



NUI Galway  
OÉ Gaillimh

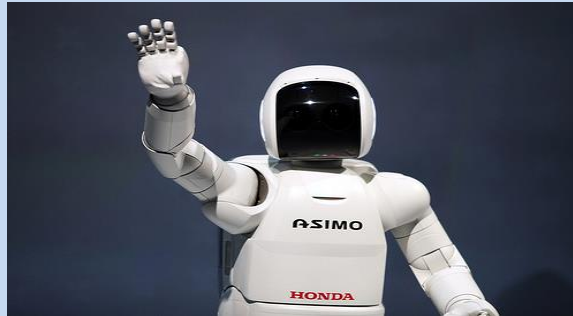


Photo by [Ars Electronica](#) under [CC-BY-NC-ND](#).

## AI Search – Informed Search

### Workshop 2 Section 2

### CT621 Artificial Intelligence - MScSED

Welcome to the second section of the week 2 workshop. In this section we examine the concept of informed search and analyse the operation of *Hill Climbing* and *Evolutionary algorithms*.

## Learning Outcomes

On completing this section and related learning activities, you should be able to:

Explain

- The characteristics of *informed search*

Demonstrate

- How a *heuristic* can be used to reduce the search space

Devise

- Heuristics for problems

Explain

- The operation of a *Hill Climbing Search Algorithm*

Describe

- The operation of *Genetic Algorithms*

## Informed Search

*An informed search strategy uses problem-specific information to search the most promising areas of the state-space first*

*An informed search strategy can:*

- Generally find a solution more quickly than an uninformed search strategy*
- Be used to find solutions when there is limited time available*

Image Licence CC0

In Workshop 2 Section 1 we looked specifically at uninformed search strategies such as depth-first and breadth-first search. Such blind searching strategies represent an exhaustive exploration of the search space guided by a rigid search pattern. That is, they systematically try every possible path until they find the goal that they are searching for. The only real difference between them is the order in which they examine the states.

In contrast an informed search strategy uses knowledge of the problem itself to guide the search of the state space. Unlike depth & breadth first search, states are not examined in a pre-defined way.

Therefore, such a strategy may guide its search based on some measure of the quality of the states. This raises the question: how can we assess the quality of different states?

We can use heuristics, which are discussed on the next slide.

## Heuristics

*A heuristic is a measure of the quality or fitness of a specific state*

*The heuristic value of a state  $s$  is the estimated cost of reaching the goal state from  $s$*

The measure we will use to assess the quality of a state is called a heuristic (sometimes called a rule of thumb, as the quality measure is an approximate estimate). Heuristics are used to evaluate each state, so that the one that appears to be the best option can be chosen. For example, in game playing, the move that seems to most likely lead to a winning position is chosen.

Informed search is the use of heuristics to guide the search. Rather than systematically searching through the search space, the states to visit are sorted on their heuristic value, and the state with the 'best' heuristic value is chosen.

## 8-Tile Puzzle

- ▶ To illustrate the role of heuristics we will focus on the selection of appropriate heuristics for the eight-tile puzzle
- ▶ The image shows the goal state.

1	2	3
8		4
7	6	5

See Workshop 1 Section 1 for a description of the 8-tile puzzle game.

## Selecting a Heuristic for 8–Tile Puzzle

- ▶ An ideal heuristic would be one that returned the exact number of moves to solve the puzzle
  - ▶ However to do this we'd need to have already done the search - defeating the purpose of doing a search
- ▶ What is required is a heuristic that **approximates** the number of moves required

In the eight-tile puzzle the aim is to solve the puzzle in as few moves as possible, so it would be extremely useful if we could devise a heuristic that would always tell us how many moves are required to solve the puzzle. Therefore, the lower the heuristic value of a given state the more preferable that state would be. The problem with this, is that in order calculate the exact number of moves, we'd need to solve the puzzle, which is exactly what we are trying to get the heuristic search to do.

The approach taken is to define heuristics that attempt to estimate the number of moves required. In this way, heuristics can be seen as **approximations** of the quality of a state.

It is important to note that an admissible (i.e. workable) heuristic should be a lower bound on the number of moves to the goal state. In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state.

## Example Heuristics for 8-Tile Puzzle

- ▶ What are possible heuristics?
- ▶ Number of tiles out of place (Good)
- ▶ Sum of distance out of place (Better, compensates for long moves)
- ▶  $2 * \text{number of direct reversals}$  (Since direct reversals are tricky)

So how can we approximate the number of moves to solve the puzzle?

Perhaps the simplest method would be to count the number of tiles out of place, since the more tiles out of place the more moves would be required. As only one tile is moved at a time, and each tile out of place must be moved at least once, this heuristic will always give a value less than or equal to the actual number of moves required.

Using the number of moves that each tile would need to get back to its correct place is more accurate, because if a tile is  $n$  moves away from its correct place, it will take at least  $n$  moves to get it to its place. Again this heuristic will always be less than or equal to the actual number of moves, but it will always be greater than or equal to the previous heuristic.

A specific problem with the eight-tile puzzle is that it is particularly difficult to swap **two** tiles, so it may be useful to record the number of pairs of tiles that are in each other's places, as these will require many moves to resolve.

## Calculating Heuristic Values for 8–Tile Puzzle

- ▶ Consider the situation where we are currently in the state **s** shown in the top image
- ▶ Tiles out of place (TOOP) = 4
- ▶ Distance out of place (DOOP) = 5
- ▶ Direct reversals (DR) = 0

2	8	3
1	6	4
7		5

State s

1	2	3
8		4
7	6	5

Goal State

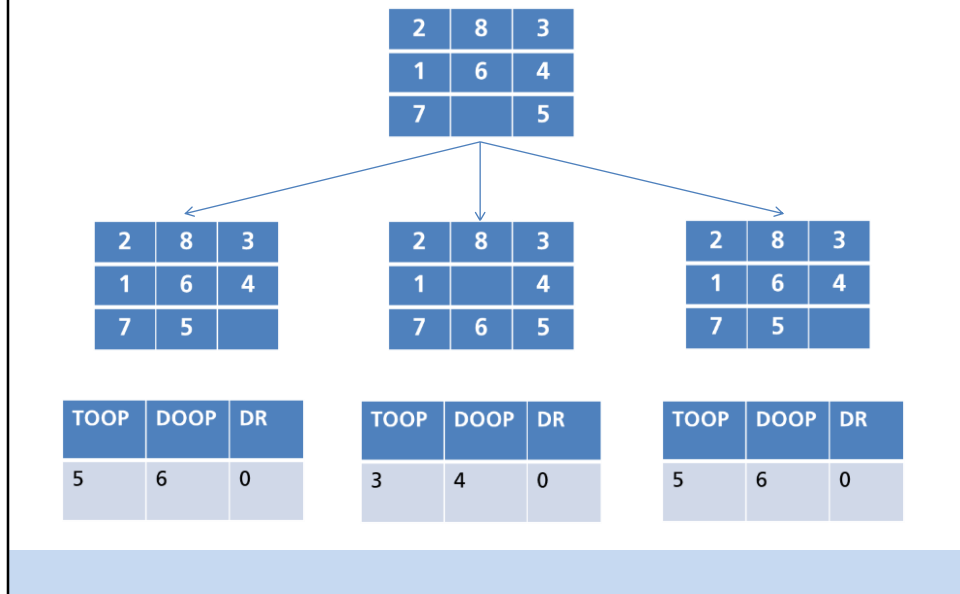
This slide shows an initial state and the values of the three heuristics that we've seen.

From this initial state, there are three possible moves, the 6 could move down, the 7 could move right and the 5 could move left.

Distance out of place is 5 => 2 is 1 out of place; 1 is 1 out of place; 6 is 1 out of place; and 8 is 2 out of place.



## Using Heuristics to Guide Search in the 8-Tile Puzzle



There are three possible actions available from the initial state. We can move the blank space up, left or right. The resulting states are depicted in the slide. For each resulting state this slide shows the values of the three heuristics that we've discussed in the previous two slides.

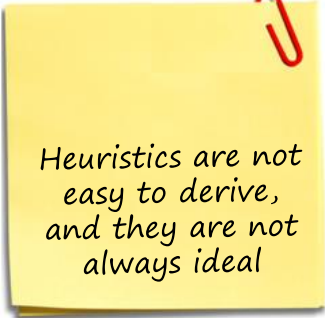
Both the TOOP and DOOP suggest that the middle move is the best one to make, as they both suggest that this move will require the fewest additional moves to solve the puzzle. The DR heuristic is not at all helpful, as there are no direct reversals.

In fact the correct move to make is to slide the 6 down, as it happens, from this state the puzzle can be solved in exactly 4 moves, as the Distance Out Of Place heuristic suggests.

If you carried this analysis on, you would find that the Distance Out Of Place heuristic would lead directly to the solution. The Tiles Out Of Place would perform slightly worse, with possibly one wrong move, and the Direct Reversals would of limited use.

## Summary of Example Heuristics

- ▶ Of the 3 heuristics:
  - ▶ Each pruned the search space
  - ▶ Sum of distances was the best
  - ▶ 2 \* direct reversals was barely better than breadth first search



*Heuristics are not  
easy to derive,  
and they are not  
always ideal*

[Image Licence CC0](#)

Despite our apparent success with the Distance Out Of Place heuristic, it won't always be so successful. The number of direct reversals WILL have an impact on the number of moves required. One possible improvement would be to extend the tiles out of place heuristic by adding on a term that indicates the number of direct reversals. This should help when there are direct reversals, and should avoid new direct reversals being created by moves.

The key thing to take from this example, is that even for relatively simple problems, accurate heuristics may be difficult to derive. They also may only work for some states, and perform badly for others.

## Examples of Informed Search Algorithms

- ▶ The remaining slides will introduce two examples of informed search algorithms
  - ▶ Hill Climbing Algorithm
  - ▶ Genetic Algorithm

## Overview of Hill Climbing Algorithm

- ▶ Hill Climbing (HC) is an iterative search algorithm that uses a heuristic to move to states that are better than the current state
- ▶ Terminates when no neighbour has a better value
- ▶ HC does not look ahead beyond its immediate neighbouring states



[Image Licence CC0](#)

Hill climbing (HC) is a simple search algorithm that iteratively searches its immediate neighbouring states. It moves to the neighbouring state with the best heuristic value as long as it's better than the heuristic value of the current state. The HC algorithm terminates when it reaches a peak where no neighbour has a better heuristic value. It does not maintain a history of prior states and will only ever consider its immediate neighbours. It has been described as “try to find the top of Mount Everest in a thick fog while suffering from amnesia” (Russell and Norvig, 2010)

### Reference:

Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson Education

## Hill Climbing Algorithm

```
function Hill-Climbing returns a state that is a local maximum
inputs problem: a problem
local variables current: a node
                  neighbour: a node

current <- MAKE_NODE(problem, INITIAL_STATE[problem])
loop do
    neighbour <- a highest-value successor of current
    if HEURISTIC[neighbour] <= HEURISTIC[current]
        then return STATE[current]
    current <- neighbour
```

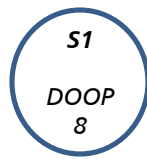
The HC algorithm can be described using the following five steps:

1. Select a random point in the state space.
2. Consider all the neighbour states of the current state.
3. Choose the neighbouring state with the best heuristic value and move to that state.
4. Repeat steps 2, 3 and 4 until all the neighbouring states are of lower quality.
5. Return the current state as the solution.

## Hill Climbing Example

### ► 8-Tile Puzzle Example

- Each state  $s$  represents a particular configuration of tiles
- HC algorithm uses Distance out of Place heuristic



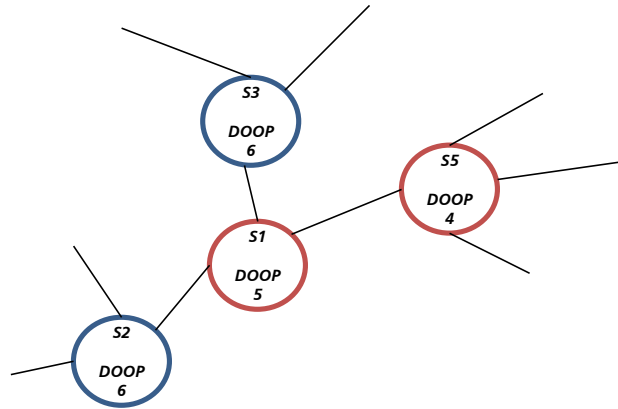
Example State

State is labelled  $S1$  and the heuristic (DOOP) has a value of 8.

These slides look at the application of a HC algorithm to the 8-tile puzzle. Each state is labelled and represents a particular configuration of tiles. The HC algorithm uses the Distance out of place heuristic to assess the quality of each state. Remember the Distance out of place heuristic measures the **sum** of distances that each tile is out of place.

Therefore, the lower the heuristic value for a state then the better the quality of the state. The goal state would have a DOOP value of 0. The objective of the algorithm is to search the sample state space to find the best solution.

## Hill Climbing Applied to 8-Tile Puzzle

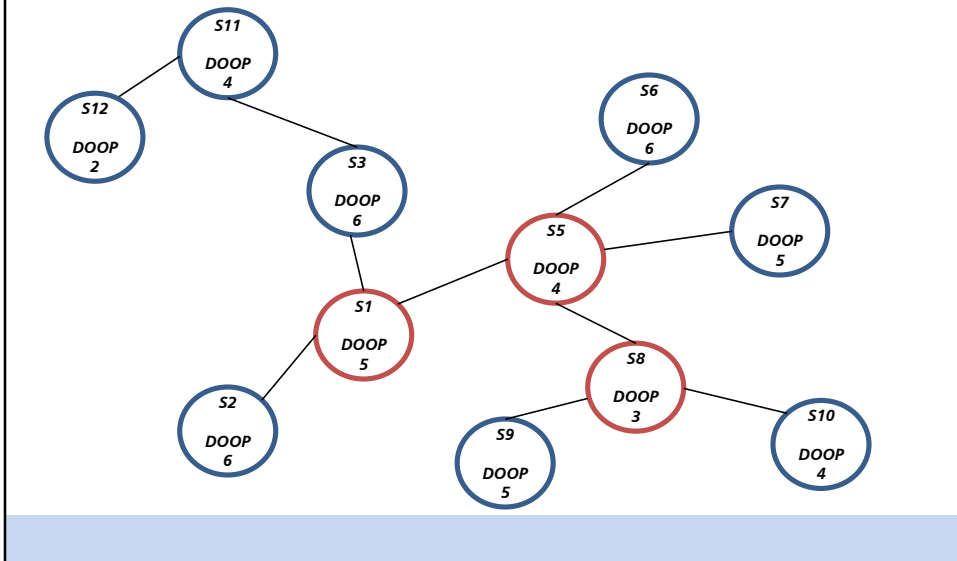


Please note that in the diagram actions are represented by links between states. Those actions visited by HC algorithm have a red outline.

The HC algorithm begins its search in the state S1. Remember, in the 8-tile game you can take 2, 3 or 4 actions from a state depending on the state. From S1 the available actions will take us to S2, S3 or S5.

The S5 has the best heuristic value (remember the objective is to minimise the heuristic value, the goal state has a value of 0). So the HC algorithm then moves to the state S5.

## Hill Climbing Applied to 8-Tile Puzzle



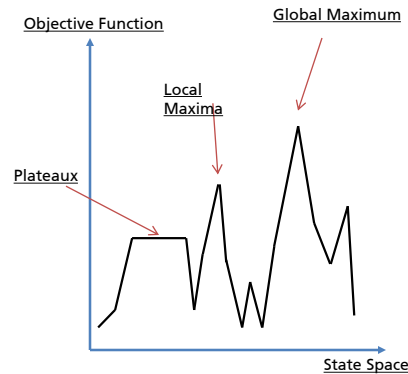
The algorithm has moved to that state S5. Note that slide now depicts a larger view of the state space. When in S5 the algorithm now has four possible actions available to it. It examines the heuristic values of each of its immediate neighbours (S8, S7, S6, S1). It takes the action that leads it to the state S8, which has the best heuristic value.

It again examines each of its available neighbours (S9, S10 and S5). Each of these states has a higher heuristic value than the current state. Therefore, the HC algorithm terminates and return S8 as the solution.



## Limitations of Hill Climbing

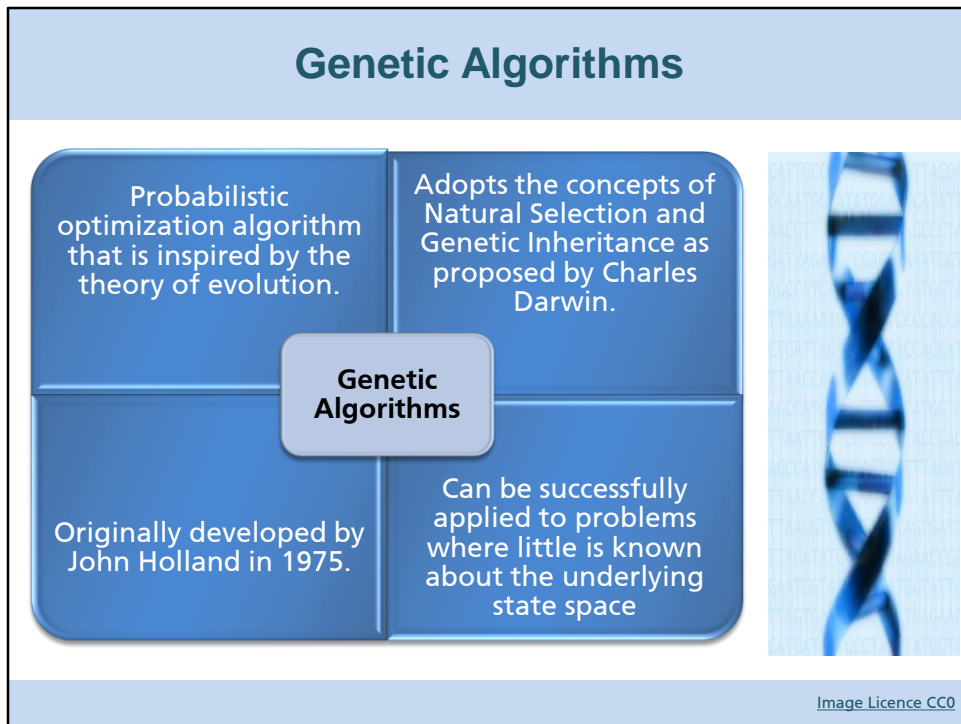
- ▶ Local Maximum – HC algorithm can become stranded on a sub-optimal peak
- ▶ Plateaux – A flat area of the state space landscape where HC algorithm becomes stuck



The main problem with the hill climbing approach to search is that it cannot guarantee that it will find the best solution. In fact, it does not offer any guarantees about the solution. Therefore, potentially it can provide a very poor solution. When the algorithm reaches a position where there are no better neighbours it terminates. However, it is not a guarantee that we have found the best solution because this solution could be a local maximum as illustrated in the graph.

Notice that in the 8-tile example on the previous slide the HC algorithm returns a solution with a heuristic value of 3. We know the game is solvable with the goal state having a value of 0. Therefore, the state returned by the HC algorithm is a local maxima. There is even a state S12, which is depicted on the previous slide, that has a better heuristic value of 2 than the solution found by the HC algorithm.

Another difficulty that a hill climbing algorithm could encounter is that of a plateaux. This is an area of the state space that is flat. It may be a flat local maximum from which no uphill exists. This results in the algorithm becoming stuck on the plateaux, as shown.



Genetic algorithms (GAs) are an AI informed search technique that applies the principles of evolution as proposed by Charles Darwin. More specifically the basic techniques of GAs attempt to simulate some of the processes observed in natural evolution such as natural selection and genetic inheritance.

GAs were first developed by John Holland in the University of Michigan (Holland, 1975). They are widely used within business, scientific and engineering circles. They have been successfully applied to problems such as route planning, robotics, automotive design and financial and investment modelling.

**Reference:**

Holland, J. J., (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor Michigan.

## Overview of a Genetic Algorithm

```
// Generate an initial population of individuals
population <- GENERATE_POPULATION()

//Evaluate the fitness of all individuals using a
heuristic
CALCULATE_FITNESS(population, HEURISTIC)

While termination condition is not satisfied do
    EVOLVE_POPULATION(population, HEURISTIC)
End while
```

[Image Licence CC0](#)

A genetic algorithm will first generate a large group of possible states. In GA terminology this group is referred to as a *population* and each state within the population referred to as an *individual*.

GAs are informed search techniques and as such use heuristics to evaluate the fitness or quality of individuals within the population. After the initial generation of a population a fitness value is assigned to each individual.

The algorithm then enters a loop. Each iteration of the loop is referred to as a generation. Each time around the loop the algorithm evolves the population of individuals. Therefore, the activities within the loop are concerned with producing a potentially fitter population of individuals. The process of evolving the population is covered in more detail on the next slide.

The loop generally keeps iterating until some individual within the population is deemed fit enough or if a certain amount of time has elapsed.

## Structure of a Genetic Algorithm

```
Function: EVOLVE_POPULATION
population is a set of individuals
HEURISTIC a function that measures an individuals fitness

new_population <- empty_set
for i=1 to SIZE(population) do

    x <- RANDOM_SELECTION(population, HEURISTIC)
    y <- RANDOM_SELECTION(population, HEURISTIC)
    child <- REPRODUCE(x,y)

    if (small random probability)
    then child <- MUTATION(child)

    add child to new_population

population <- new_population
CALCULATE_FITNESS(population, HEURISTIC)
```

[Image Licence CCO](#)

The process of evolving a population is performed within a **for** loop. The old population is gradually evolved to a population of *new individuals*. During each iteration of the **for** loop a single evolved individual is produced and added to the *new population*. Once a *new population* is produced, a fitness level is assigned to all individuals within the population.

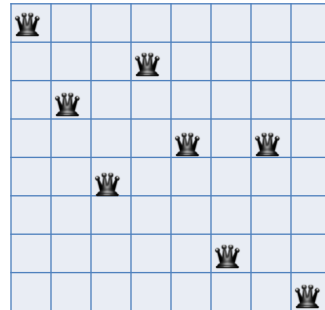
The steps involved in this evolution process are as follows:

1. **RANDOM\_SELECTION**. Two individuals from the current population are selected to proceed to step 2. The fitter the individual the higher the probability that they will be selected. This reflects Darwin's idea of Natural Selection. The more suited the individual is to their environment the more likely they are to survive and reproduce.
2. **REPRODUCE**. Those individuals selected from step 1 are recombined to form a single new individual (it is also common in GAs to produce two new individuals at this stage). Reproduction combines elements of two parents to produce a new offspring. A subset of the elements of one individual are combined with a subset of the elements of another individual to produce a new individual. We will see an example of this in later slides.
3. **MUTATION**. The mutation operator will replace a randomly selected element or set of elements in an individual. The objective of the mutation operator is to prevent stagnation in the population and introduce more diversity. There is a low probability that the process of mutation will occur. Every GA does not use mutation.

We have now looked at the high level structure of GAs. We will analyze GAs in more detail in subsequent slides. Firstly, we will introduce a classical problem called the 8-Queens problem that will be used to help illustrate the operation of a GA.

## 8-Queen Puzzle

- ▶ The objective of the 8-Queens puzzle is to place 8 chess queens on an 8x8 chessboard so that no two queens attack each other (This diagram does not show a solution state)



The 8-Queens problem will be used as an example to illustrate the operation of Genetic Algorithms over the coming slides. The 8-Queens can be formulated as a AI search problem. A state is any placement of the 8 queens on the chess board. The goal state is the arrangement of the 8 queens on the board in such a way that none can be attacked.

Therefore, a goal state should be configured so that no two queens share the same row, column, or diagonal.

The next few slides provide more detail on the specific operations of a Genetic Algorithm.

## Representation

- ▶ Before generating a population we must decide how to represent each individual (state) within the GA
- ▶ Representation is an important consideration
- ▶ There are many types of representation but typically each state is represented as a string over a finite alphabet

[Image Licence CC0](#)

Deciding on a representation for an individual is a important task in GAs. An appropriate representation can make an individual more meaningful. Also it is important to note that the chosen representation can reduce the search space considerably.

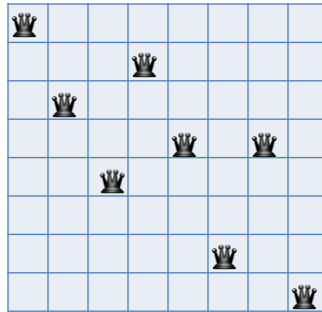
For example, let's consider a representation for the 8-Queens puzzle. We could use a representation that specifies the position of 8 queens, each in a column of 8 squares. We could use a binary representation or a numerical digit representation.

A binary representation would need three bits to express the location of each queen. Recall that one bit can represent two different things (0 and 1). Two bits can represent four different things and three bits can represent eight different things. With a total of 8 queens we would need  $(8 \times 3)$  24 bits to represent any given configuration.

Alternatively a state could be represented by 8 numerical digits (1-8), each with a valid range from 1 to 8. Therefore, a representation string using numerical digits would only be 8 digits long. Clearly, the numerical representation (length of 8) would be more intuitive than the binary representation (length of 24).

A chosen representation can reduce the search space considerably. In the representation outlined above for the 8-Queens puzzle we model the position of each queen on the board. Alternatively we could represent each individual square, which would dramatically increase the search space.

## Example of Representation



8	6	4	7	5	2	5	1
---	---	---	---	---	---	---	---

The eight digits are a representation of the state shown in the diagram. Each numerical digit specifies the row occupied by a queen in a particular column. The first digit indicates that the queen in the first column occupies row 8. The second indicates that the queen in the second column occupies the sixth row and so on.

## Population Generation and Fitness Evaluation

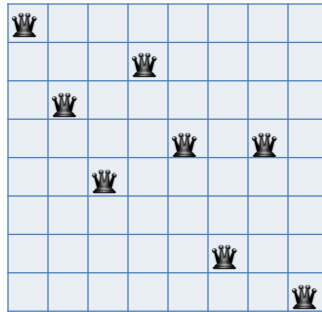
- ▶ Initially a GA will generate a random population
- ▶ Heuristic used to assess fitness of all individuals
- ▶ For the 8-Queen puzzle we use the number of non-attacking pairs of queens on the board

A GA will initially generate a population of individuals. Each individual is randomly generated and the population could consist of a few hundred individuals.

The next step is to evaluate the fitness of each individual through the use of an appropriate heuristic. For our 8-Queens problem we will use the number of non-attacking pairs of queens on the board as a heuristic function. Therefore the higher the value returned from the heuristic the fitter the individual.



## Fitness Evaluation Example



Fitness Value = 25

[Image Licence CC0](#)

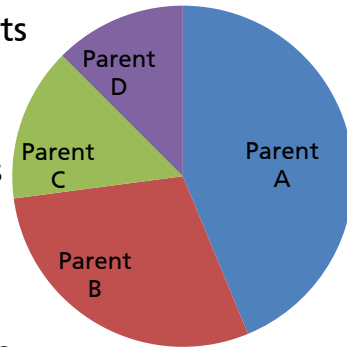
The heuristic chosen to calculate the fitness value of an individual is the number of non-attacking pairs of queens on the board. In the example, the total number of non-attacking queen pairs is 25. Remember one queen can attack another if it is on the same vertical, horizontal or diagonal line as the other queen.

## Evolving the Population

- ▶ The GA algorithm will repeatedly evolve a population using *Random Selection*, *Recombination* and *Mutation*
- ▶ We will examine each of these operations in the context of the 8-Queens puzzle

## Random Selection

- ▶ Random Selection is the process of choosing two individuals from the population to become parents in the reproduction stage
- ▶ The higher an individual's fitness the more likely they are to be chosen (perhaps multiple times)
- ▶ Probability of individual selection is proportionate to fitness (Roulette method)



*Random Selection* selects two individuals from the population. These individuals will be recombined in the next step to produce a new individual. The selection process is biased by the fitness of an individual. The fitter an individual the more likely it is that they are selected.

A common selection mechanism used is the roulette wheel approach where the probability of an individual being selected is proportionate to their fitness. For example, the slide depicts a simple Roulette wheel, where the probability of a parent being chosen is proportional to the amount of the wheel they occupy. Clearly, Parent A stands the best chance of being selected.

As a further example consider three separate 8-queen configurations which we will refer to as  $q_1$ ,  $q_2$  and  $q_3$ . The fitness of  $q_1$  is 10,  $q_2$  is 20,  $q_3$  is 30.

The probability of  $q_1$  being selected is  $10/(10+20+30) = 0.16$

The probability of  $q_2$  being selected is  $20/(10+20+30) = 0.33$

The probability of  $q_3$  being selected is  $30/(10+20+30) = 0.5$

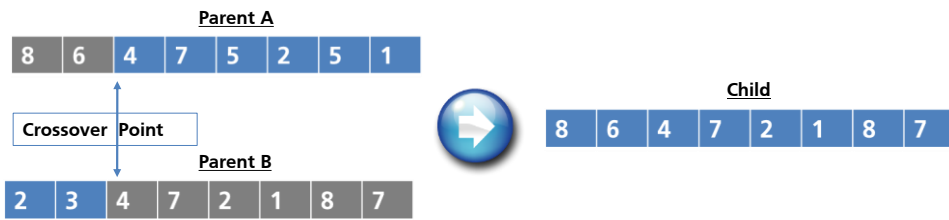
Notice that the fitter the individual the higher the probability of that individual being selected.

## Reproduction

- ▶ Reproduction involves the creation of a child individual by recombining elements of the two parents
- ▶ This is achieved by selecting a crossover point within the string representation of the parents
- ▶ The child is created by crossing over the parents at this crossover point

Reproduction involves the recombination of two adults to produce a child individual. Let's assume an individual is represented using a numerical string. The GA randomly selects a position within the string that becomes the *crossover point*. A child individual is created by crossing over the parent strings at the crossover point. For example if we select a crossover point of 3 then the child may receive the first three digits from the first parent and the remaining digits from the second parent. The next slides shows a more detailed example of reproduction in the context of the 8-Queens puzzle.

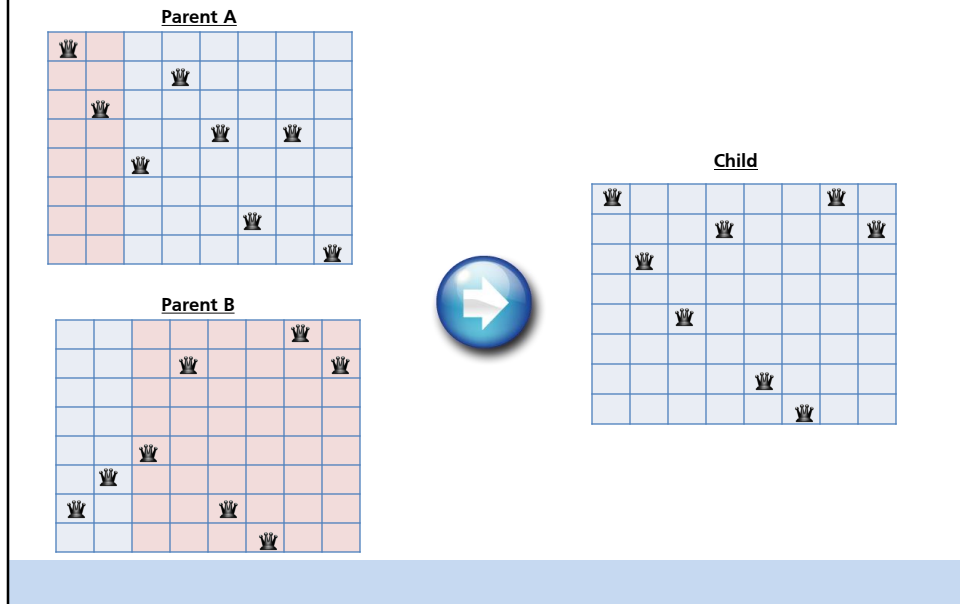
## Reproduction Example



- ▶ In the example a crossover point of 2 is selected
- ▶ Child individual is constructed by combining the first two digits of parents A with the last six last digits of parent B

Parent A and Parent B in the slide represent two different configurations for the 8-Queens problem. A graphical representation of these configuration is depicted in the next slide. The reproduction stage in the GA randomly selects a crossover point of 2. It then constructs a new individual by appending the last six digits of Parent B to the first two digits of Parent A.

## Reproduction Example



This slide depicts the actual chess board configuration for ParentA, ParentB and Child, which were represented by numerical strings in the previous slide. Notice the Child is composed of the placement of the queens from the first two columns of ParentA and the last six columns of ParentB.

## Mutation Example

- ▶ Mutation is a genetic operator used to maintain diversity in the population
- ▶ Mutation alters one or more elements of an individual to produce a new child individual
- ▶ Probability of mutation occurring should be low
- ▶ Not all GAs include a mutation operator

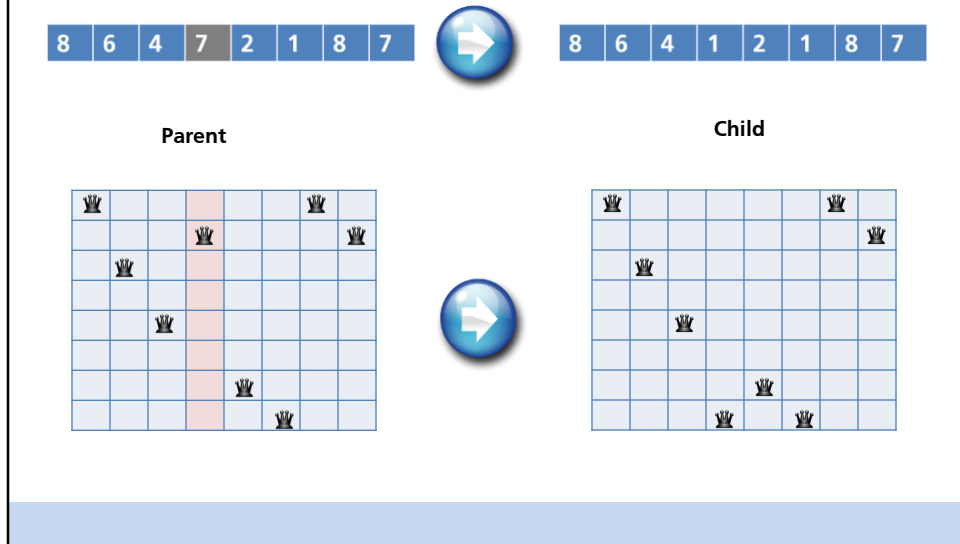
Image Licence CC0

The *mutation operator* in a GA is used to maintain genetic diversity from one generation of a population to the next. It is analogous to the occurrence of biological mutation where one or more gene values are altered within a chromosome. In mutation an element or set of elements within an individual can be altered to create a new child individual.

Consider an individual represented by the binary string “01100111”. The classical example of a mutation would be to flip one bit within the string. For example, if the mutation operator flips the first bit in the string we produce a new child individual “11100111”. The mutation operator could just as easily mutate a number of bits within the string.

Mutation is very useful for introducing diversity into a population. A user-defined probability can control the likelihood of mutation occurring. This probability should be set low. If it is set to high, it can disrupt the search turning it into a random search.

## Mutation Example



This slide uses the 8-Queens puzzle to illustrate the concept of mutation. The mutation operator will mutate one digit at random. In the 8-Queens puzzle this translates to choosing a queen at random and moving it to a random position.



## Summary

Informed  
Search and  
Heuristics

Explored the area of informed search algorithms

Illustrated and analysed the use of heuristics

Hill  
Climbing  
and  
Genetic  
Algorithms

Examined the operation of a Hill Climbing search algorithm

Introduced and analysed Genetic Algorithms

## References

ArsElectronica (2010) ASIMO [photo], ([CC BY-NC-ND 2.0](#)), available:  
<https://www.flickr.com/photos/36085842@N06/4945217670/> [accessed 18 Apr 2019].

All CC0 licensed images were accessed from Articulate 360's Content Library