Nature Inspired Optimisation for Delivery Problems
Chapter 3: Applying Evolution to Vehicle Routing Problems

Neil Urquhart

May 27, 2022

These slides are designed to accompany the book "Nature Inspired Optimisation for Delivery Problems : From Theory to the Real World".

`https://link.springer.com/book/10.1007/978-3-030-98108-2`

Evolutionary Computation

Introduction

- Evolutionary Computation (EC) describes a range of metaheuristics that may be used for problem solving ,which are inspired by Darwinian Evolution.
- EC is based around the concept of *evolving* a solution to a problem by applying random changes to a *population* of solutions over time.
- Solutions within the population can be evaluated in order to measure their quality (using a *fitness function*
- Where new solutions represent an improvement they replace solutions of a lesser quality
- Over time the quality of solutions within the population increases

Evolutionary Algorithms

- The term Evolutionary Algorithm covers a wide variety of algorithms based around the evolutionary metaphor.
- In order to allow evolution to take place a *fitness function* is required to evaluate the solution
- A simple fitness function might return a single value in the range 0 to ∞, 0 representing the optimal solution and the values increasing as the solution becomes less optimal.

Fitness functions: example

Let us consider a simple example; the max ones problem, the objective is to evolve a binary string of all 1s, the fitness f being the sum of all of the 1s in the string.

11011101 has a fitness of 6
11011100 has fitness of 5

Over time the evolutionary process will promote strings with more 1s until finally a string (11111111) is evolved where $f = 8$.

Populations and the Evolutionary Cycle

- A set of solutions, known as the *population* is maintained.
- Each member of the population is evaluated and its *fitness* calculated
    - any two (or more) members of the population may be compared and the most fit solution identified.
- The fitness of solutions within the population is improved over time
    - Select individuals (parents) with a bias towards higher fitness
    - Create new individuals (children) by copying the parents and applying some random changes
    - Add the children back into the population, replacing members with a worse fitness
- Over time the fitness of the population improves until it reaches a level that represents an acceptable solution to the end user.

The Evolutionary Cycle

```
Procedure Evolve()
Initialise population
while !finished do
    select parents
    create children
    evaluate children
    replace children into population
end
Return best in population
```

Creating Children

Having selected parents, new *child* solutions are produced using a combination of operators:

- **Crossover** (Sometimes known as recombination) Crossover takes two (or more) parents and creates a child based on features from both parents.
- **Cloning** Creates a child based on copying a single parent
- **Mutation** Takes a child created by recombination or cloning and introduces a small random change.

Having created children they are copied back into the main population, replacing existing population members, with lesser fitness values.

Applying Evolution to the CVRP

Applying Evolution to the CVRP

- **Representation** A permutation of visits which will form a grand tour, this is known as a *genotype*.
- **Fitness function** The fitness function splits the grand tour permutation into the phenotype which comprises the set of vehicle routes. The fitness value is the length of the routes within the phenotype.
- **Selection Strategy** Tournament selection picks k individuals randomly from the population, the individual with the the best fitness of those picked is then selected.
- **Crossover operator** The crossover operator uses a form of *single point* crossover which copies a randomly selected section of the genotype from the first parent (from the start to a randomly selected point) and then copies the remaining material from the second parent.

...

Applying Evolution to the CVRP ... cont

- **Mutation Operator** The mutation operator randomly selects a single gene and moves to a random location within the genotype. This has the effect of moving a customer visit with the delivery plan, possibly between routes.
- **Replacement Strategy** The replacement strategy performs a inverse tournament selection and the looser is replaced by the child, assuming the child has a better fitness.
- **Halting Criterion** The algorithm will halt after a fixed number of solutions have been evaluated.

Decoding and fitness functions

Our scheme uses what is know as an *indirect representation*, solutions are stored and manipulated as genotypes which are then decoded into solutions (phenotypes) which represent the actual solution .

- The decoding stage within the fitness function splits the grand tour into sub tours using the capacity constraint in the same manner as the grand tour described in chapter 2.
- The decoding stage can become quite complex as it ensures that the solution it is constructing is feasible.
- You may wish to think of the phenotype as containing a set of instructions for the decoder to construct the phenotype containing the solution.

Having constructed the phenotype the fitness of the individual may now be calculated, in this example it is the sum of the sub tour lengths.

Creating New Solutions

Two types of operator are typically used to create new solutions:

- Crossover creates a new child genotype incorporating elements of both parents.
- Mutation introduces a random change into a genotype

Usually crossover will be applied to two parents, creating a new child solution, which will then be subject to mutation.

Crossover: A Simple Example

Suppose we have 2 parents which are permutations of visits A-F

Parent 1 :	A	B	C	D	E	F
Parent 1 :	F	E	B	A	C	D

Select two random points (e.g. 2 and 4) - copy the section of p1 to the child:

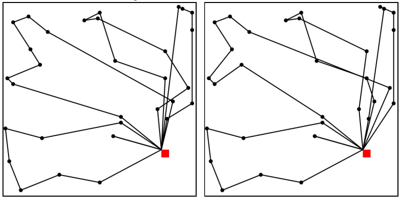
Child :		B	C	D		

Now copy items from P2 in the order they appear to complete the child (don't copy anything that is already in the child)

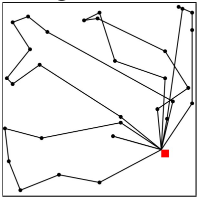
Child :	F	B	C	D	E	A

Crossover

Two parent solutions:



The child phenotype resulting from the crossover of the parents:



Care must be taken when coding the mutation operator to ensure that the child is still a valid permutation!

Mutation

- The mutation operator selects an element at random from within the genotype.
- The selected element is moved to a randomly selected position within the genotype

For example:

Genotype:	A	B	C	D	E	F

Select and remove an individual at random:

Genotype:	A	B	C	E	F

Add the individual at a randomly selected position:

Genotype:	A	D	B	C	E	F

Creating new solutions

```
Procedure CreateChild()
if $randBoolean() == true$ then
    parent = tournament(2)
    child = copy(parent)
end
else
    parent1 = tournament(2)
    parent2 = tournament(2)
    child = crossover(parent1,parent2)
end
child = mutate(child)
evaluate(child)
Return child
```

Using EAs

- Because of the random elements in selection, crossover and mutation EAs are *stochastic*.
- Repeated runs of an EA will not necessarily produce the same result.
- It is good practice to run an EA several times and use the best result obtained

When carrying out a more formal comparison of stochastic algorithms techniques such as *t-tests* may be employed.

EA Parameter Setting and Operator Selection

- The performance of an EA may be influenced by parameters such as:
  - Population size
  - Mutation rate
  - Crossover rate
  - Evaluation budget (no of solutions to evaluate before halting)
- Literature will suggest possible settings (as will the code supplied with the book)
- Careful experimentation may also be useful
- A number of researchers have looked at ways of optimising the parameters

EA Parameter Setting and Operator Selection

- There are many operators and representations that can be used with an EA
- Look at the literature to see what has been used successfully on similar problems
- Multiple mutation (and crossover) operators can be used in the one algorithm
- Carefully experiment to see what works for your problem

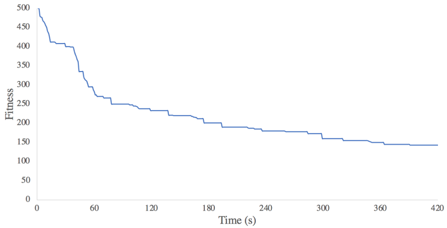
Summary

Summary

- This chapter has introduced the Capacitated Vehicle Routing Problem (CVRP)
- The CVRP requires the construction of *network* of routes, as formulated here, is still a single-objective problem based on minimising distance.
- Evolutionary Algorithms are a nature-inspired technique (based on concepts from Darwinian Evolution) which have been applied to many optimisation problems in different domains.

Summary

As an EA runs, it evolves solutions with a better fitness, through selection, crossover, mutation and replacement.



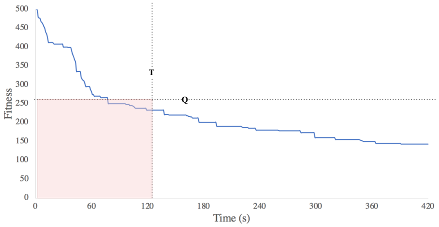
A typical EA run, showing how the fitness of the best individual in the population improves over time.

Summary

When considering a real-world application we can envisage two subjective variables which will affect the ability of our algorithm to satisfy the user:

- Q - the quality threshold beyond which the user will accept a solution
- T - the time that they are willing to wait for a solution

An example of how Q and T may relate to the run time of the EA is shown below.


Fitness
T
Q
Time (s)