COMP 3522

Object Oriented Programming in C++
Assignment 2

GENETIC ALGORITHMS

Genetic algorithms

- Computing has been helpful in many problem domains
- Many problem domains have, in turn, lent problem solving strategies to computing
- Genetic algorithms are inspired by the process of natural selection described in contemporary biology
- Useful for large problems with solutions that are difficult to find
- Useful for optimization, search problems.

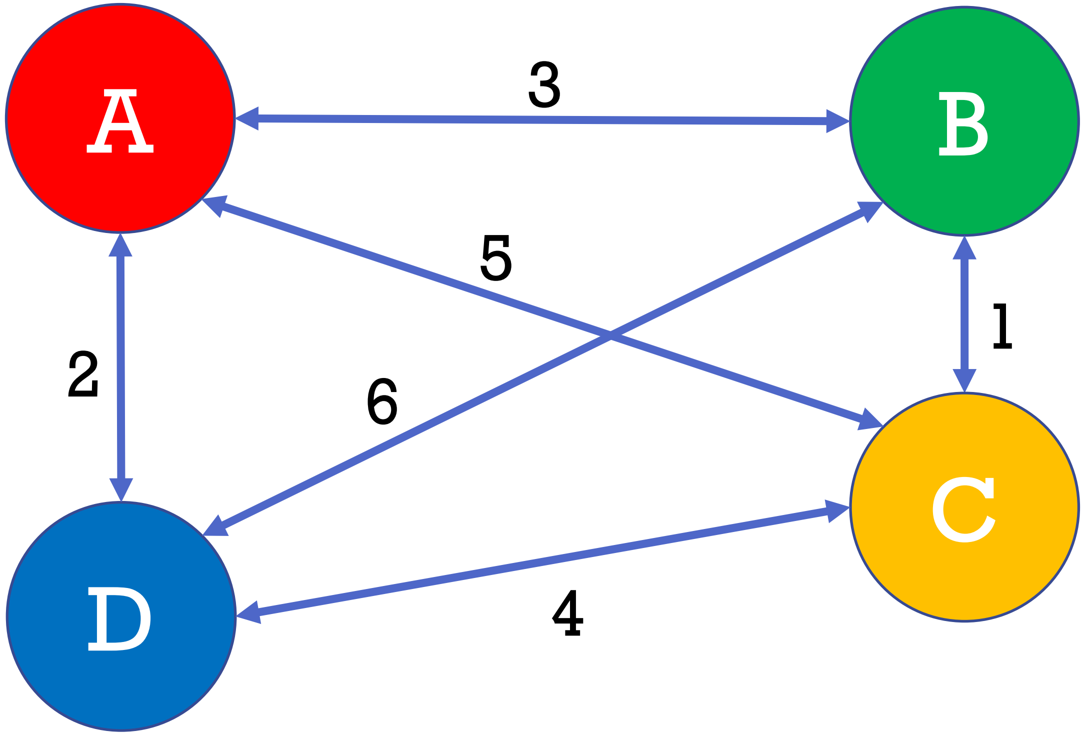
Genetic algorithm

```
{
        initialize population
        evaluate population
        while (termination criteria not reached)
        {
                select solutions for next population
                perform crossover and mutation
                evaluate population
        }
}
```

GENETIC ALGORITHM: Travelling salesperson

Travelling salesperson problem

- Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?



Travelling salesperson problem

- Input:
  - List of cities to visit
- Requirements:
  - Visit all the cities
    - Return to original city
  - Minimize travelling distance

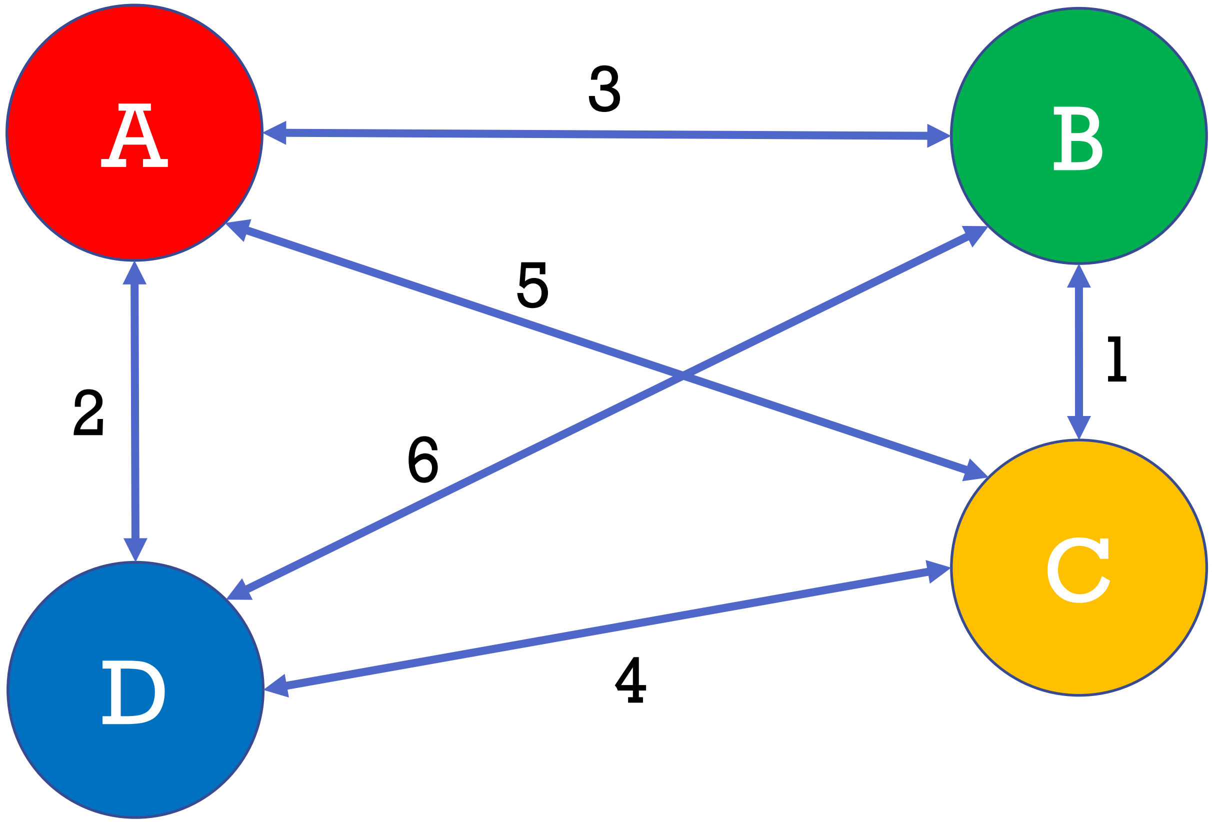
This sounds easy (we can do this by hand with a few cities)

O(**n!**) – this becomes impractical with 20 cities

What if we have 200 cities, or 200,000 cities, or ALL the cities and towns and villages in the world?

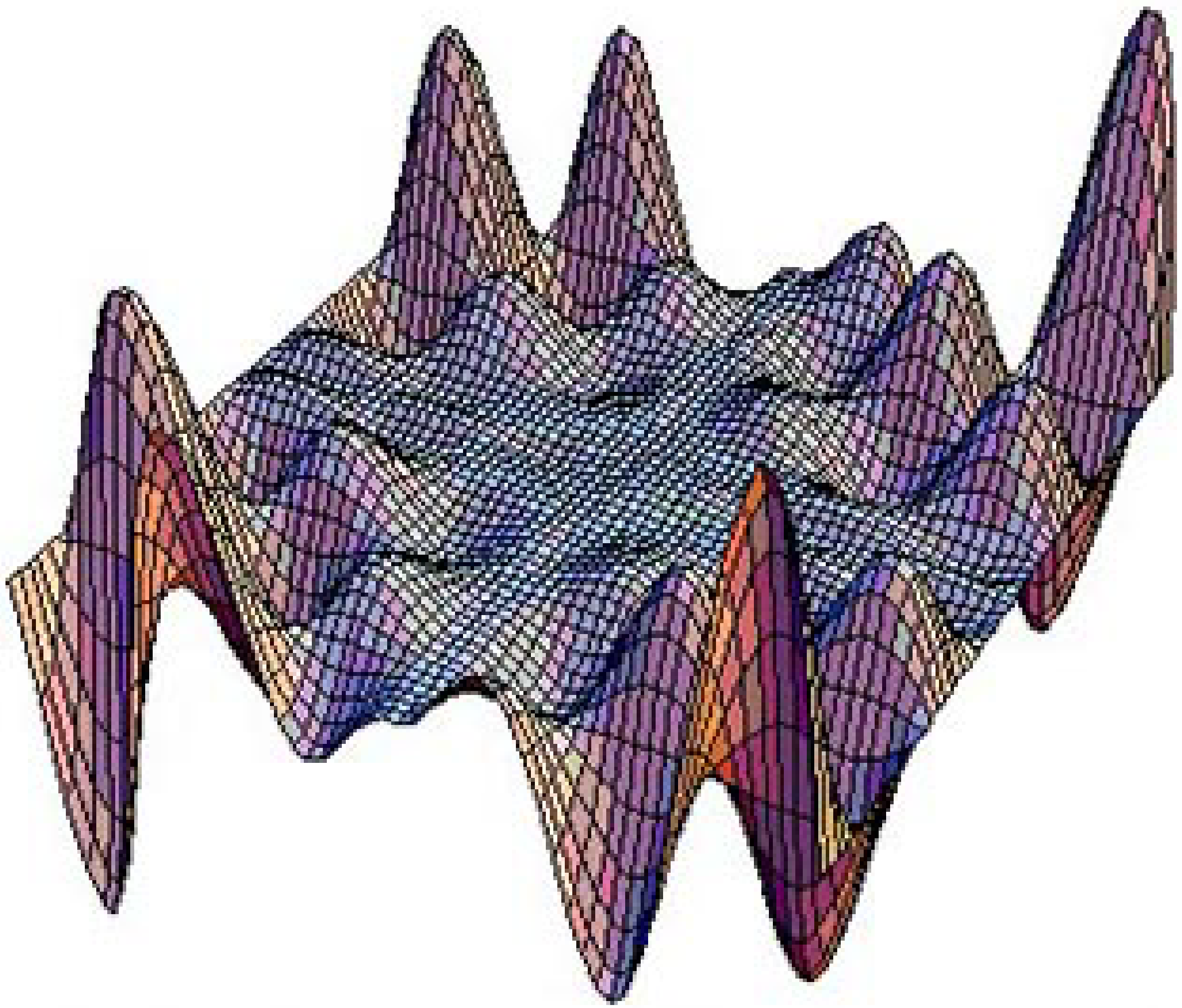
Let’s travel around 4 cities

- **ABCDA = 3+1+4+2 = 10**
- ACBDA = 5+1+6+2 = 14
- ADBCA = 2+6+1+5 = 14
- **ADCBA = 2+4+1+3 = 10**
- …
- DBCAD = 6+1+5+2=14
- DBACD = 6+3+5+4=18



- 24 possibilities = 4! (4 factorial: 4 x 3 x 2 x 1)

Imagine the 'solutionscape'



http://classes.yale.edu/fractals/CA/GA/Fitness/FitnessLandscape.gif

Some tours are very short

Some tours are very long

There are too many to find a global solution.

OUR GENETIC ALGORITHM: Terminology

Some terms we are using

City: A location that has a name and x/y coordinates.

Cities_to_visit: an invariant (unchanging) list of City structs that we want to visit. The "master list"

ABCD

Tour: a list of pointers to the cities we want to visit. We can shuffle the pointers easily to compare different orderings of cities without modifying the "master list"

D*B*C*A*

Population: a collection of candidate Tours. We keep the population "sorted," i.e., the "fittest" tours are at the front of the list.

D*B*C*A*
B*D*A*C*,
C*A*D*B*...

Some terms we are using

Fitness: Each candidate Tour in the population has a **fitness**, i.e., how "good" it is. For us, a fit Tour has a short travel distance. A Tour with a **shorter distance has better fitness**.

D*B*C*A* = 6 + 1 + 5 = **12**

Elite: Each generation, we can designate one or more Tours that are so amazing they don't cross, they get carried over to the next 'generation'. These Tours are "**elite**."

D*B*C*,A* = 12
A*B* C*D* = 10 → **ELITE!**
C*A*D*B* = 18

Parents: Each iteration, we select some non-elite **parents** from the Population of Tours and use the parents' contents to generate a new Tour for the next iteration.

D*B*C*,A*
B*D*A*,C*
C*A*D*B*

Randomly select non-elite **Parents** from previous population

Some terms we are using

Crosses and Crossover: Each generation we create new Tours by crossing "parents." The crossover algorithm is basic.

D*B*C*A* x C*A*D*B* -> D*C*A*B*

Mutation: Each iteration, we randomly "mix up" a few of the Tours in our population. This mimics the random mutations that take place as cells divide, etc.

D*C*A*B* -> D*A*C*B*

Mutation rate: If we 'roll' less than the rate, we swap a few cities in the Tour being mutated.

OUR GENETIC ALGORITHM

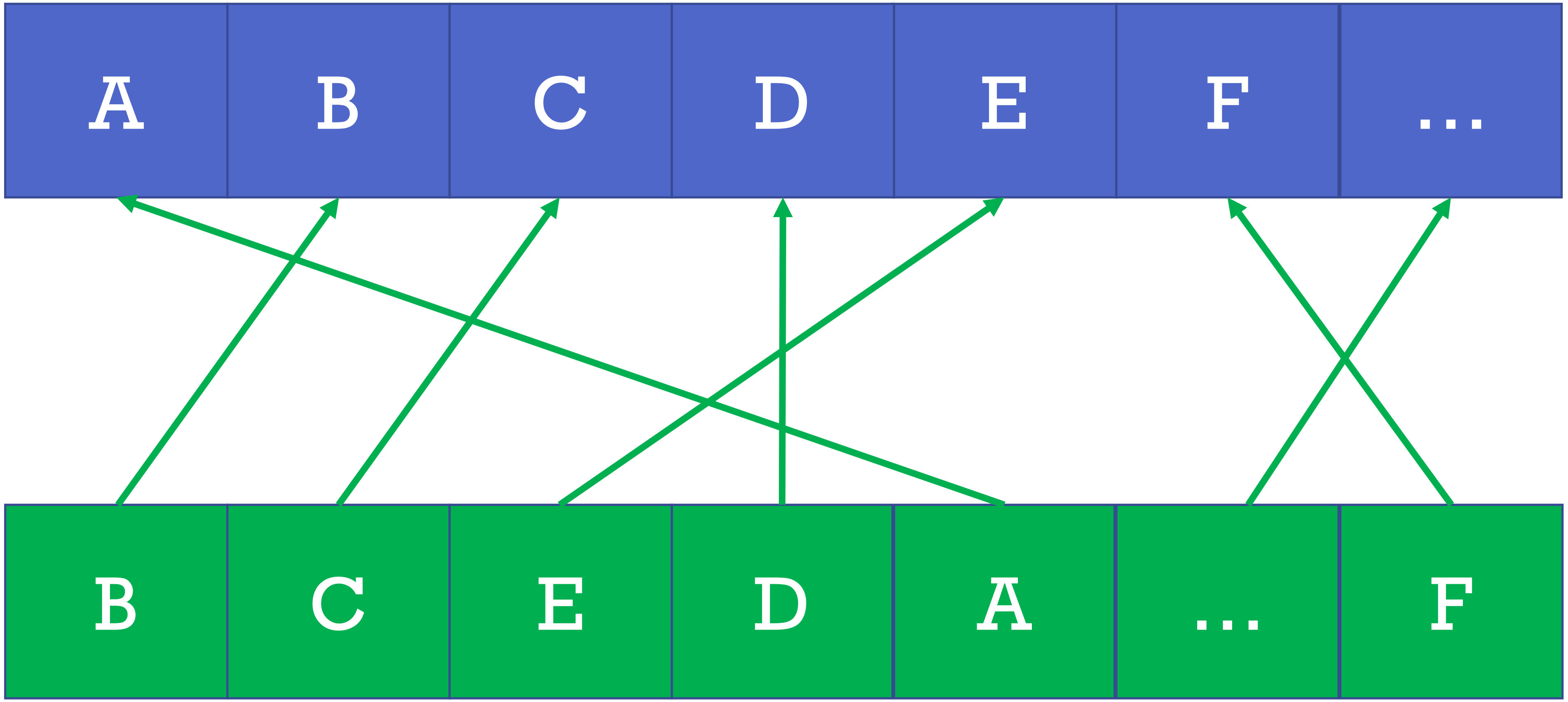
Our genetic algorithm

```
{
    create cities and tours
    evaluate tours' fitness
    while ((fitness < improvement) and (iterations < 1000))
    {
        move elite to front
        perform crossover and mutation of tours
        evaluate tours' fitness
    }
}
```

Our algorithm

1. Create our master list of cities named A, B, C, …, R, S, T. Our master list is 20 Cities long.
2. Create a Population of Tours. The Population contains 30 candidate Tours. Each Tour contains pointers that point to the cities in the master list. Each Tour is shuffled randomly.

Master city list – doesn't change



Tour – pointers to cities

Have 30 tours each with random pointers to cities

Our algorithm

3. Find the shortest travelling distance in the randomly shuffled Tours. That's our starting point.

Population – list of tours

Tour 1 = 100 cost	
Tour 2 = 200 cost	
…	
Tour 29 = 70 cost	ELITE!
Tour 30 = 150 cost	

Our algorithm

4. (Loop) While we haven't reached our goal (or still have iterations)
    1. Find the best tour in the Population, call it an Elite, and move it to the front of the list so we can keep an eye on it.

Population – list of tours

Tour 29 = 70 cost	Move ELITE to front
Tour 2 = 200 cost	
…	
Tour 28 = 100 cost	
Tour 30 = 150 cost	

- Identified the best tour of the existing tours
- Next create a new list of tours based on the existing tours

OUR GENETIC ALGORITHM: Crossing

Our algorithm

4. (Loop) While we haven't reached our goal (or still have iterations)
    2. Create a temporary list of Tours called Crosses.
    3. Generate a new Tour by crossing parents for each remaining Tour in the the Original population

Original Population
Elite Tour = 70 cost
Tour 2 = 200 cost
…
Tour 28 = 100 cost
Tour 30 = 150 cost

→

Crosses – new population
Elite Tour = 70 cost
?
?
?
?

Step 4.3 Crossing parents

Original Population

Elite Tour = 70 cost
Tour 2 = 200 cost
…
Tour 28 = 100 cost
Tour 30 = 150 cost

Set 1

Tour 26 = 500 cost
Tour 11 = 700 cost
Tour 27 = 800 cost
Tour 14 = 900 cost
Tour 30 = 150 cost

Set 2

Tour 13 = 700 cost
Tour 2 = 200 cost
Tour 12 = 300 cost
Tour 10 = 600 cost
Tour 9 = 350 cost

- Pick **two sets** of **5 random tours** from the Original population

Step 4.3 Crossing parents

- Find the **fittest tour** in each set.
- These **two parents** will be crossed to generate a new child

Set 1

Tour 26 = 500 cost	
Tour 11 = 700 cost	
Tour 27 = 800 cost	
Tour 14 = 900 cost	
Tour 30 = 150 cost	**Parent 1**

Set 2

Tour 13 = 700 cost	
Tour 2 = 200 cost	**Parent 2**
Tour 12 = 300 cost	
Tour 10 = 600 cost	
Tour 9 = 350 cost	

Step 4.3 Crossing parents

- WARNING – All tours should have **20 cities** in this example:
- ie:
  - Tour 30 [A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T]
  - Tour 2 [D,C,A,B,E,S,G,H,I,T,K,L,N,M,O,P,F,R,Q,J]
- But we're shortening it to **5 cities** a tour for demonstration purposes

Parent 1

Tour 30 = 150 cost
A B C D E

Parent 2

Tour 2 = 200 cost
DCABE

Child tour
?????

Step 4.3 Crossing parents

- Pick a random index and copy all cities up to and including that index from parent 1
  - Randomly pick **index 1**. Start from beginning of **Parent 1**, copy everything up to and including index 1 from **Parent 1** to **Child**

Parent 1	Parent 2
Tour 30 = 150 cost A B C D E	Tour 2 = 200 cost DCABE

Child tour
?????

Step 4.3 Crossing parents

- Pick a random index and copy all cities up to and including that index from parent 1
  - Randomly pick index 1. Start from beginning of **Parent 1**, copy everything up to and including index 1 from **Parent 1** to Child

Parent 1

Tour 30 = 150 cost
A B C D E

Parent 2

Tour 2 = 200 cost
DCABE

Child tour
A????

Step 4.3 Crossing parents

- Pick a random index and copy all cities up to and including that index from parent 1
  - Randomly pick index 1. Start from beginning of **Parent 1**, copy everything up to and including index 1 from **Parent 1** to Child

Parent 1

Tour 30 = 150 cost

A B C D E

Parent 2

Tour 2 = 200 cost

DCABE

Child Tour

AB???

Step 4.3 Crossing parents

- After hitting index 1 of parent 1, start from beginning of parent 2
  - Skip duplicate cities in **Parent 2** and **Child**. Copy over non-duplicate cities

Parent 1

Tour 30 = 150 cost
A B C D E

Parent 2

Tour 2 = 200 cost
DCABE

Child Tour
ABD??

Step 4.3 Crossing parents

- After hitting index 1 of parent 1, start from beginning of parent 2
  - Skip duplicate cities in **Parent 2** and **Child**. Copy over non-duplicate cities

Parent 1

Tour 30 = 150 cost
A B C D E

Parent 2

Tour 2 = 200 cost
DCABE

Child Tour
ABDC?

Step 4.3 Crossing parents

- After hitting index 1 of parent 1, start from beginning of parent 2
  - Skip duplicate cities in **Parent 2** and **Child**. Copy over non-duplicate cities

Parent 1

Tour 30 = 150 cost

A B C D E

Parent 2

Tour 2 = 200 cost

DCABE

Skip duplicate A

Child Tour

ABDC?

Step 4.3 Crossing parents

- After hitting index 1 of parent 1, start from beginning of parent 2
  - Skip duplicate cities in **Parent 2** and **Child**. Copy over non-duplicate cities

Parent 1

Tour 30 = 150 cost
A B C D E

Parent 2

Tour 2 = 200 cost
DCABE

Skip duplicate B

Child Tour
ABDC?

Step 4.3 Crossing parents

- After hitting index 1 of parent 1, start from beginning of parent 2
  - Skip duplicate cities in **Parent 2** and **Child**. Copy over non-duplicate cities

Parent 1

Tour 30 = 150 cost
A B C D E

Parent 2

Tour 2 = 200 cost
DCABE

Child Tour
ABDCE

Our algorithm – insert merged tours

- This is our **new merged tour**. Repeat previous steps for the rest of the new population of tours

Merged Tour 1
ABDCE

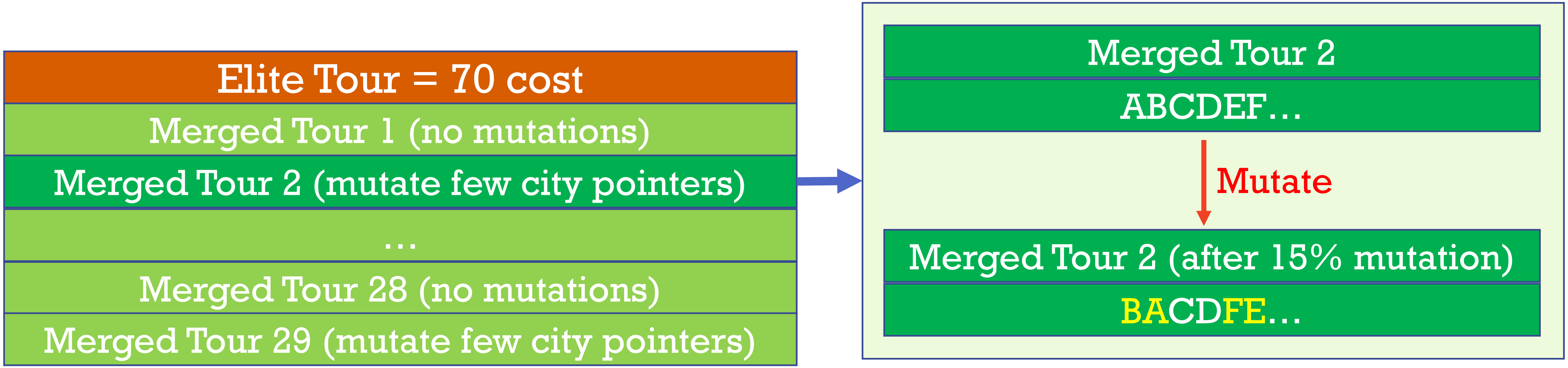
4. (Loop) While we haven't reached our goal (or still have iterations)
    4. Replace all the Tours in our Population (except the Elite Tour) with the new crosses

Elite Tour = 70 cost
Merged Tour 1
Merged Tour 2
…
Merged Tour 28
Merged Tour 29

OUR GENETIC ALGORITHM: Mutating

Our algorithm – mutate population

4. (Loop) While we haven't reached our goal (or still have iterations)
    5. Mutate some of the population (except the Elite) by swapping around some of the Cities in each Tour.



Our algorithm – evaluate fitness

4. (Loop) While we haven't reached our goal (or still have iterations)
   6. Evaluate the fitness (distance) and report it.

Elite Tour = 70 cost
Merged Tour 1
Merged Tour 2
Merged Tour …
Merged Tour 28
Merged Tour 29

Elite Tour = 70 cost	
Merged Tour 1 = 300 cost	
Merged Tour 2 = 50 cost	ELITE!
Merged Tour …	
Merged Tour 28 = 170 cost	
Merged Tour 29 = 210 cost	

5. Bam. You just implemented a genetic algorithm in C++!