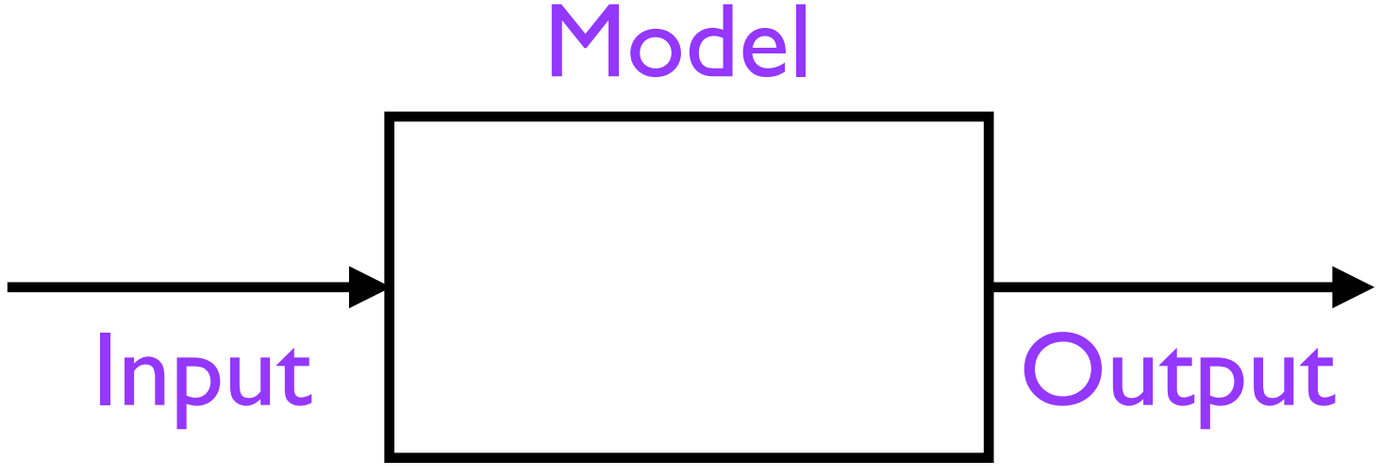
3 Problems To Be Solved

Types of Problem

- problems can be classified in different ways:
  1. black box model
  2. search problems
  3. optimisation vs constraint satisfaction
  4. NP problems
- we'll take a look at all of these

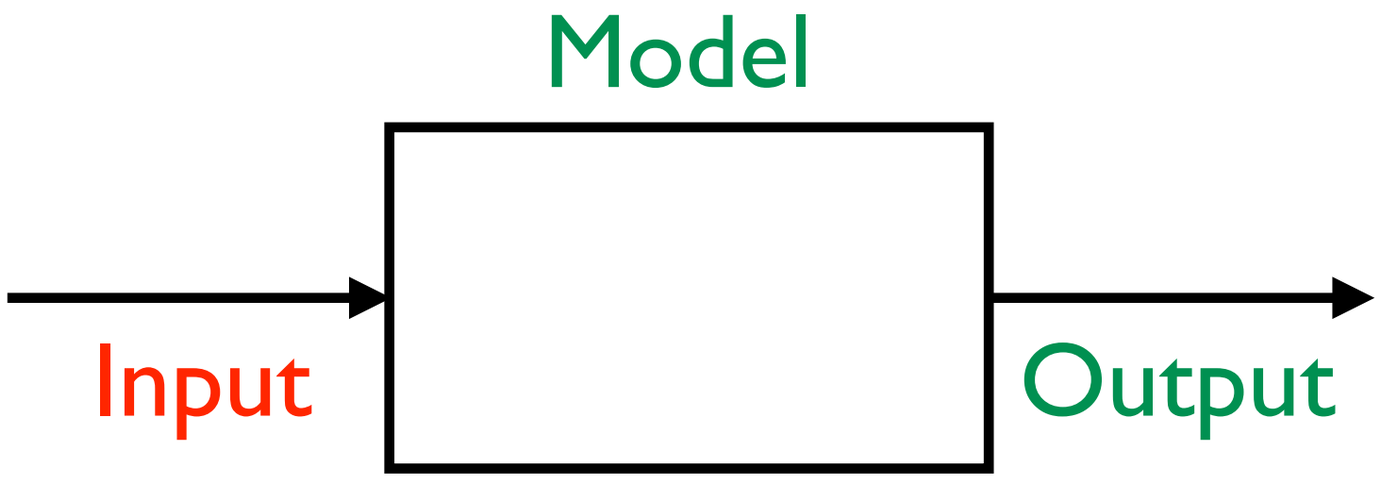
Black Box Model

- the model consists of 3 components:



- when one component is unknown we have a new problem type

Black Box Model: Optimization



- model and output are known
- *task:* to find the inputs which provide the desired output
- examples:
    - creating class schedules
    - travelling salesman problem (TSP)
    - shape of satellite antenna
    - eight-queens problem

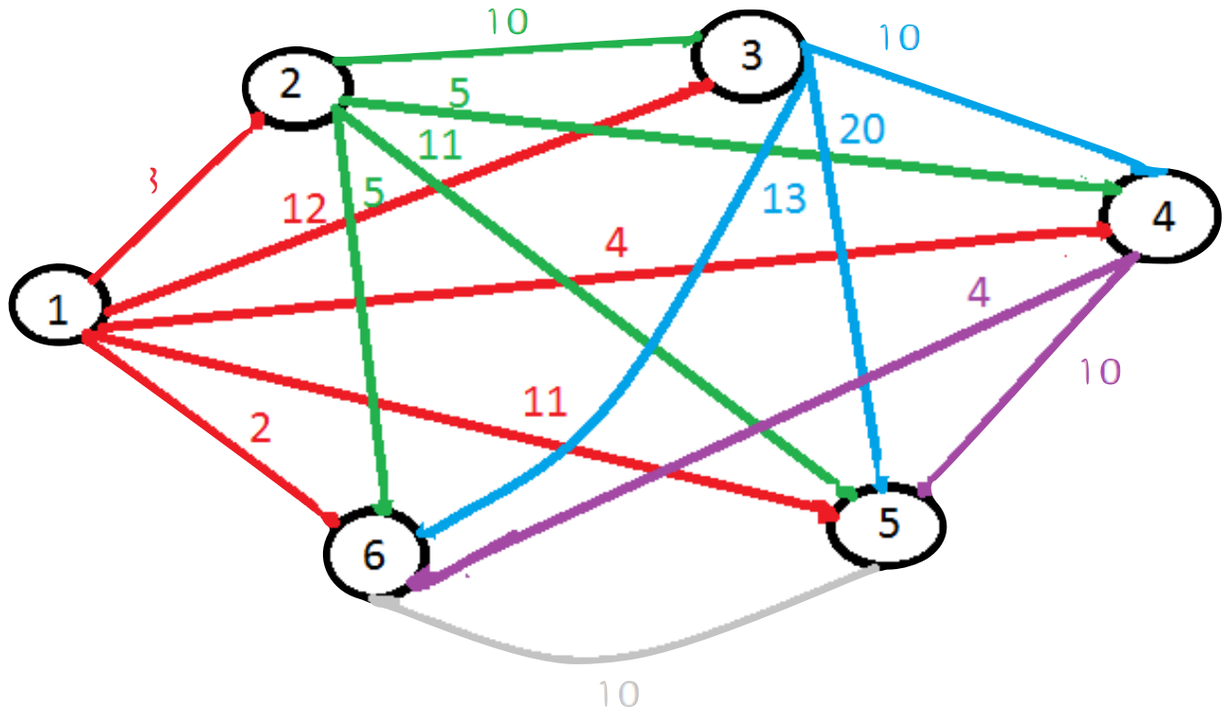
Optimization Example: University Class Scheduling

- huge search space
  - too big to exhaustively search
  - most ‘potential’ solutions in the search space are infeasible
- timetables must be ‘good’
  - minimizing clashes for courses normally taken together
  - minimizing travel distance between classes
  - ensuring room availability for class size
  - ensuring instructor availability
  - and so on

Optimization Example: travelling salesman problem

"Given a list of cities and the distances between each pair of cities, what is the shortest possible tour that visits each city and returns to the origin city?"

- number of possible tours is `n!`
- so exhaustive search is not feasible



Optimization Example: Space Technology 5 (ST5) Antenna

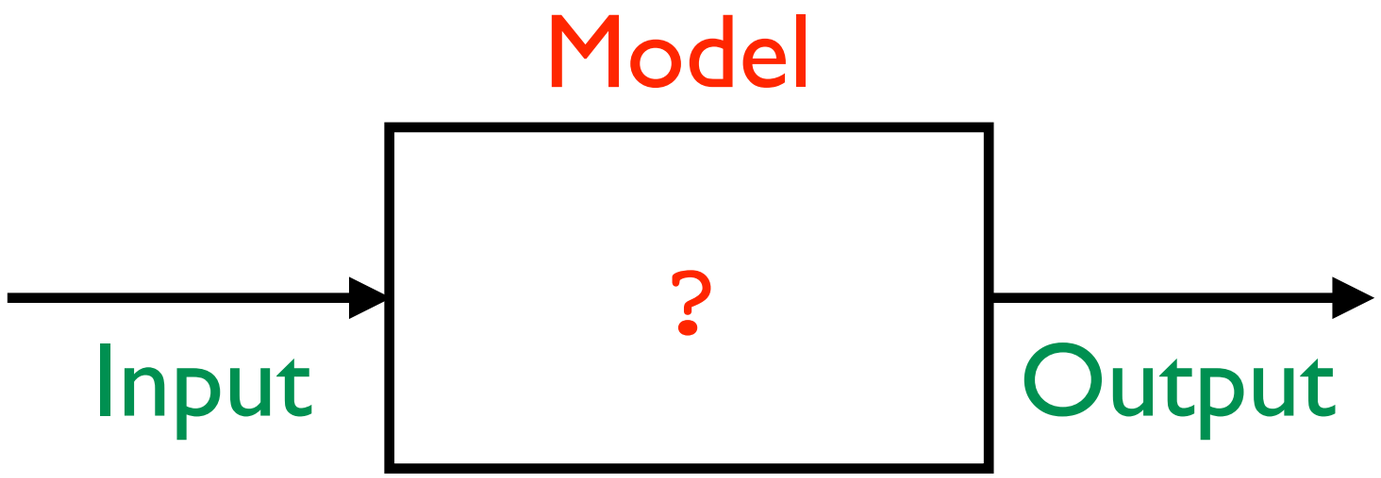
- 3 small satellites acting together (constellation)
- to test technologies for future space missions
- evolved shape of antenna
- fitness measure: gain
- shape discovered by an evolutionary algorithm running for days on a supercomputer

Optimization Example: 8 Queens Problem

- place 8 queens on the chessboard so that no 2 queens threaten each other
    - none share the same row, column or diagonal

(try it here)

Black Box Model: Modelling



- inputs and corresponding outputs are known
- *task:* to find a model that delivers the correct output for every known input
- *note:* modelling problems can be transformed into optimization problems
- examples:
    - loan applicant creditability rating
    - predicting the stock exchange

Modelling Example: Traffic Signs

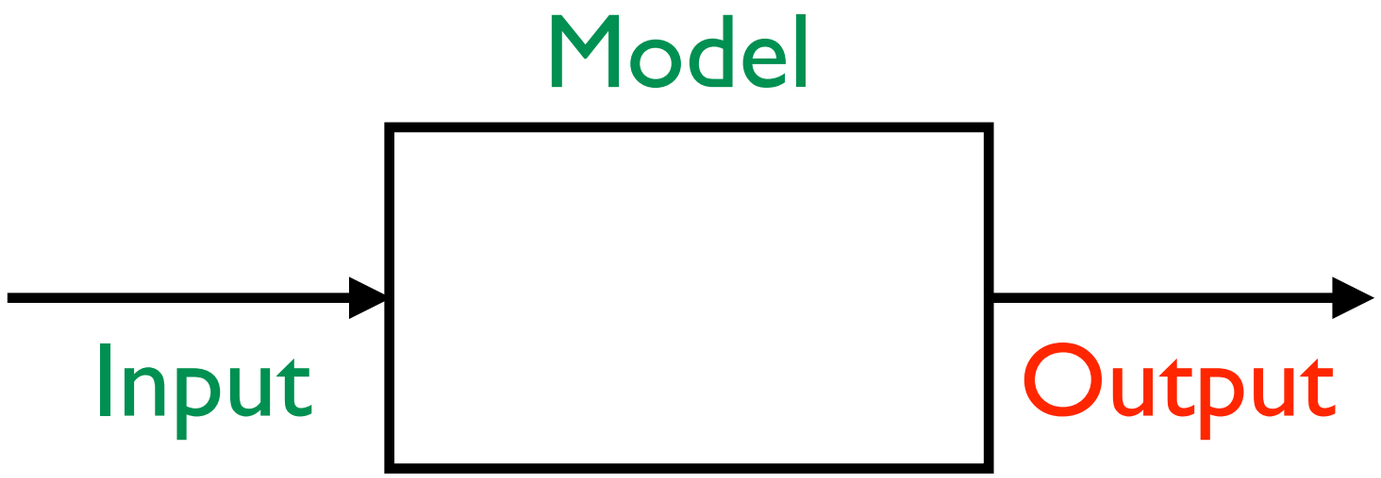
Modelling Example: Loan Applicant Creditability

- UK bank evolved creditability model to predict the loan paying behaviour of new loan applicants
- evolved the prediction models
- fitness measure: accuracy of the model when tested on historical data
    - a common technique

Modelling Example: Predicting the Stock Exchange

- Wilson & Banzhaf
- genetic program to predict prices of a number of stocks
- fitness measure: profit!
- see their paper in Brightspace (or here)

Black Box Model: Simulation

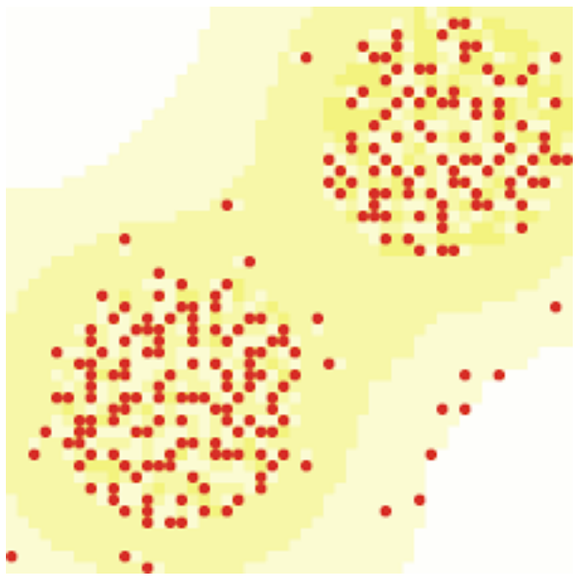


- model and sets of input conditions are known
- *task:* to find the outputs that arise for each set of input conditions
- often used to answer "what-if" questions in evolving dynamic environments
    - artificial societies
    - artificial life
        - we've seen some of this already with CA such as Evoloops
    - weather forecasting systems

Simulation Example: Artificial Societies

Sugarscape (in *Growing Artificial Societies* by Epstein & Axtell)

- simulates the behaviour of artificial people (agents) located on a landscape of a generalized resource (sugar)
- agents have attributes including vision, metabolism and speed
- movement governed by simple local rule:
    - look around as far as you can
    - find the spot with the most sugar
    - go there and eat the sugar
- emergent social phenomena emerge
- example: seasons cause migration and hibernation
- extended model has a second resource (spice)
    - leads to an emergent economic model



Search Problems

- problems that are solved through a search through a (typically) huge set of possible solutions
    - so optimization and modelling problems are search problems
    - but simulation problems - with an unknown desired output - are not
- search space: collection of all objects of interest, including the desired solution
- example: recall search space for travelling salesman problem is all possible tours, with size `n!`
- problem solver: a method that tells us how to move through search spaces

Optimisation versus Constraint Satisfaction

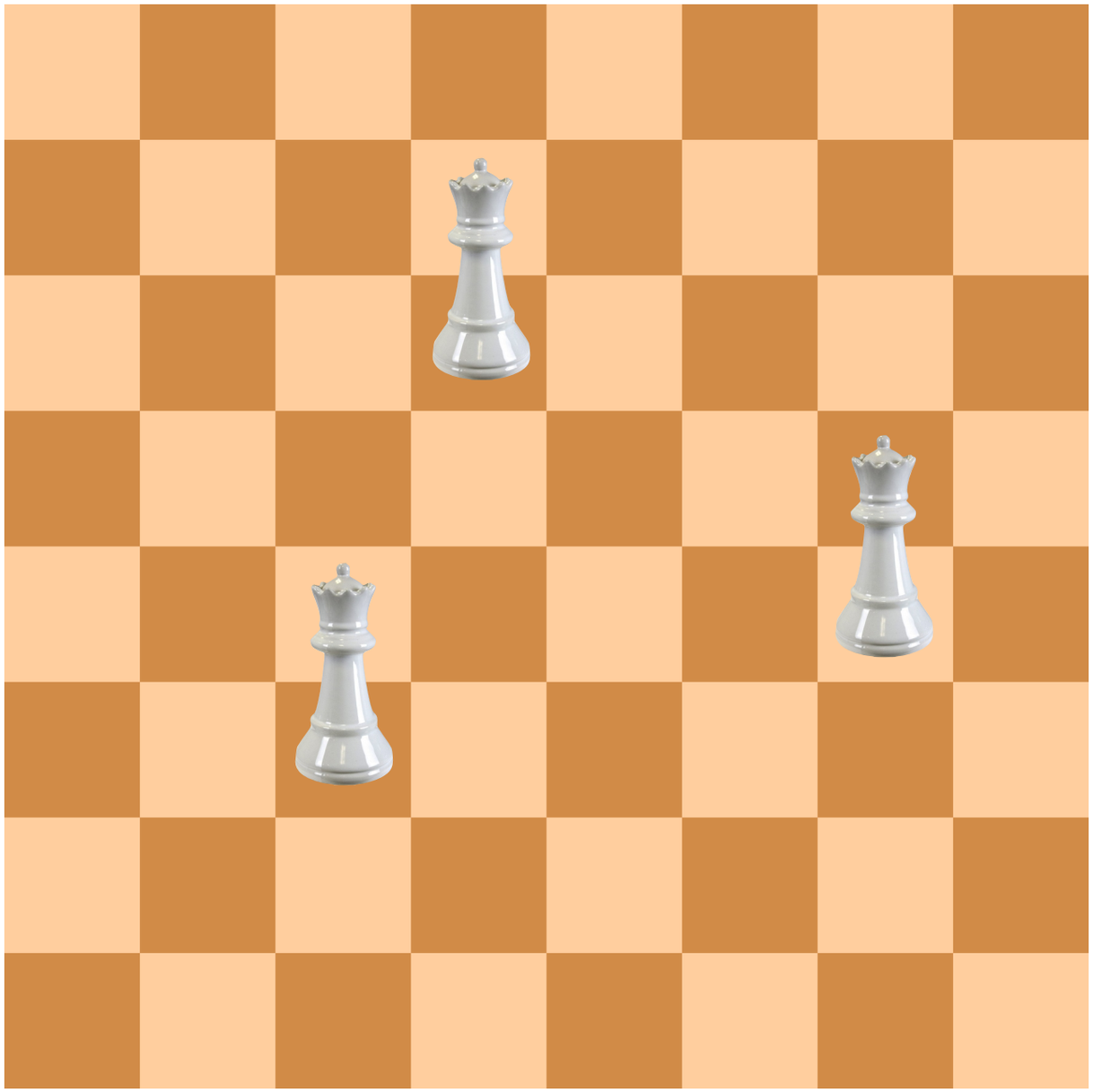
- objective function: a way of assigning a value to a possible solution that reflects its quality on a scale
    - number of un-checked queens (maximize)
    - length of a tour visiting given set of cities (minimize)
- constraint: binary evaluation telling whether a given requirement holds or not
    - find a configuration of eight queens on a chessboard such that no two queens check each other
- mixture:
    - find a tour with minimal length where city X is visited after city Y

Optimisation versus Constraint Satisfaction

	Objective Function	
Constraints	Yes	No
Yes	Constrained optimisation problem	Constraint satisfaction problem
No	Free optimisation problem	No problem

- *note:* constraint problems can be transformed into optimization problems
    - this can help if we have an algorithm (a problem solver) that can solve free optimization problems well, but cannot cope with constraints
- *question:* how can we formulate the 8-queens problem in to a FOP/CSP/COP?

8 Queens



Non-Deterministic Polytime Problems (NP Problems)

- combinatorial problem: has a search space defined by discrete variables
    - search space is finite, or countably infinite
    - so the space can be searched exhaustively by checking all possible *combinations* of variables
- can classify combinatorial problems in terms of their difficulty to solve, roughly:
    - easy: there exists a fast solver for it
    - hard: there doesn't!

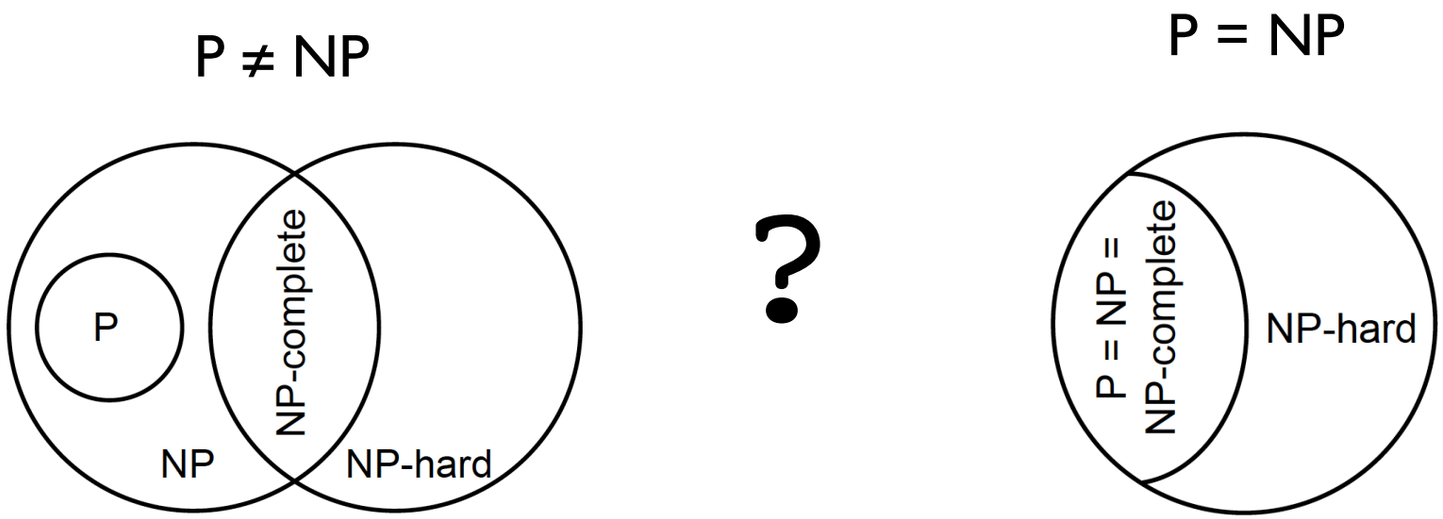
NP Problems

- problem size: the dimensionality of the problem and the number of different values for the problem variables
    - so the space can be searched exhaustively by checking all possible *combinations* of variables
- running-time: number of operations the algorithm takes to terminate
- can determine worst-case performance as a function of problem size:
    - polynomial ← easy
    - exponential ← hard

Classes

- ‘difficultness’ of a problem can be classified:
- Class P:
    - algorithm can solve the problem in polynomial time
    - (worst-case running-time for problem size n is less than F(n) for some polynomial formula F)
- Class NP:
    - problem can be solved and any solution can be verified within polynomial time by some other algorithm ($P \subset NP$)
- Class NP-complete (very well known set of problems):
    - problem belongs to class NP and any other problem in NP can be reduced to this problem by an algorithm running in polynomial time
- Class NP-hard:
    - problem is at least as hard as any other problem in NP-complete but solution cannot necessarily be verified within polynomial time

P = NP?



- known that P is different from NP-hard
- not known whether P is different from NP
    - if you figure this out, you'll be rich!
- so, assuming P ≠ NP, how do we solve NP problems?
    - by using approximation algorithms and metaheuristics
    - which is where nature-inspired computing can play a role

Reading & References

- slides largely based on Chapter 1 slides for Eiben & Smith's *Introduction to Evolutionary Computing*
  - this chapter is worth reading if you have the text
- other recommended reading:
  - see RESOURCES > Interesting Papers in Brightspace
  - maybe go search for articles about P versus NP
- cool stuff:
  - try the 8 Queens puzzle here
    - or here if you want a little help from the computer!