Fitness functions

Dominique Chu

School of Computing
University of Kent, UK
D.F.Chu@kent.ac.uk

Intelligent Systems, University of Kent

What is Artificial Intelligence (AI) all about?

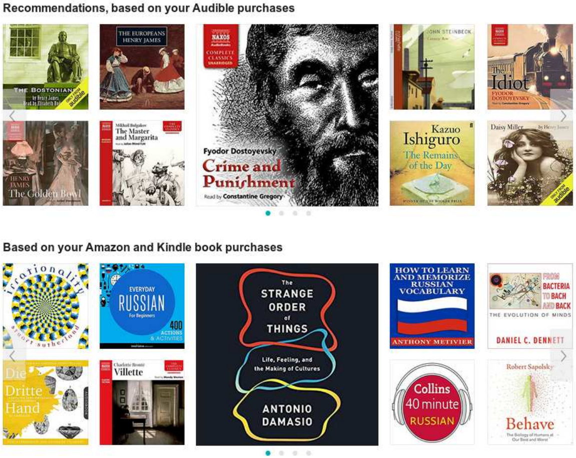
- Research into “Intelligence” is actually at most a small part of the field.
- Typical usage of AI is to find algorithms that can solve specific computational problems well.
- An important research direction at the moment is to get computers to solve problems that are traditionally hard for machines but easy for humans.
- This is not all there is to AI. Automation and optimisation are also important application areas.

Example: Handwriting recognition

A typical example is for the computer to recognise letters and numbers from pictures of handwritten text.

By Josef Steppan,
`https://commons.wikimedia.org/w/index.php?curid=64810040`

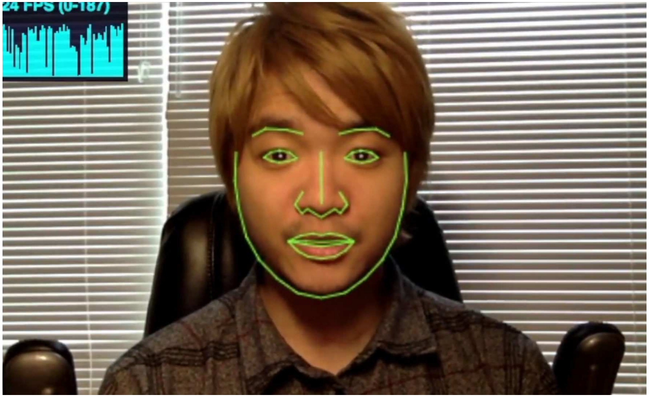
Example: Recommendation systems (such as on Amazon, Netflix)



Recommendations, based on your Audible purchases

Based on your Amazon and Kindle book purchases

Example: Face recognition



https://www.flickr.com/photos/pennstatelive/27110522957/in/photost

Anatomy of a typical AI problem

A typical AI problem consists usually of:

- A computational problem to solve, for example:
  - Which number is depicted on the image? (Classification task)
  - Who is the person on the picture? (Classification task)
  - What is the shortest path from A to B (Optimisation task)
  - Based on recent sales, how much is this house worth? (Prediction task)
- A rule to generate **candidate solutions** to these problems (**encoding**).
- There is a **fitness function** *f* that takes any possible solution and returns a value that indicates how good the solution is.

$$\text{fitness function } f : \{\text{All possible solutions}\} \to \mathbb{R}$$

- An algorithm to use existing solutions and their fitness to generate better solutions.

Note that the fitness function itself is typically unknown. If it was known, then we could just find a solution to the computational problem directly.

Functions

CO528 is all about optimising functions. So, I would like to give a short informal introduction to what a function is. Mathematically this is written like so:

$$f : \mathbb{X} \to \mathbb{Y}$$

- Here $\mathbb{X}$ and $\mathbb{Y}$ are the source and target sets respectively.
- You can think of $\mathbb{X}$ as the set of all allowed inputs, for example $\mathbb{N}$ the set of natural numbers.
- You can think of $\mathbb{Y}$ as the set of all allowed outputs, for example again $\mathbb{N}$.

Different ways to implement functions

- $f(x) = x^2$
-
```
def myFunction(x,y):
    return(x+y)
```
- Neural network
- ...

Optimisation problems

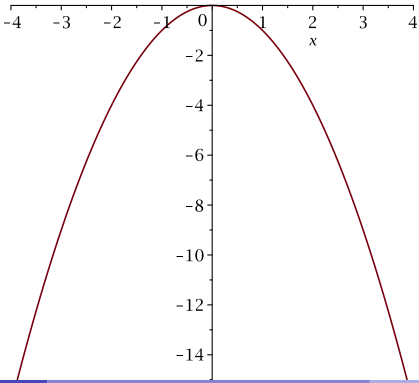
Imagine that you have a function and that you wish to find the extremum of that function.
Say, the function is:

$$f(x) = -x^2$$

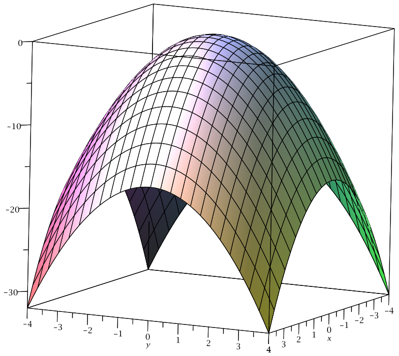
We know that this is easy to solve:
Just calculate:

$$\frac{d}{dx}f(x) = 0 = -2x$$

...and solve this for x to obtain $x^* = 0$, the value where $f(x)$ is minimal/maximal.



Let us now assume that we have a function $f(x, y) = -x^2 - y^2$. With a similar procedure we can find its extremum value.



We can now further extend to a function $f(x, y, z) = -x^2 - y^2 - z^2$ and do the same thing. I can again calculate the maximum value, but I can no longer draw the function.

Example: Black box functions

- Imagine that you have a box with many dials and a display.
- Whenever you change the dials, then the number displayed may change.
- For a given combination of dial-positions, the number displayed is always the same.

You are now given the task to find the button settings that maximise the displayed result.

Control questions

If you wanted to solve this problems using some AI algorithm, what. . .

- . . . are the candidate solutions and how do you generate them?
- . . . would a suitable fitness measure be?
- . . . could one use as encoding?

Example: Black box functions

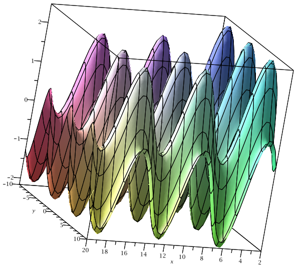
- Image that you have a box with many dials and a display.
- Whenever you change the dials, then the number displayed may change.
- For a given combination of dial-positions, the number displayed is always the same.

You are now given the task to find the button settings that maximise the displayed result.

- Try out all of the button settings.
- Use some clever method to find the solution quicker.

What could quicker methods be?

- It depends on the nature of the problem.
- It could be the total maximum is just the maximum over all individual dials
  - Maximise dial 1
  - Maximise dial 2
  - …
- It could equally be the case that this is not so.
  - It could be that the display number is apparently random (turning dials a bit or very much has, on average same result)
  - The display number could depend smoothly on the dial positions with a single maximal value (Fujiyama landscape).
  - The display number could be smooth, but with more than one optimum (local minima).



Digression: heuristic search

- You will often hear the expression “heuristic search algorithm” or simply “heuristics.”
- The idea is that you make assumptions about certain properties of your search space and design algorithms accordingly.
- Example: The fitness between two neighboring solutions is similar.
- The point is that the heuristic may or may not be true.
- When it is true, the heuristic algorithm can find good solutions very fast, but may fail to find the optimum.

Random values

There is nothing we can do in this case, except brute force:

- Enumerate all combinations of dials
- Search for the combination with the highest value.

Smooth landscape with single maximum (greedy search)

Let us, for simplicity, assume that all display values are greater than 0.

1. Start from a random dial position.
2. Record current display value as o.
3. Make a small change (into a random direction) to a randomly chosen dial.
4. Record new display value as d.
5. If $d < o$ then undo the change to the dials.
6. Goto 2

If we run through a procedure like the above for a sufficient number of timesteps, then we will eventually find the unique global maximum.

Control question:

If local minima exist then greedy search does not work any more. Why?

Local minima exist

- This is a difficult situation.
- Some mechanism is required to avoid getting stuck on local minima.
- There is typically a trade-off between
  - . . . **convergence** (the ability to find good solutions)
  - . . . **exploration** (the ability to avoid local minima).
- Parameters of heuristic search algorithms have built-in parameters that control this trade-off.
- Intuitively, the idea is to design algorithms that both explore the fitness landscape locally **and** occasionally make big jumps.
- The limiting case of only exploration is random search (usually not very efficient).
- The limiting case of local exploration is greedy search.

Example: Shortest Path in Travelling Salesman Problem

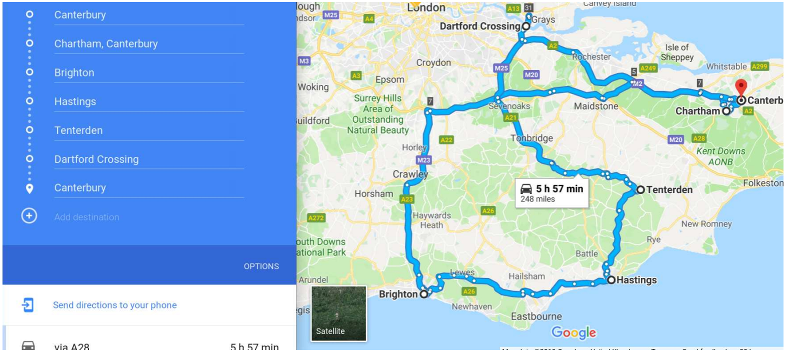
- Imagine you need visit Tenderden, Canterbury, Hastings, Chartham, Brighton and Dartford. Say, you start in Canterbury.
- You (clearly) want to minimise your driving (mileage) on the day.

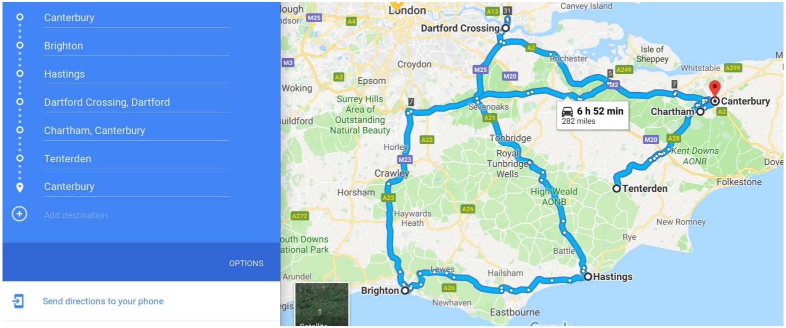
Question

- What is a candidate solution in this case?
- What is the fitness of a candidate solution?
- What is the fitness function in this case?

Travelling salesman problem continued

- The fitness in this case is the **total distance travelled** when visiting all cities.
- A candidate solution in this case is a sequence of cities where each city apears exactly once.
- The fitness function is the mapping from a sequence of cities to the distance.
- Example itineries are:
  - Canterbury,Chartham,Brighton, Hastings, Tenderden and Dartford (248 miles)
  - Canterbury, Brighton, Hastings, Dartford, Chartham, Tenderden (282 miles)
  - ...


Canterbury
Chartham, Canterbury
Brighton
Hastings
Tenterden
Dartford Crossing
Canterbury



Canterbury
Brighton
Hastings
Dartford Crossing, Dartford
Chartham, Canterbury
Tenterden
Canterbury

Solving the TSP is hard

- The travelling salesman problem is computationally intractable.
- The time required to find the best route grows very (!) fast with the number of cities.
- Already for a moderately large number of cities it is impossible to check all combinations, because it would take too long.
- At the same time, there are no short cuts or tricks to find the optimal solution quickly.

The approach

- The TSP and many similar optimisation problem can be solved by algorithms inspired by natural evolution.
- There is a large class of these algorithms, called **Evolutionary algorithms (EA)**.
- We will now look at a particular type of EA, namely **Genetic algorithms** (GA).
- These algorithms can find very good solutions to difficult optimisation problems quickly.
- They cannot be relied on to find the single best solution.
- They are best applied to problems where:
  - Similar candidate solutions have similar fitness.
  - There are local minima.
  - There are no short-cut solutions that can find the optimal solution directly.
  - Brute force is not feasible.

Background: Theory of Natural Evolution

- Genetic code is a string of 4 letters (A,C, G, T).
- When a new organisms is created the genome is copied.
- This copying process is mostly reliable, but occasionally errors occur, i.e. **mutations** occur.
- Most of the time, these errors lead to solutions (organisms) that are not viable.
- Sometimes they lead to solutions that are actually “better.”
- Looking around the room here, it is clear that evolution has come up with some quite impressive results.

The **fitness function** is a function that assigns a **fitness value** to each possible combination of genes.
Link: If you want to play with evolution: Biomorphs

Can we harness the power of evolution to optimse fitness functions?

- Evolutionary algorithms are a class of optimisation algorithms that are inspired by natural evolution.
- They generally consist of the following key components:
    - An encoding of a solution.
    - A fitness function
    - Selection operators.
- They usually start with a **population** of random candidate solutions.
- Each candidate solution is assigned a fitness value.
- A new population is created from the old one by mutation/selection. Individuals with a lower fitness are less likely to proceed.
- Typically, there is a tension between fast convergence (optimum is found) and exploration (local minima are avoided).

Genetic algorithm: Basic outline

1. Create an initial population of candidate solutions.
2. Determine the fitness of all candidate solutions.
3. Create a new population from the existing population by applying a selection procedure.
4. Goto 2

Encoding candidate solutions

- The first question that needs to be addressed when solving a problem using a GA is the encoding.
- We need a way to write down possible solutions to the problem in a representation that can be manipulated by the GA.
- Binary vectors are a popular choice, but not always the best.
- For some problems it is useful to use different choices, such as vectors of floats or integers.

Question

What would be a good encoding for candidate solutions of the TSP problem?

Choosing a population size

- Genetic algorithms usually consider a large number of candidate solutions at the same time.
- Before starting your GA, you need to decide on a population size, i.e. how many solutions you use to search the fitness landscape in parallel.
- A larger population is usually advantageous in that it increases the chance to find good solutions.
- Larger populations also increase the resource usage, in particular run-time. This can become a limiting factor.
- As the population size becomes larger, further increases have diminishing returns.

Selection

- The overall aim of selection is to modify the current population so as to get better solutions (on the whole).
- There are two aspects to selection
  1. Select a candidate to go forward from the previous population to the next one.
  2. With a given probability the chosen candidate is modified using **mutation** or **selection** or both.
- Selection can be done **with or without replacement**. If with replacement then a specific candidate solution can be chosen more than once to proceed to the new population. If without replacement, chosen solutions are discarded from the current population.
- Whatever selection mechanism you use, it is important to ensure that the population size does not change.

Selection schemes

The aim of the selection scheme is to produce a new population from the current one. The new population should be similar to the current one, but not completely the same. Ideally it has a better fitness.
There are a number of possible selection schemes.

- **Fitness proportional selection**: We select a solution from the current population to go to the next one with a probability that is proportional to its fitness. This is also called roulette wheel selection.
- **Tournament selection**:
  1. Randomly pick two or more solutions from the current population.
  2. Determine the solution that is best amongst the chosen ones.
  3. Add the best one to the new population.
- **Elitism**: Always choose the *n* best solutions for the new population. This is not a selection scheme in its own right, but can be combined with the previous ones.

Example: Fitness proportional selection

Assume you have a population of size 5. The solutions in your current population have the fitness:

Index	0	1	2	3	4
Fitness	120	9	10	90	0

We now want to calculate the probability to choose the first candidate solution (with index 0):

1. Calculate the sum of all fitnesses: $P = 120 + 9 + 10 + 90 + 0 = 229$
2. Divide the fitness of the first solution by P to obtain the probability to choose the first candidate solution p_0.

$$p_0 = \frac{120}{229} \approx 0.524$$

Question

Without calculating, what is the probability to choose the fifth candidate solution?

Example: Fitness proportional selection

Now that you have calculated all the selection probability, choose a particular candidate solution to go forward:

1. Calculate the following table:

$P_0 = p_0$	0.524
$P_1 = p_0 + p_1$	0.5633
$P_2 = p_0 + p_1 + p_2$	0.607
$P_3 = p_0 + p_1 + p_2 + p_3$	1.0
$P_4 = p_0 + p_1 + p_2 + p_3 + p_4$	1.0

2. Now draw a random number r between 0 and 1.
3. Find the index j such that $P_{j-1} < r \leq P_j$ (assuming that $P_{-1} = 0$).
4. Choose candidate solution j to go to the next population.

Question:

Assume that $r = 0.601$. Which candidate solution will be chosen?

Example: Tournament selection

Assume you have a population of size 5, and your tournament size is 3. The solutions in your current population have the fitness:

Index	0	1	2	3	4
Fitness	120	9	10	90	0

1. Choose at random 3 (different) candidate solutions. (We choose here indices 1,2,4).
2. Find the one that has the highest fitness amongst the tournament. In our example this is index 2.
3. Move the chosen index to the new population and start again untill the new population has reached the required size.

Questions

- Fitness proportional selection may result in very bad solutions going forward to the new population. Why bother and not just go with elitism all the time, which would guarantee that only the best solutions go forward?
- How to use fitness proportional selection when *lower* fitness is better, as is the case with TSP, for example?
- In tournament selection, what is the effect of changing the size of the tournament to the size of the population?

Mutation

- Once a solution is chosen for the new population, we could change it.
- *Mutation* is a **small** change of an existing solution.
- When solutions are binary strings than this is often chosen to be the flip of a single bit.
- If solutions are vectors of floats then one could implement mutations as a small variation of one of the numbers.
    - In this case, you have to be careful to respect any limits on the number (for example, the encoding could be such that it has to be positive).
    - Also, be careful to ensure that the change is actually small. You could decide to add/substract 1% at most.
- Mutations are done with a probability usually called the **mutation rate**, such that only some of the solutions are subject to mutation.
- The mutation rate will impact on the performance of the GA.
- Mutation is the major mechanism that leads to the exploration of the fitness-landscape in that it leads to new (possibly good) solutions.

Crossover

- Crossover takes two different solutions and produces a new solution from the old one.
- Assume possible solutions are: [0,1,1,1,0,0,1,0] and [0,0,0,1,0,1,1,1].
- We can now create a new solution from these two by taking the head of the first and the tail of the second one to obtain: [0,1,1,1,0,1,1,1].
- Another possibility is to create a new solution by taking the head of the second and the tail of the first solution to obtain: [0,0,0,1,0,0,1,0].
- In this example I chose the *crossover point* to be 4 (in the middle). Instead, you could choose it to be anything, including a random number each time.
- One can also choose more than one crossover points.
- When applying crossover, it is important to think of a scheme to find the second parent. You could, for example, choose it at random.

Discussion question

Discussion question: What is the purpose of crossover?

Bringing it all together

1. Create an initial population of N candidate solutions.
2. Determine the fitness of all candidate solutions.
3. Create a new population from the existing population by applying the selection procedure until the new population contains N candidate solutions:
    1. Select a candidate solution using tournament selection, roulette wheel selection or similar schemes.
    2. Determine whether or not to subject the chosen solution to mutation.
    3. Determine whether or not to subject the chosen solution to crossover.
    4. Add the selected and modified solution to the new population.
4. Make the new population the current one.
5. Determine the fitness of each candidate solution of the population.
6. If you have found a satisfactory solution stop, otherwise Goto 3.

Vocabulary (review)

- encoding of a solution
- fitness function
- mutation/selection/crossover
- selection operator
- candidate solution
- global/local optimum
- convergence

Example problem to solve:

- Imagine that you have *N* items that you would like to take with you on vacation.
- Each item has a *weight* and a *utility value* for you.
- You are allowed to bring at most 15kgs.
- How do you pack your suitcase so as to optimise the total utility of your choice (sum of utility of individual items).

Item	Weight	Utility
0	3	9
1	4	12
2	1	17
3	1	1
4	1	1
5	9	19
6	6	8
7	1	5
8	4	6
9	5	1
10	8	4

- What is a good encoding?
- How to mutate a candidate solution?
- How does crossover work?