Lecture 01 – Introduction

Optimisation CT5141

James McDermott

NUI Galway

OÉ Gaillimh
NUI Galway

Overview

1. **What is Optimisation?**
2. Spherical rocks and special cases
3. A binary guessing game
4. Lots of applications

What is Optimisation?

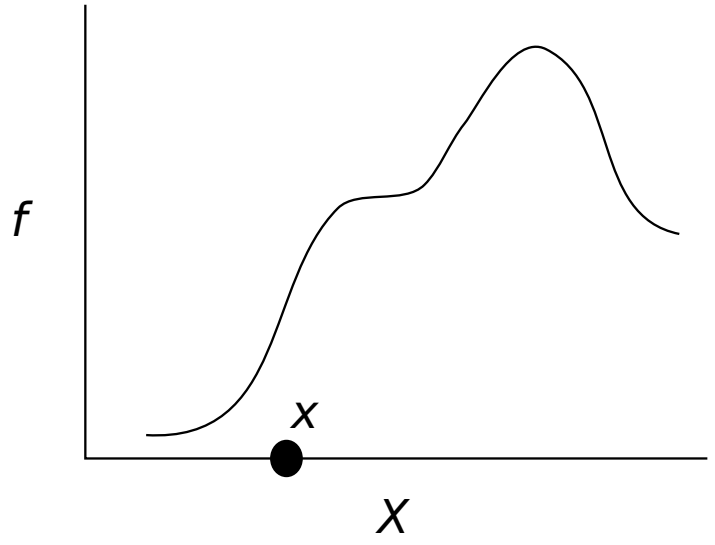
optimum: noun, from Latin, *optimus*, “best”

What is Optimisation?

"The Science of Better" http://www.scienceofbetter.org/

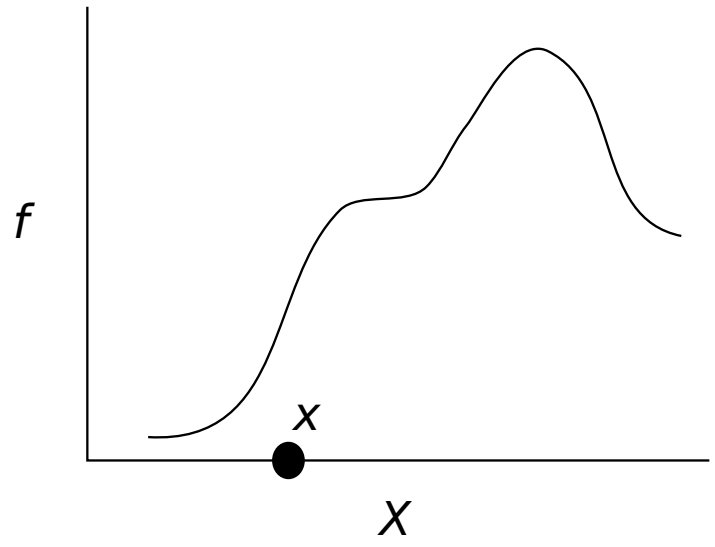
- It's about improving things, reducing costs, maximising profits, saving fuel, reducing wasted time, ...
- Optimisation methods are scientific/mathematical, but the process of formalising a problem can require creativity and intuition.

What is Optimisation?



We search for $x \in X$ which maximises the value of $f : X \to \mathbb{R}$

What is Optimisation?



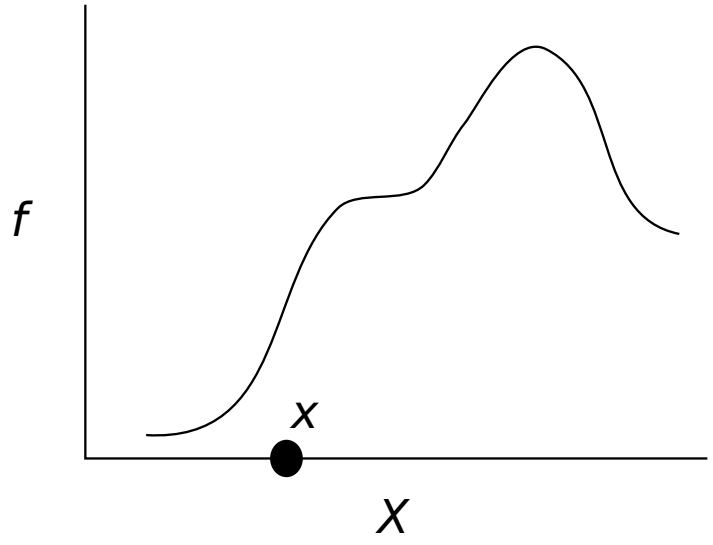
We search for $x \in X$ which maximises the value of $f : X \to \mathbb{R}$

- X is a **search space**, the set of possibilities.
- f is an **objective function**: $f(x) \in \mathbb{R}$ measures "how good" is x. E.g., "how much energy is produced by the generator design x?"

require — min

max

What is Optimisation?



We search for $x \in X$ which maximises the value of $f : X \to \mathbb{R}$

- X is a **search space**, the set of possibilities.
- f is an **objective function**: $f(x) \in \mathbb{R}$ measures "how good" is x. E.g., "how much energy is produced by the generator design x?"

In other words, we try to find: $\arg\max_{x \in X} f(x)$

What is Optimisation?

Optimisation = Search

What is not Optimisation?

(… for our purposes)

- Rewriting an algorithm to achieve $O(\log n)$ instead of $O(n^2)$
- Optimising compilers
- Search in the sense of *pathfinding*, e.g. A* or breadth-first search.

Many problems, many algorithms

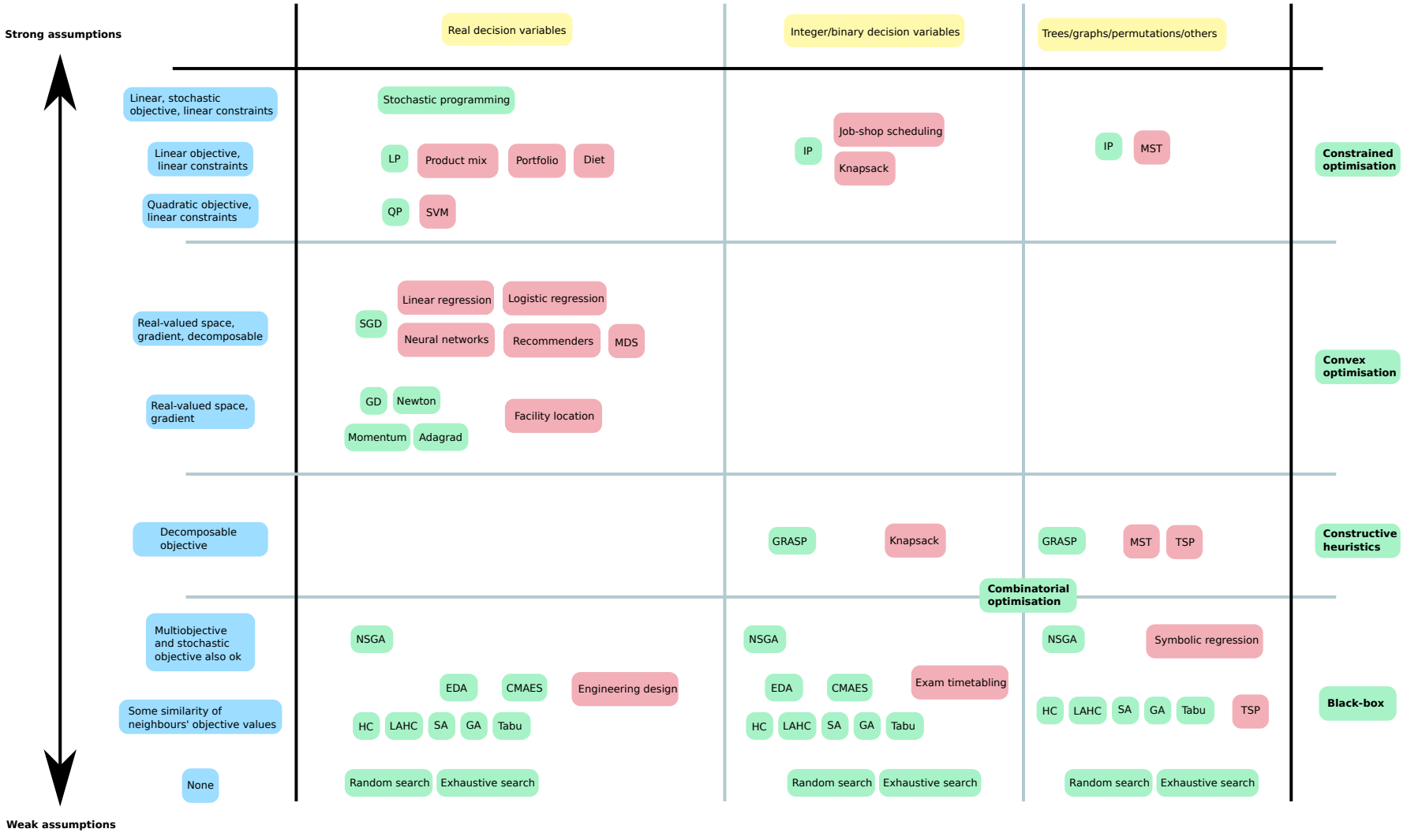
There are many optimisation problems and many algorithms. The right algorithm may depend on:

- The type of search space (e.g. real vectors, trees, permutations, bitstrings…)
- The properties of the objective (e.g. smooth, has a gradient, convex, multiobjective, …)
- Constraints (i.e. some solutions are **invalid**)

Optimisation Landscape
James McDermott
MSc Artificial Intelligence
National University of Ireland, Galway

Legend | Assumptions | Search space type | Applications | Techniques

OÉ Gaillimh
NUI Galway



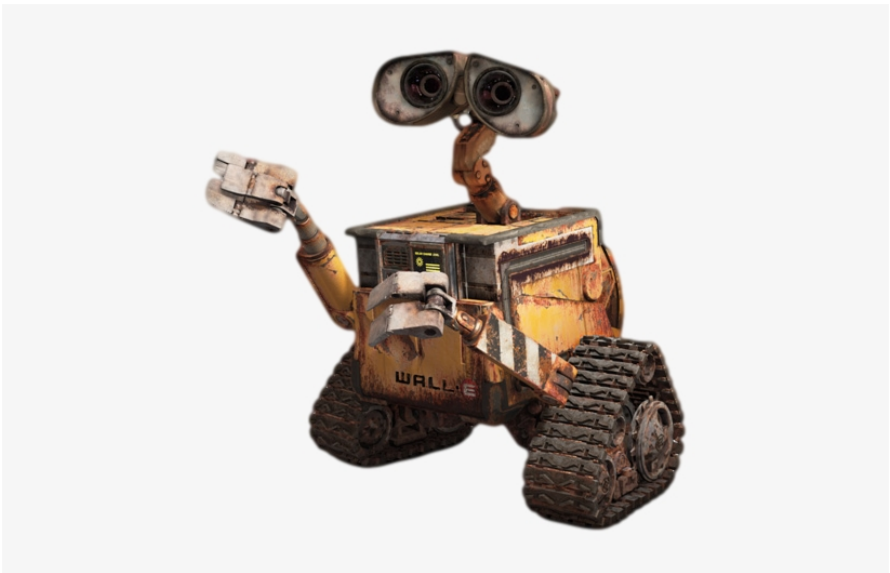
Overview

1. What is Optimisation?
2. **Spherical rocks and special cases**
3. A binary guessing game
4. Lots of applications

Example: finding spherical rocks

How come there are lots of jagged rocks, irregular rocks, oval rocks – but no spherical rocks?

Example: finding spherical rocks



Suppose I wanted to **find** a spherical rock. Suppose I have a robot which can drive around, pick up rocks and scan them with a camera.
How could I use it to find the **most spherical** rock? I want it to do all this automatically, while I have lunch.

Example: finding spherical rocks

"Scanning" means it produces a data file that describes the 3D shape in detail. If you like, it's just a cubic 3D array containing a 1 for every filled voxel and a 0 for every empty voxel.

E.g., this is a very small, flat, triangular rock:

```
[[[0,1,0],
  [0,1,1],
  [0,0,0]],
 [[0,1,0],
  [0,1,1],
  [0,0,0]],
 [[0,0,0],
  [0,0,0],
  [0,0,0]]]
```

Example: finding spherical rocks

Suppose we have a function that calculates `sphericalness(x)` for any given rock `x` in this 3D format. We can then just program the robot to try every single rock in the collection `X`, as follows:

```
best = None
bestf = -1
for x in X:
  if sphericalness(x) > bestf:
    best = x
    bestf = sphericalness(best)
```

Example: finding spherical rocks

Suppose we have a function that calculates `sphericalness(x)` for any given rock `x` in this 3D format. We can then just program the robot to try every single rock in the collection `X`, as follows:

```
best = None
bestf = -1
for x in X:
  if sphericalness(x) > bestf:
    best = x
    bestf = sphericalness(best)
```

Python gives us a shorthand for that common idiom:

```
max(X, key=sphericalness)
```

Enumerative search

```
max(X) # X a list of numbers
```

Enumerative search

```
max(X) # X a list of numbers
```

```
max(X, key=f) # pass each x to f
```

Enumerative search

```
max(X) # X a list of numbers
```

```
max(X, key=f) # pass each x to f
```

This is $\arg\max_{x \in X} f(x)$. Enumerative search is **guaranteed** to find the **optimum** – the very best point in X – because it tries every point in X.

An objective function: sphericalness

So the only hard part is to define `sphericalness`. How would you do it?

Reflection

- If X is very large, what happens?
- Is there any way to get a (fairly good) result much faster?

When would we use enumerative search?

- If we need the optimum, not just a “very good” point
- If we have no “smarter” algorithm
- If time/computational budget allows

- What about computational complexity?

When would we use enumerative search?

- If we need the optimum, not just a “very good” point
- If we have no “smarter” algorithm
- If time/computational budget allows

- What about computational complexity?

It’s a trap!

Random search

```
def rand_search(X, f, its=10000):
    return max(random.sample(X, its), key=f)
```

`its` is short for **iterations**.

Random search

```
def rand_search(X, f, its=10000):
    return max(random.sample(X, its), key=f)
```

`its` is short for **iterations**.

- Disadvantage: no guarantee of finding the optimum
- Advantage: can be as fast as we want

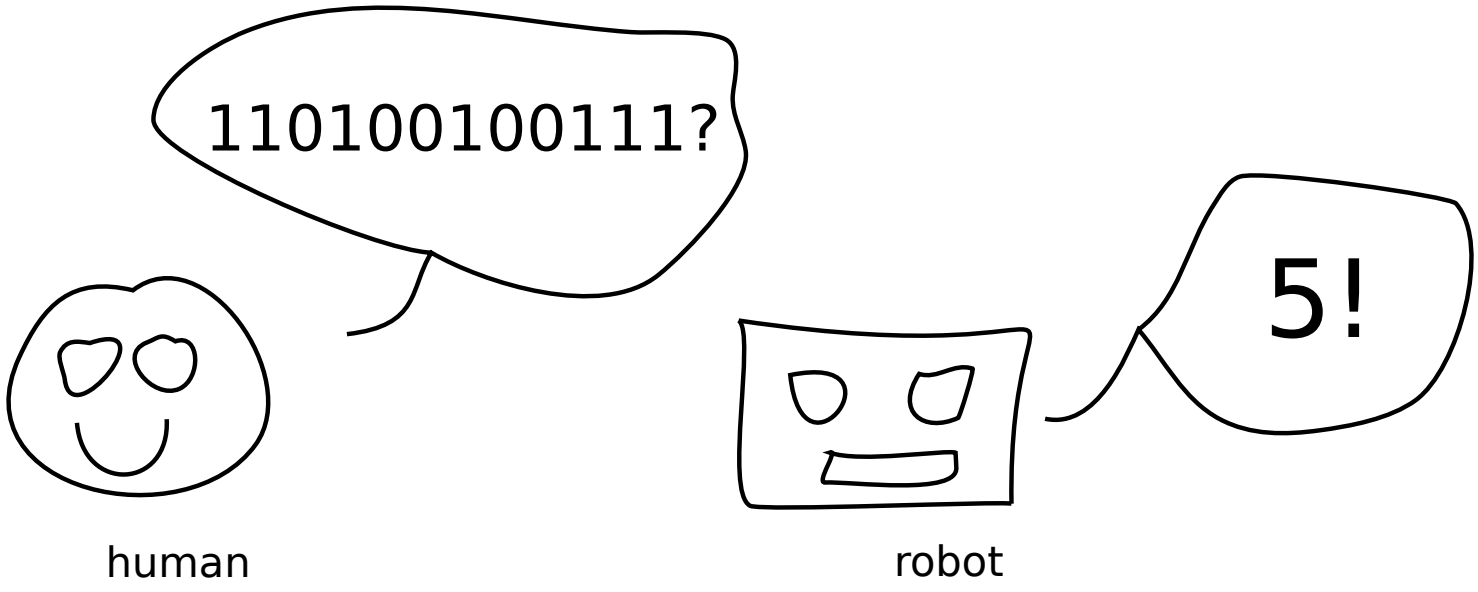
Reflection

- Random search is not very smart
- When we observe `sphericalness(x1)` for a particular rock `x1`, what does it tell us about `sphericalness(x2)` for the next rock `x2`?

Overview

1. What is Optimisation?
2. Spherical rocks and special cases
3. **A binary guessing game**
4. Lots of applications

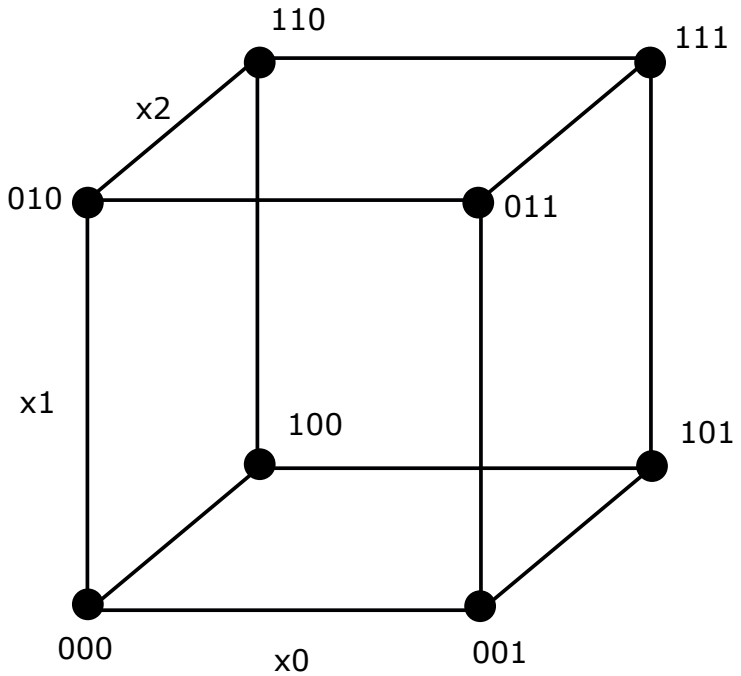
A guessing game (is an optimisation problem)



- A robot chooses a secret bitstring of length n
- We try to guess the bitstring
- Each time we guess, the robot tells us **how many bits are correct**.
- The bits are called **decision variables**.

What is the search space?

$X = \mathbb{H}^n$ (Hamming cube in n dimensions, i.e. space of bitstrings of length n)



What is the objective function?

Maximise: $f(x)$ = number of bits correct

What is the objective function?

Maximise: $f(x)$ = number of bits correct

Or equivalently, **minimise**: $f(x) = d_H(t, x)$, the **Hamming distance** between the guess x and the computer's target t.

What is the objective function?

Maximise: $f(x)$ = number of bits correct

Or equivalently, **minimise**: $f(x) = d_H(t, x)$, the **Hamming distance** between the guess x and the computer's target t.

secret	110100100111
guess	110000110111
differences	000100010000
Hamming distance	2

Hill-climbing

```
def hill_climb(init, nbr, f, its=10000):
    x = init()
    for i in range(its):
        xnew = nbr(x) # try a new point...
        if f(xnew) > f(x):
            x = xnew # improvement!
    return x
def init(): # random bitstring
    return [random.randrange(2) for i in range(n)]
def nbr(x): # make a "neighbour" of x
    x = x.copy()
    i = random.randrange(n)
    x[i] = 1 - x[i] # flip a random bit
    return x
```

Hill-climbing

- Hill-climbing may be the simplest true optimisation algorithm
- Notice that the structure is similar to our `max` code: initialise `x`, then every time we find an improvement, replace `x`.
- We'll study this properly in a few weeks.

The curse of dimensionality

The curse of dimensionality: when the number of dimensions grows, every problem (machine learning, optimisation, search, ...) gets harder, and soon becomes intractable.

Reflection

1. In the guessing game, how does the size of the space grow as bitstring length n grows?

Reflection

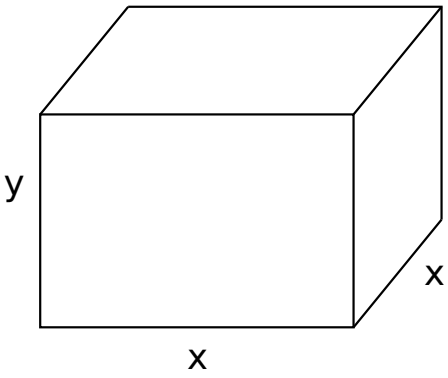
1. In the guessing game, how does the size of the space grow as bitstring length n grows?
2. For each guess x, we received $f(x)$, a single number.

   What if, instead, we received more information – e.g. exactly **which** bits are incorrect?

Overview

1. What is Optimisation?
2. Spherical rocks and special cases
3. A binary guessing game
4. **Lots of applications**

Applications: Design



A box having a square base and an open top is to contain 108 cubic feet. What should its dimensions be to minimise materials? From http://www.themathpage.com/aCalc/applied.htm.

We use "good old differentiation": we formulate M, the amount of materials, as a function of x (the side of the base): $M = x^2 + 432/x$. We can just set $dM/dx = 0$ (and also check the second derivative).

Applications: Travelling salesman problem

X is the set of possible tours of some cities. f is a measure of **total tour distance**. We might use a **constructive heuristic**.

Applications: Energy production schedules

Given data on regional electricity requirements and power plants, design a schedule to switch plants on/off to meet demand. X is all possible schedules. f_1 represents cost of fuel, and f_2 total emissions. (The "unit commitment" problem.) We might use **multiobjective optimisation**.

Applications: Timetabling

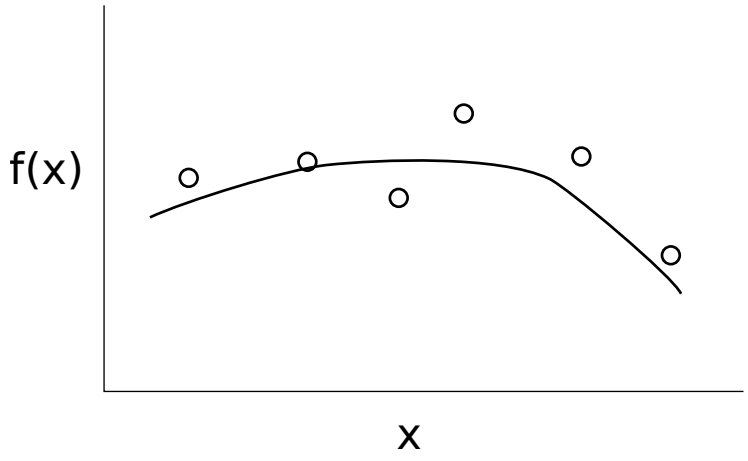
- X is the set of possible exam timetables. We have a **constraint** that any timetable with a clash is **invalid**.
- $f(x)$ might measure the number of students who have one exam right after another.
- We might use a **genetic algorithm**.

Applications: Portfolio allocation

Given a fixed amount of money to be invested and a set of n stocks to invest in, we want to maximise expected return and minimise risk.

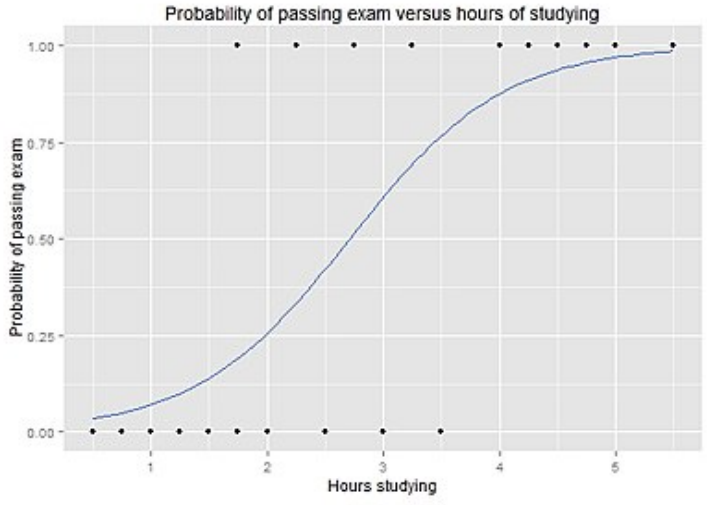
- $X \subset \mathbb{R}^n$ is the set of possible allocations. There are n decision variables.
- f is a measure of expected return and risk.
- We might use **linear programming**.

Applications: Curve-fitting



- Given a non-linear regression model $f(x)$ with fixed form but unknown parameters c
- e.g. $f(x) = c_0 x^{c_1} \cdot \log(c_2 x)$
- and given training data X and y
- we try to find the parameters c which minimise the loss $\sum_i (f(x_i) - y_i)^2$. These are the **decision variables** in our optimisation.
- We might use **non-linear least squares**.

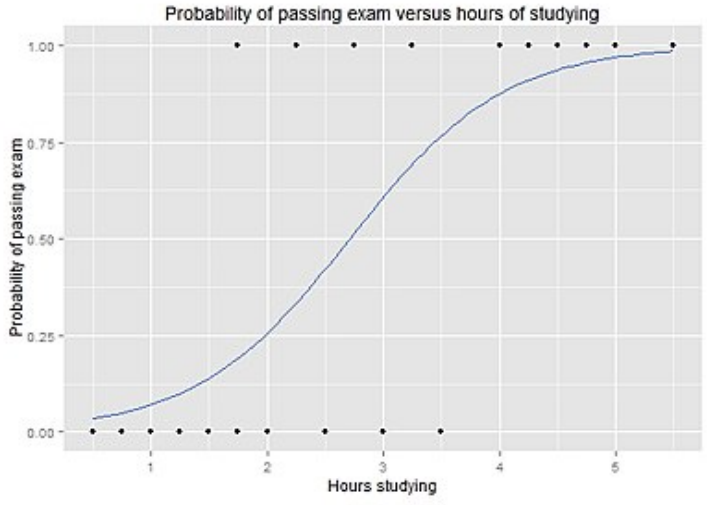
Applications: Machine Learning



Wiki

- Logistic regression: X is the set of possible **weight values** for the LR model. f might be the **loss** on training data. We might use **gradient descent**.

Applications: Machine Learning



Wiki

- Logistic regression: X is the set of possible **weight values** for the LR model. f might be the **loss** on training data. We might use **gradient descent**.
- If we have a neural network, we need **gradient descent with backpropagation**.

Applications: Program synthesis

- Given a set of input-output examples (X, y), we try to find a program $p \in X$ such that $p(x_i) = y_i$. f might measure how many examples our program p gets correct. We might use **genetic programming**.

Applications: Cache expiry algorithms

- A size-limited hardware or software cache uses some algorithm to decide which items to throw out when it becomes full, e.g. a least-recently used (LRU) algorithm.
- X is the set of possible parameters of that algorithm and f could be a measure of throughput.
- We might use simulation to calculate f. We might use **covariance matrix adaptation** to search for x.

Applications: Menus

From topsecretrecipes.com

- Suppose I'm going to host a party for 10 small kids. They'll eat sausages (EUR5/kg), chips (EUR2/kg), and ice-cream (EUR4/kg). The kids don't care what they get so long as they get 500g of food each. Suppose I don't care about their health, only about cost. How much of each type of food should I buy?