6G6Z1109: Software Agents and

Term 2, Lecture 6: velling Salesman Problem

Travelling Salesman Problem (TSP)

- We might also refer to this (in gender-neutral terms) as the "Delivery Driver Problem"
- TSP is the c title, so we will go with that
- Archetypal NP-complet problem (depending on formulation of question)
- Used as a test-bed for many different optimisation algorithms

TSP formulation

- Given a set of *cities* distributed in space, find the *shortest* route that v *only once* and then returns to the start poin
- That is, find a *circular tour* of minimum length

Wikipedia

Why is the TSP interesting?

- Apart from being inherently hard (and therefore challenging), the TSP finds many applications in various domains such as
  - Transporta hool bus to pick up *n* childr eliveries to *n* locations)
  - Circuit board drilling (minimise "travel time" for drill head to make *n* holes in a board)
  - Crew scheduling, advertisement loop placement, etc. etc.

TSP formulation

- Some variants of the TSP include the edges (corresponding to a set of roads)
- We focus on TSP, which simply distributes cities in 2D space, and assumes that there exists a direct straight road between any possible pair of cities)

TSP hardness

- Why is the TSP hard?
- For n cities (*n*>=3), there are (n-1)!/2
- So, for 20 citi
  (19*18*17*16...*1)/2
  possible tours
- 60,822,550,204,416,000
  for this particular instance

TSP hardness

- Why (n-1)!/2 possible tours?
- For n=20, if we start at arbitrary city, A, we then have 19 to choose from…
- Once we move to the next city, we then have 18 to choose from, and so on.... (which gives us the
- n-1, because we ha to make (as, by def return to the start point)
- We *divide the number of tours by two*, because we don't care about the *direction* of the tour (that is, ACDBE is considered identical to EBDCA)
- A tour is a *permutation* of cities (that is, an ordering of cities)

A

Lee Jacobson

TSP encoding

- Given that a tour is simply an ordering of cities, it seems natural to use a string of city la to represent a solution
- Each individual, therefore, starts with a shuffled (randomised) version of the city list as its genome

TSP evaluation

- We then need to calculate the *length* of the tour encoded by the list of cities
- This gives u our (where lower (ie. sh nsidered better)
- So, for the sequence DBCAE, we would add together distances D-B, B-C, A-E, then E back to D

Takes string of city labels as input

New class, `Map`, which is a version of the `Graph` class

For each pair of city labels (including “wrap d to start”)….

New class, `Map`, which is a version of the `Graph` class

appropriate nodes from the ordered list, calculate distance between them, and add to running total

New class, `Map`, which is a version of the `Graph` class

New class, `Map`, which is a version of the `Graph` class

Distance between two points

Added to `Node` class

Calculates *absolute* distances in X and Y (that way, it doesn't matter which heading the other node is from the current node)

Then uses Pythagoras' theorem to calculate distance

Proces


```
i_st
()
functi
```

Problem (1): Crossover

Imagine we have two genomes:

ADBCE

DBCEA

Crossing them d one-point

operator will generate *ille* …

AD | BCE

DB | CEA = ADCEA DBBCE

Visit some cities more than once, don't visit all cities

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order they appear in the second parent


```
P1: IEHDGCFBJA
P2: ABCDEFGHIJ
Ch: AEHDGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order they appear in the second parent


```
P1:  IEHDGCFBJA
P2:  ABCDEFGHIJ
Ch:      DGCFB
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order they appear in the second parent


```
P1:  IEHDGCAFBJA
P2:  ABCDEFGHIJ
Ch:      DGCFBI
```

Is I alrea
in child?
so include

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order appear in the second parent


```
P1: IEHDGCAFBJA
P2: ABCDEFGHIJ
Ch:     DGCFBIJ
```

Is J alre
in child
so inclu

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order they appear in the second parent

Is A already in child? N, so include

```
P1: DGCFBIJA
P2: ABCDEFGHIJ
Ch: A  DGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order they appear in the second parent

Is B already in child? Y, so skip

```
P1:  ABCDEFGHIJ
P2:  ABCDEFGHIJ
Ch:  A   DGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order appear in the second parent

Is C alread in child? Y, so skip

```
P1: I  BCJA
P2: ABCDEFGHIJ
Ch: A  DGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order appear in the second parent

Is D alre
in child? Y,
so skip

```
P1: IE   JA
P2: ABCDEFGHIJ
Ch: A  DGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order appear in the second parent

Is E al
in child? N
so include

```
P1: IEH      A
P2: ABCDEFGHIJ
Ch: AE DGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order they appear in the second parent

Is in child? Y, so skip

```
P1: IEHDGCFBAJ
P2: ABCDEFGHIJ
Ch: AE DGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order appear in the second parent

in child? y
so skip

```
P1: IEHDG
P2: ABCDEFGHIJ
Ch: AE DGCFBIJ
```

Solution: new crossover operator

Ordered crossover: take a random subsequence of cities from first parent, then fill in *remaining* cities in the order appear in the second parent

in child? N
so include

```
P1: IEHDGCFBAJ
P2: ABCDEFGHIJ
Ch: AEHDGCFBIJ
```

Problem (2): fitness values

- If we are using fitness values that can take a wide *absolute* range, some fitnesses can come to dominate, leading to early stagnation
- Also, as we sa of very small fitness values ating pool
- Also, situation is complicate t that we are *minimising* the function
- Solution: use a *different selection operator* that allows us to easily compare fitnesses (to minimise, and to allow for very small fitnesses) and doesn't suffer from excessive selective pressure

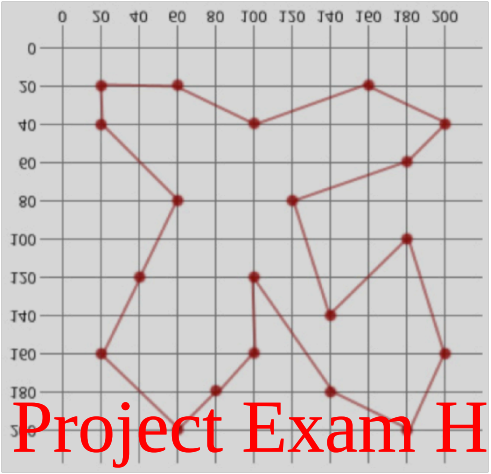
Tournament selection

- Essentially, sample a random selection of *n* population members, then just pick the one with the *lowest tour length* (i.e., the "winner)
- *n* is called th d can be tuned like other pa
- We can also tune the *sele* *sure* of the operator, by only selecting the winner a *certain proportion of the time* (otherwise, we just select a random member of the tournament)
- In practice, this prevents early stagnation

Guarantees the first tour we check will be the shortest, by default

Build the tournament

winner/
return random
individual



n=940

However, our solution uses a population of 500, rather than 50. Using a population of 50 tends to generate inferior solutions.

= Population size can dramatically alter performance…

Next lecture

- Next week: Local search
- This week's lab: Implement ordered crossover