Artificial Intelligence (AI)

CCS-3880 – 3rd Semester 2023

CO4: Genetic Algorithms (Global Search)

Dr. Abdullah Alshanqiti

Genetic Algorithms

CO4: Investigate the performance of local, global, and optimized search methods.

Outline - Heuristic Search

- Greedy Best First Search,
- A* Algorithm
- Local Search Algorithms
  - Hill-climbing search
  - Gradient Descent
  - Simulated annealing (suited for either local or global search)
- Global Search Algorithms
  - Genetic Algorithms

Genetic Algorithms

So far, we have now studied different techniques based on the idea of a ***Local Search***

And also, we discussed the *Simulated Annealing* which relies on controlled random movement

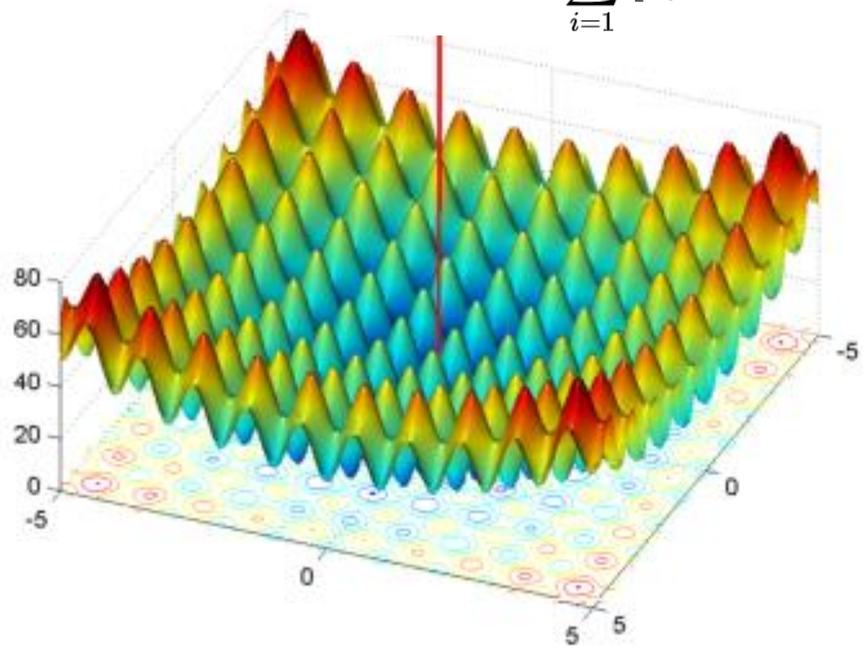
Main goal

- To avoid getting stuck in local minima

Additional goals

- Explore a larger part of the search space
- Attempt to find the global (not just a local) optimum

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right]$$



Rastrigin function

Genetic Algorithms

- The genetic algorithm isn’t really a single algorithm, but a collection of algorithms and techniques that can be used to solve a variety of problems in a number of different problem domains.
- A successor state is generated by combining two parent states
- Start with k randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by *selection*, *crossover*, and *mutation*

Let's first discuss shortly its evolution.

Genetic Algorithms

- In early evolutionary strategy, the population size was restricted to two members, the parent and child.
- The child member was modified in a random way (*a form of mutation*), and whichever member was more fit (parent or child) was then allowed to propagate to the next generation as the parent.

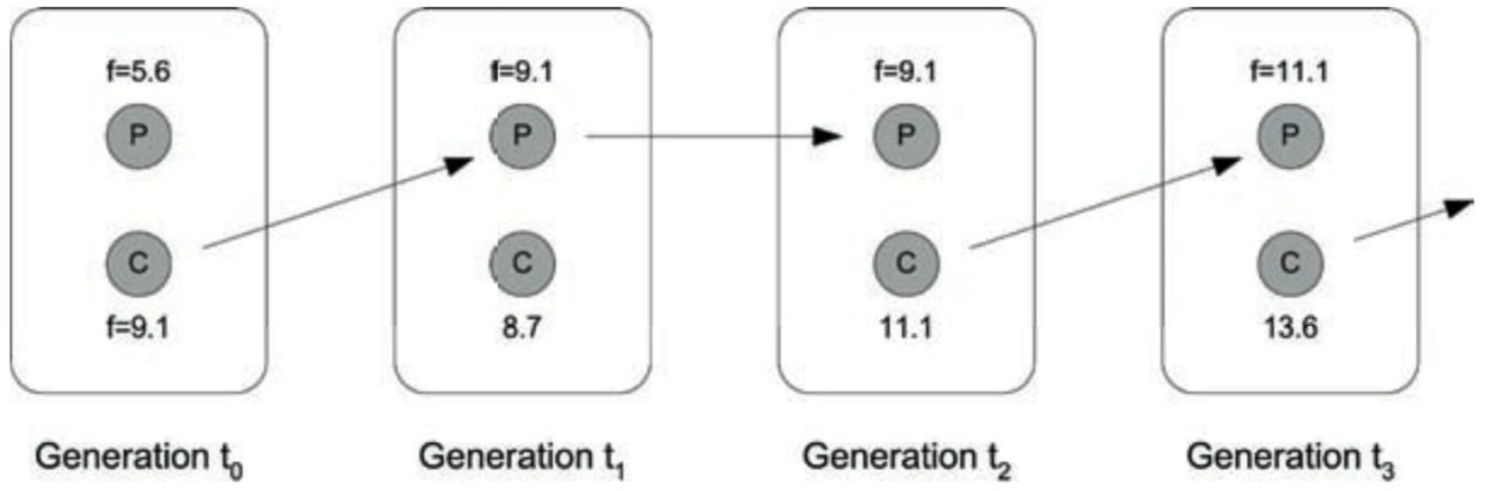


FIGURE 7.1: **Demonstrating the simple two member evolutionary strategy.**

Genetic Algorithms

- John Holland introduced the idea of genetic algorithms in the 1960s as a population-based algorithm.
- Evolutionary strategies used *mutation* as a way to search the solution space, Holland's genetic algorithm extended this with additional operators straight from biology.
- Potential solutions (or chromosomes) are represented as strings of bits instead of real values.
- In addition to mutation, Holland also used *crossover* and *inversion* to navigate the solution space.

  - All living organisms consist of cells, where each cell contains a set of chromosomes (strings of DNA).
  - Each chromosome is made up of genes, each of which can encode a type (behavioral or physical).
  - These chromosomes serve as the basis for genetic algorithms, where a potential solution is defined as a chromosome, and the individual elements of the solution are the genes.

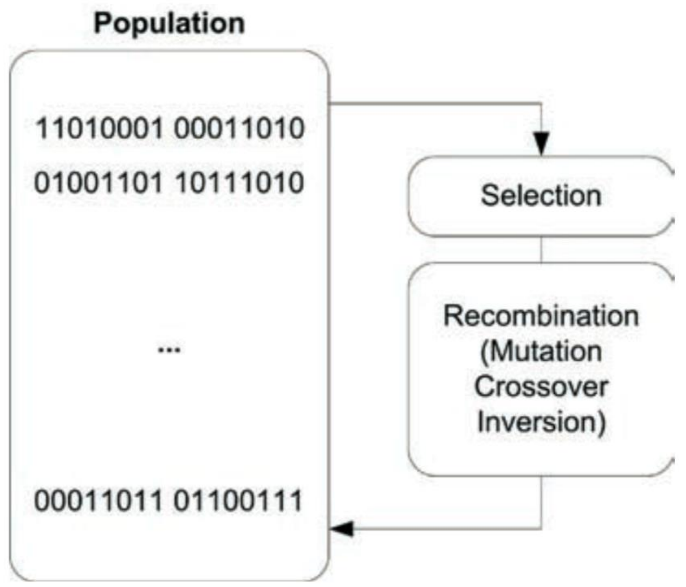


FIGURE 7.3: **Holland's bit-string genetic algorithm.**

Genetic Algorithms

- One attempt to understand why genetic algorithms work is called the Building-Block Hypothesis (BBH).
- This specifies, for binary GA, that the crossover operation (splitting two chromosomes and then swapping the tails) improves the solution.
- One can think of this as genetic repair, where fit building blocks are combined to produce higher fitness solutions.
- Additionally, using fitness selection (higher fit members are selected more often), less fit members and their corresponding building blocks die out and thus increasing the overall fitness of the population.

Genetic Algorithms

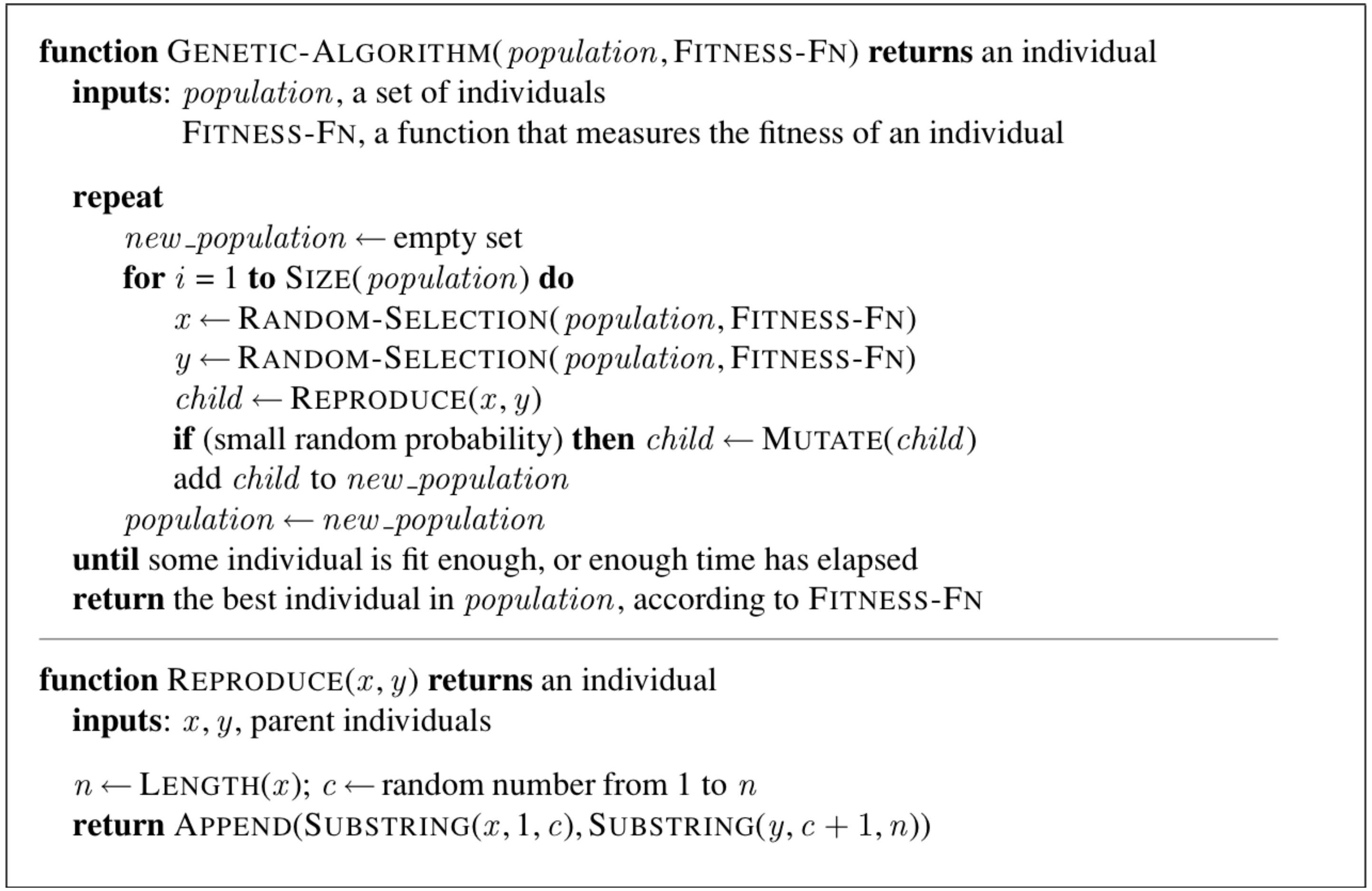
- Provide efficient, effective techniques for optimization and machine learning applications.
- Widely-used today in business, scientific and engineering circles
  - Function Optimization
  - AI (Games, Pattern recognition ...)
  - Basic idea: intelligent exploration of the search space based on random search
- Classical GA: Binary encoding

Genetic Algorithms

A problem to solve ..

▪ Encoding technique	(*gene, chromosome*)
▪ Initialization procedure	(*creation*)
▪ Selection of parents	(*reproduction*)
▪ Genetic operators	(*mutation, recombination*, ..)
▪ Evaluation function	(*environment*)

Genetic Algorithms

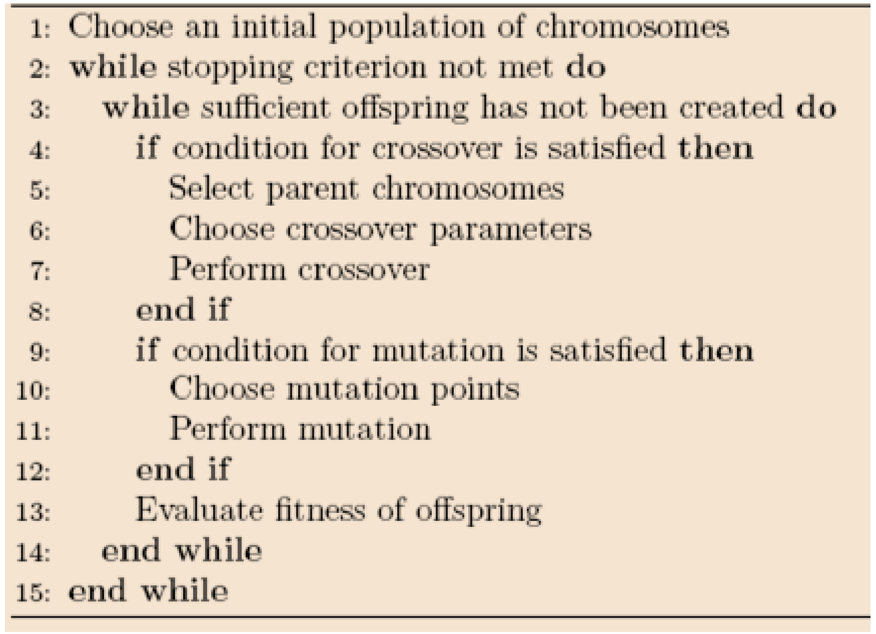


```
function GENETIC-ALGORITHM($population$, FITNESS-FN) returns an individual
   inputs: $population$, a set of individuals
           FITNESS-FN, a function that measures the fitness of an individual

   repeat
      $new\_population \leftarrow$ empty set
      for $i$ = 1 to SIZE($population$) do
         $x \leftarrow$ RANDOM-SELECTION($population$, FITNESS-FN)
         $y \leftarrow$ RANDOM-SELECTION($population$, FITNESS-FN)
         $child \leftarrow$ REPRODUCE($x, y$)
         if (small random probability) then $child \leftarrow$ MUTATE($child$)
         add $child$ to $new\_population$
      $population \leftarrow new\_population$
   until some individual is fit enough, or enough time has elapsed
   return the best individual in $population$, according to FITNESS-FN
```

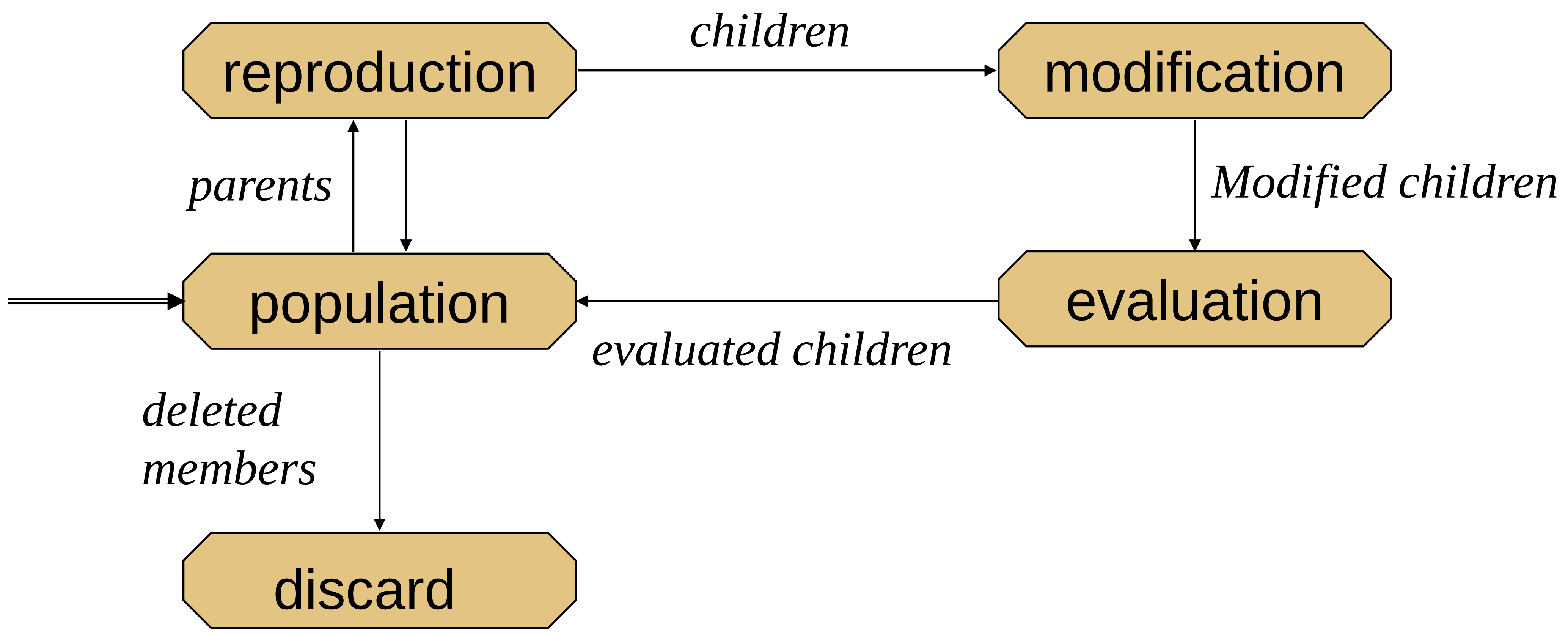
```
function REPRODUCE($x, y$) returns an individual
   inputs: $x, y$, parent individuals

   $n \leftarrow$ LENGTH($x$); $c \leftarrow$ random number from 1 to $n$
   return APPEND(SUBSTRING($x, 1, c$), SUBSTRING($y, c + 1, n$))
```

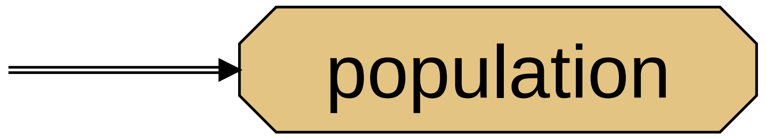


```
Choose an initial population of chromosomes
while stopping criterion not met do
   while sufficient offspring has not been created do
      if condition for crossover is satisfied then
         Select parent chromosomes
         Choose crossover parameters
         Perform crossover
      end if
      if condition for mutation is satisfied then
         Choose mutation points
         Perform mutation
      end if
      Evaluate fitness of offspring
   end while
end while
```

The GA Cycle of Reproduction



Population



Chromosomes could be:

• Bit strings	(0101 ... 1100)
• Real numbers	(43.2 -33.1 ... 0.0 89.2)
• Lists of rules	(R1 R2 R3 ... R22 R23)
• Program elements	(genetic programming)
• ... any data structure ...	

Genetic Algorithms

- The overall genetic algorithm can be defined by the simple process shown in Figure 7.7.
- First, a pool of random potential solutions is created that should have adequate variety.
- Next, the fitness of each member is computed.
- Next, members of the population are selected based on some algorithm. The two simplest approaches are *roulette wheel* selection, and *elitist* selection (see Figure 7.8).

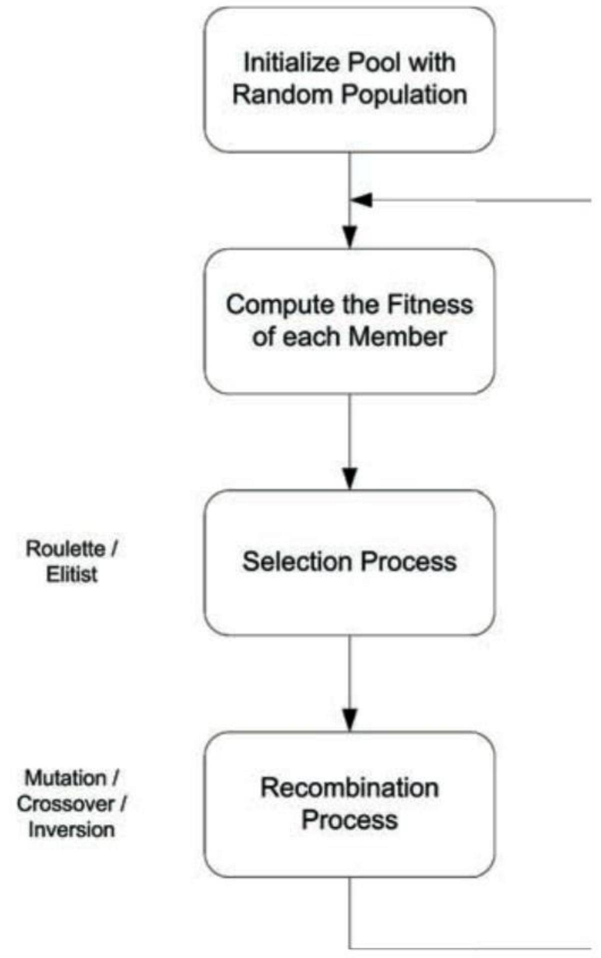
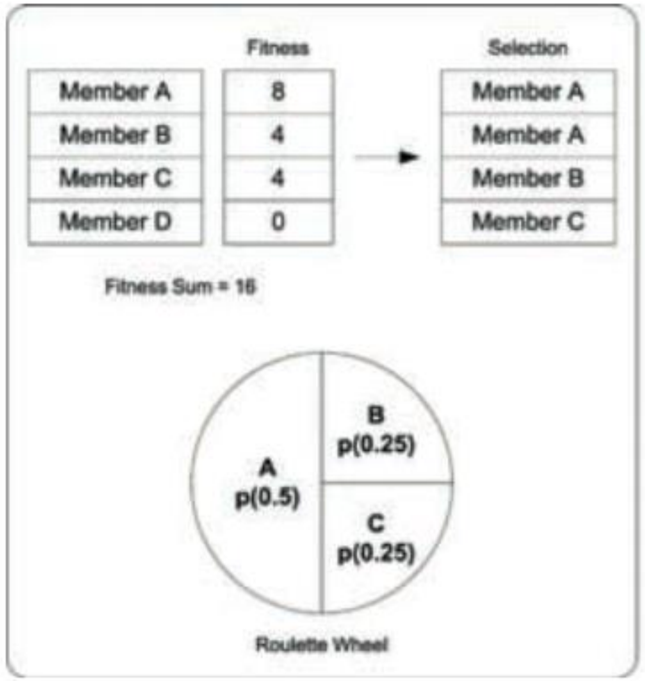

Initialize Pool with Random Population
Compute the Fitness of each Member
Roulette / Elitist
Selection Process
Mutation / Crossover / Inversion
Recombination Process

FIGURE 7.7: **Simple flow of the genetic algorithm.**

Selection Process: Wheel vs. elitist selection

From the selection process, we have several members that have the right to propagate their genetic material to the next population.



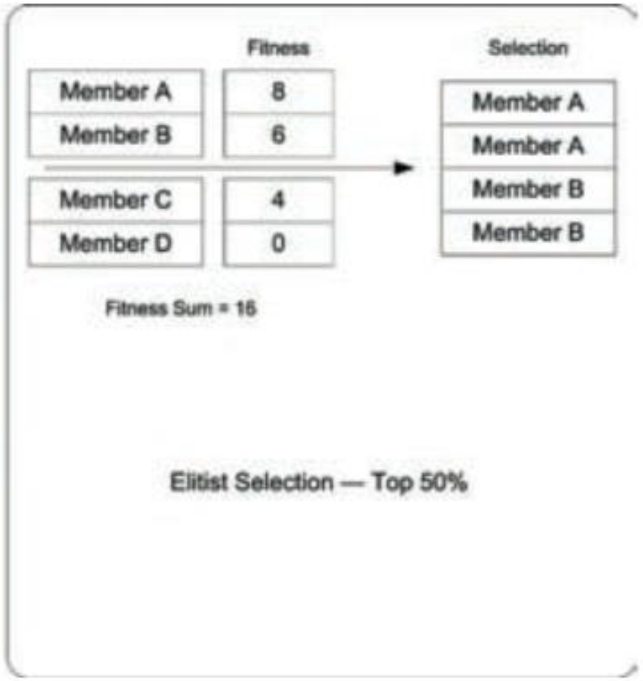
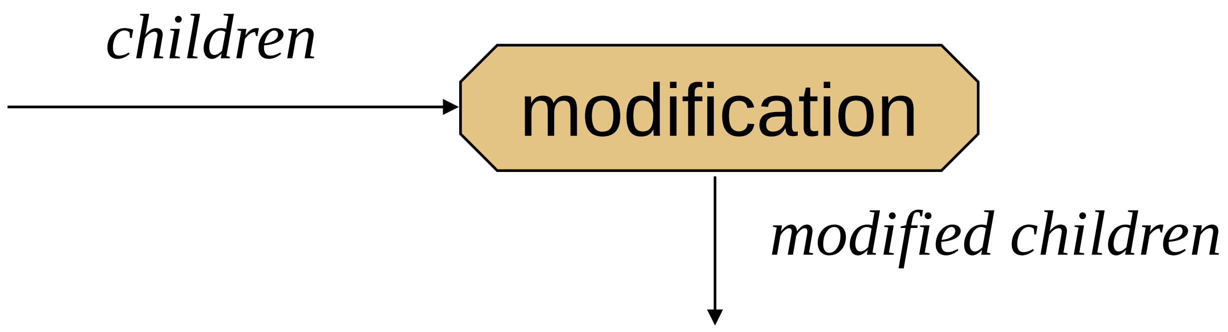


FIGURE 7.8: Two of the simpler GA selection models.

Chromosome Modification



- Modifications are stochastically (randomly) triggered
- Operator types are:
    - Mutation
    - Crossover (recombination)

Recombination Process: Crossover Operators

- The next step is to recombine these members’ material to form the members of the next generation.
- Commonly, parents are selected two at a time from the set of individuals that are permitted to propagate (from the selection process).
- Given two parents, two children are created in the new generation with slight modifications of the recombination process (with a given probability that the genetic operator can occur).

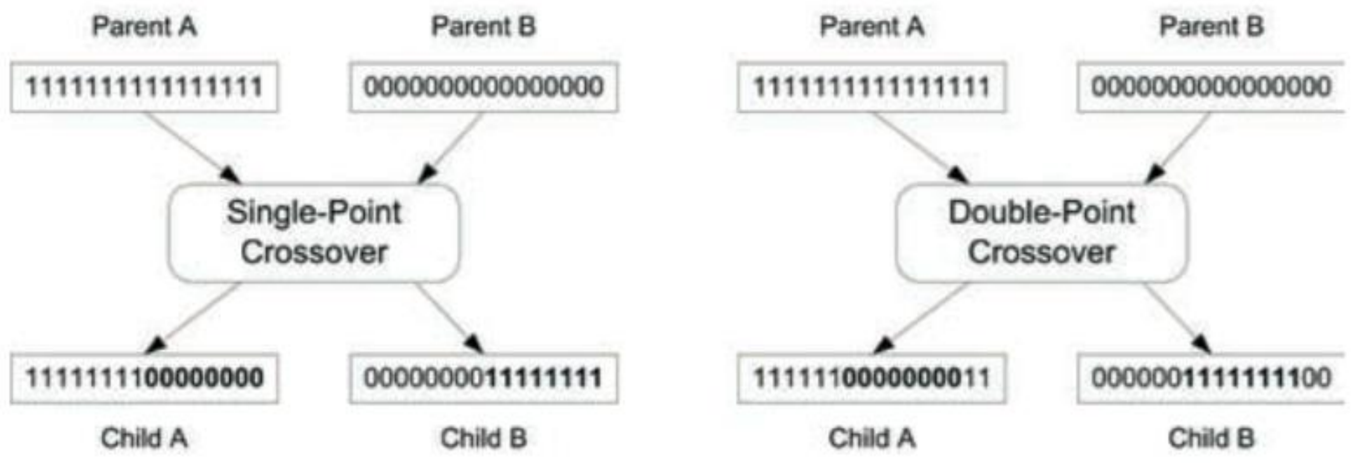


FIGURE 7.9: **Illustrating the crossover operators in genetic recombination.**

Recombination Process: Crossover Operators

P1 (0 1 | 1 0 1 0 0 0) $\rightarrow$ (0 1 | 1 1 1 0 1 0) C1

P2 (1 1 | 0 1 1 0 1 0) (1 1 | 0 0 1 0 0 0) C2

Crossover is a critical feature of genetic algorithms:

- It greatly accelerates search early in evolution of a population
- One parent is selected based on *fitness*
- The other parent is selected *randomly* until (at least) M offspring are created
- Random choice of cross-over point

Recombination Process: Crossover Operators

Crossover combines *inversion* and recombination:

Parent1	(8 5	7 2 1 6	4 8)
Parent2	(2 5	7 6 8 1	3 4)
Child	(5 8	7 2 1 6	3 4)

This operator is called order-based crossover.

Mutation: Local Modification

Before: (1 0 1 $\boxed{*}$ 0 1 1 0)

After: (1 0 1 $\boxed{0}$ 0 1 1 0)

Before: (1.38 $\boxed{-69.4}$ 326.44 0.1)

After: (1.38 $\boxed{-67.5}$ 326.44 0.1)

- Causes movement in the search space
- Restores lost information to the population

Mutation

Mutation involves reordering of the list:

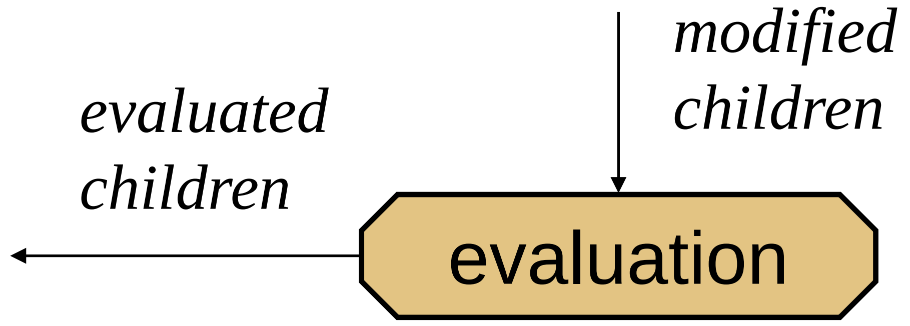
Before: (5 8 7 2 1 6 3 4)

After: (5 8 6 2 1 7 3 4)

Evaluation of Individuals

For every step in the evolution

- Selection of individuals for genetic operations
- Creation of new individuals (reproduction)
- Mutation
- Selection of individuals to survive

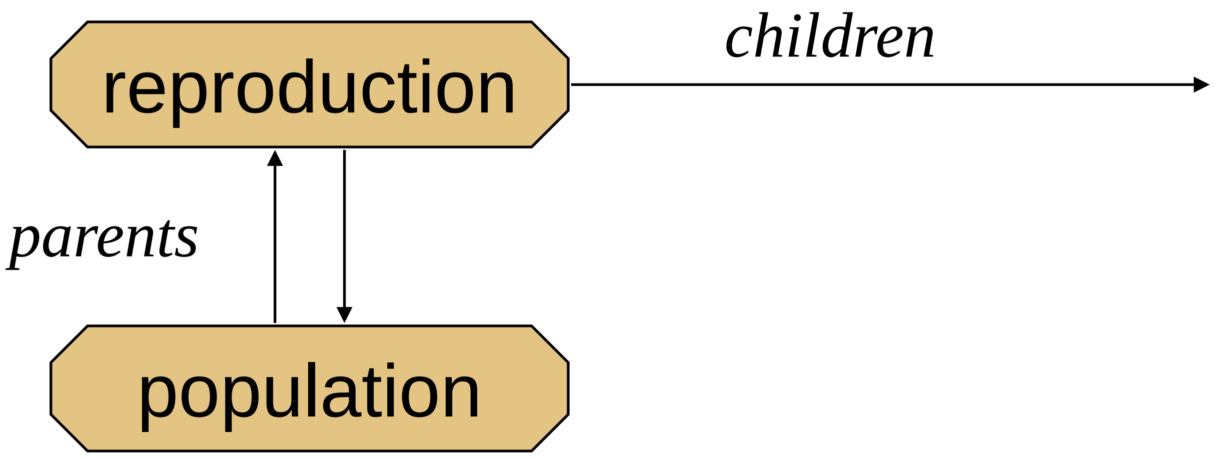


The evaluator decodes a chromosome and assigns it a fitness measure

- Adaptability – ”fitness”
- Fitness is maximized
- Used in selection (”Survival of the fittest”)
- Often normalized

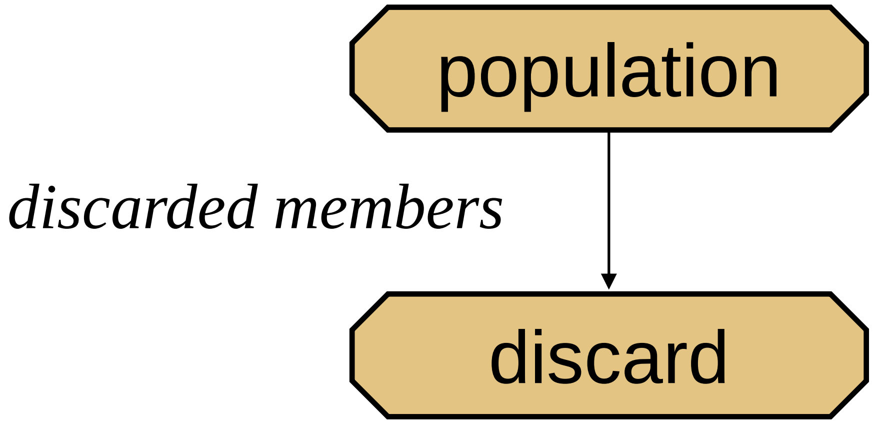
$$f : S \rightarrow [0,1]$$

Reproduction



Parents are selected at random with selection chances biased in relation to chromosome evaluations

Deletion



- **Generational GA:** *entire populations replaced each iteration*
- **Steady-state GA:** *a few members replaced each generation*

Termination Criterion

How it terminates?

- There are a number of ways that we can terminate the process.
- The most obvious is to end when a solution is found, or one that meets the designer’s criteria. But from the algorithm’s perspective, we also need to account for the population, and its ability to find a solution.

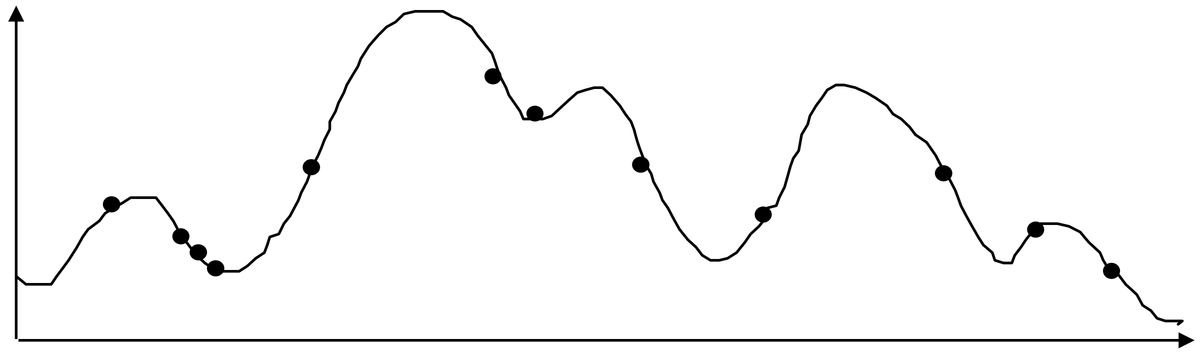
Another termination criterion

- Potentially returning a suboptimal solution when the population lacks diversity.
- Inability to adequately search the solution space.
- When the members of the population become similar, there’s a loss in the ability to search. To prevent this, we terminate the algorithm early by detecting if the average fitness of the population is near the maximum fitness of any member of the population.

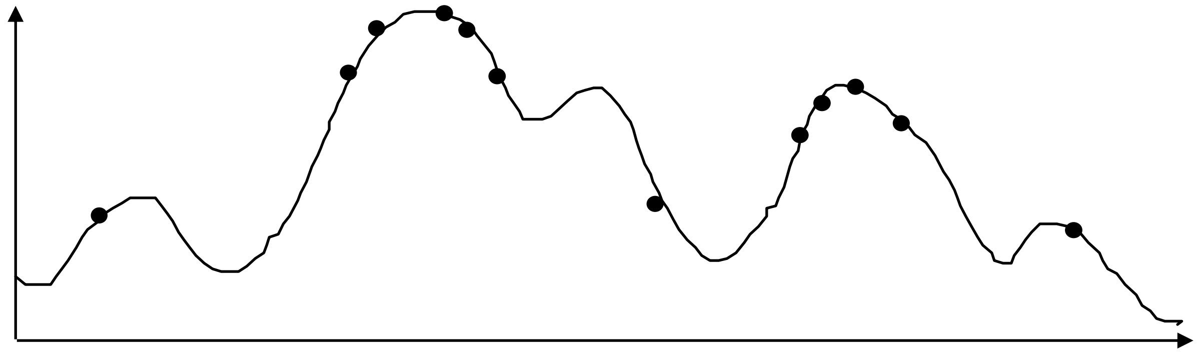
GA – Standard Reproduction Plan

- **Population size**
  - Small populations: undercoverage
  - Large population: computationally demanding
  - Optimal size increases exponentielly with the string-length in binary encodings
- **Standard cross-over**
  - One parent selected according to fitness
  - The other selected randomly
  - Random cross-over point
- **Mutation**
  - A certain probability that an individual mutate
  - Random choice of which gene to mutate

An abstract Example



Distribution of Individuals in Generation 0



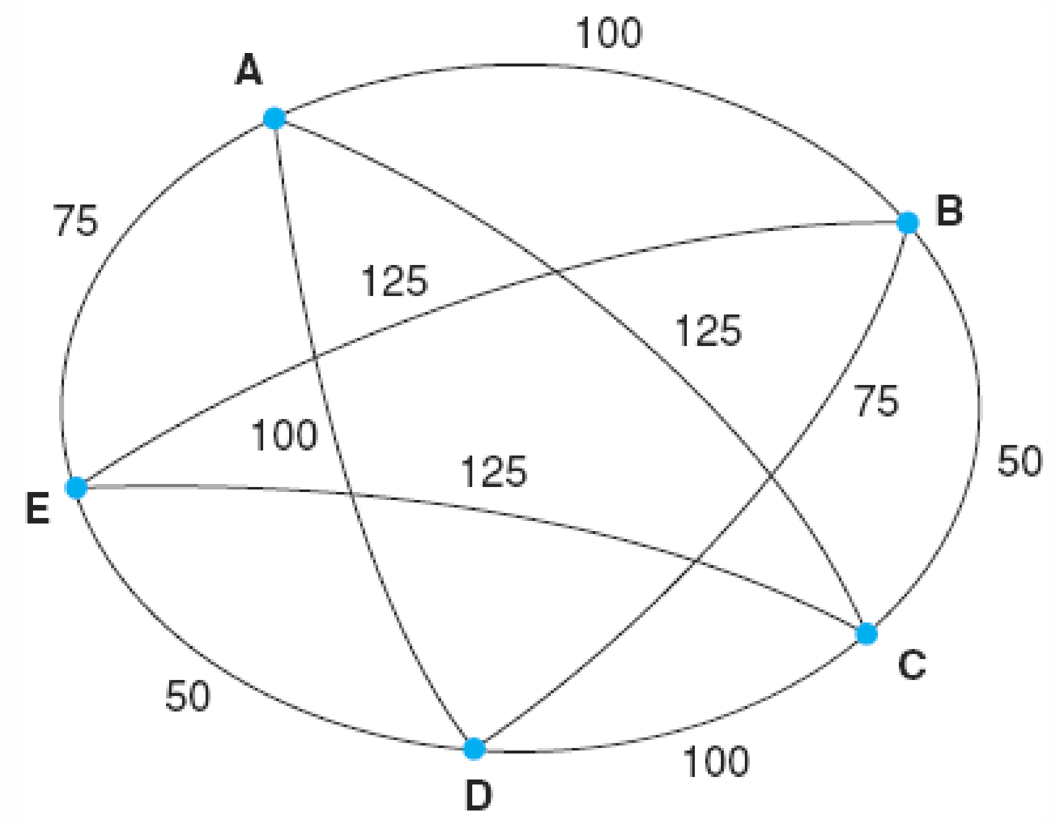
Distribution of Individuals in Generation N

TSP Problem: example

The Traveling Salesman Problem

Find a tour of a given set of cities so that

- each city is visited only once
- the total distance traveled is minimized



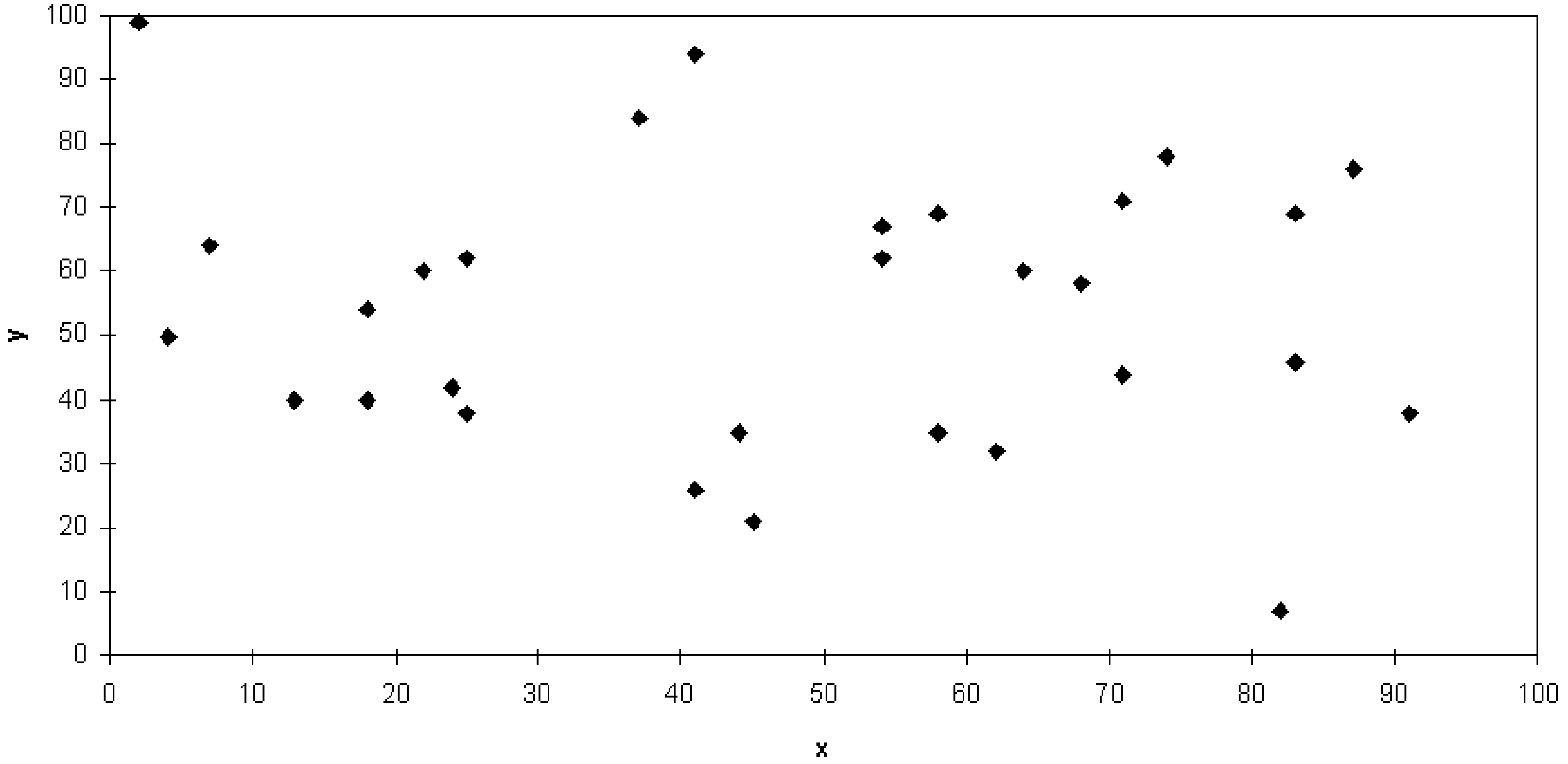
Representation: example

Representation is an ordered list of city numbers known as an *order-based* GA.

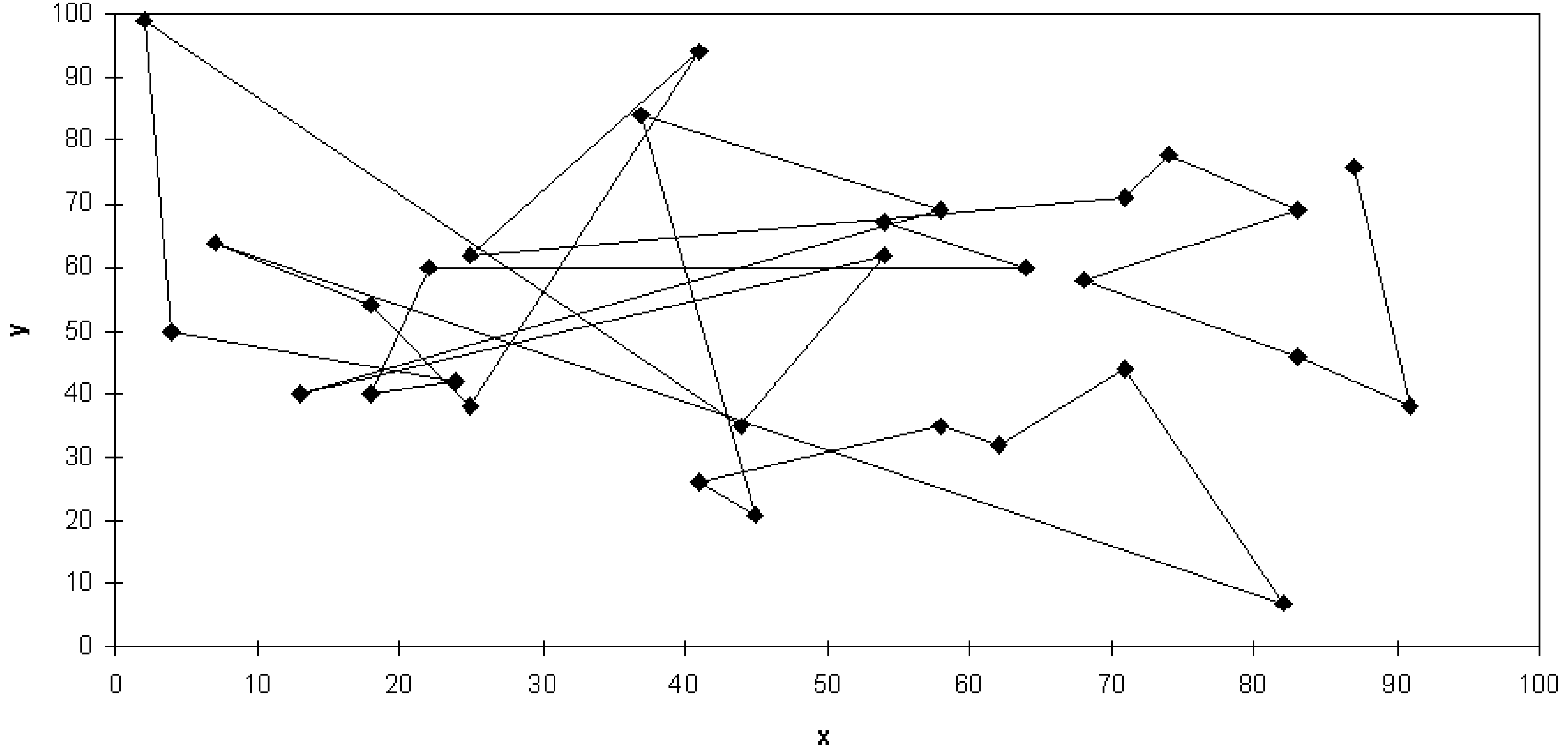
1) Madinah	3) Makkah	5) Yanbu	7) Tabuk
2) Jeddah	4) Riyadh	6) Damam	8) Taif

City List 1	(3 5 7 2 1 6 4 8)
City List 2	(2 5 7 6 8 1 3 4)

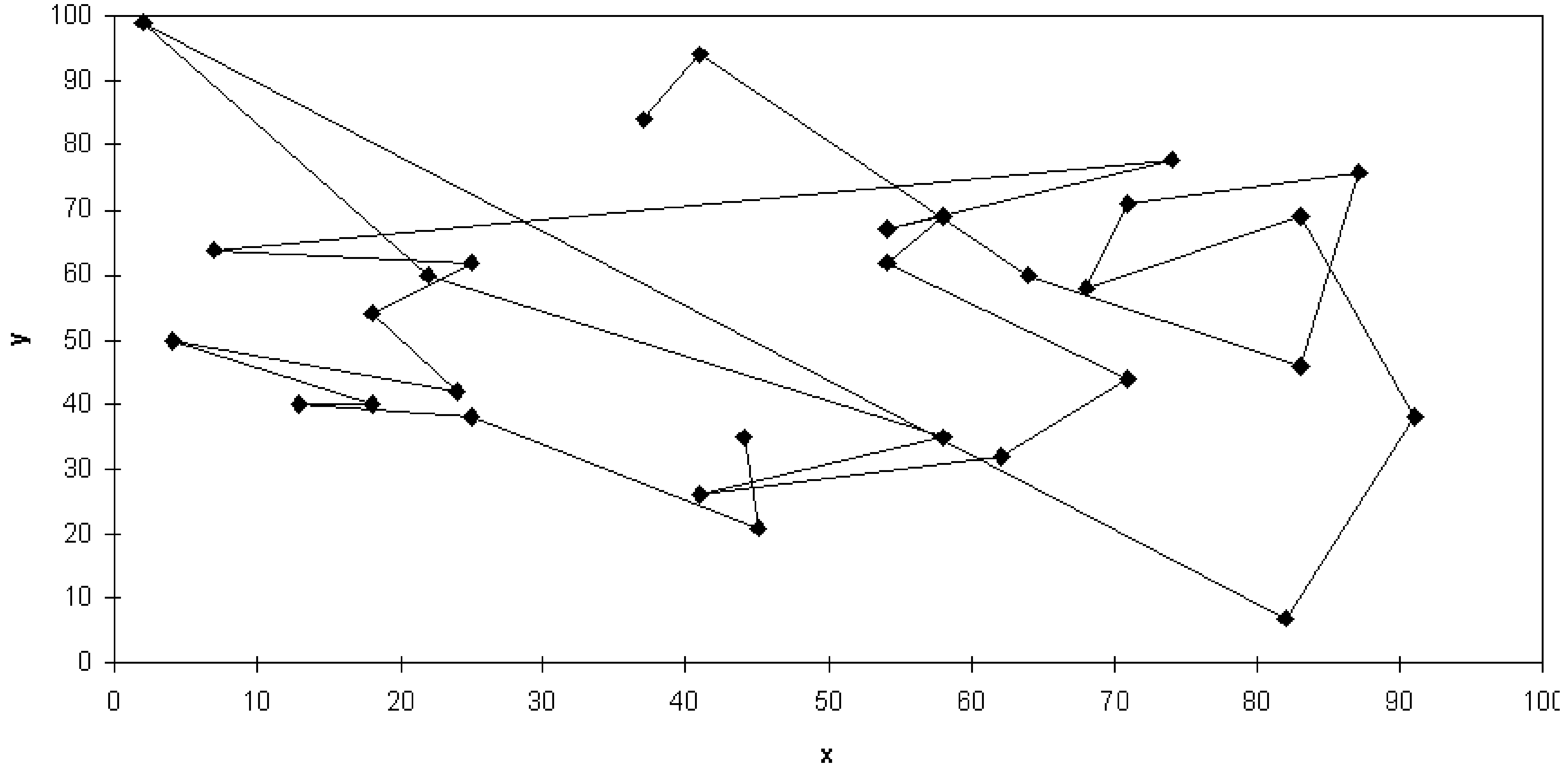
TSP Example: 30 Cities



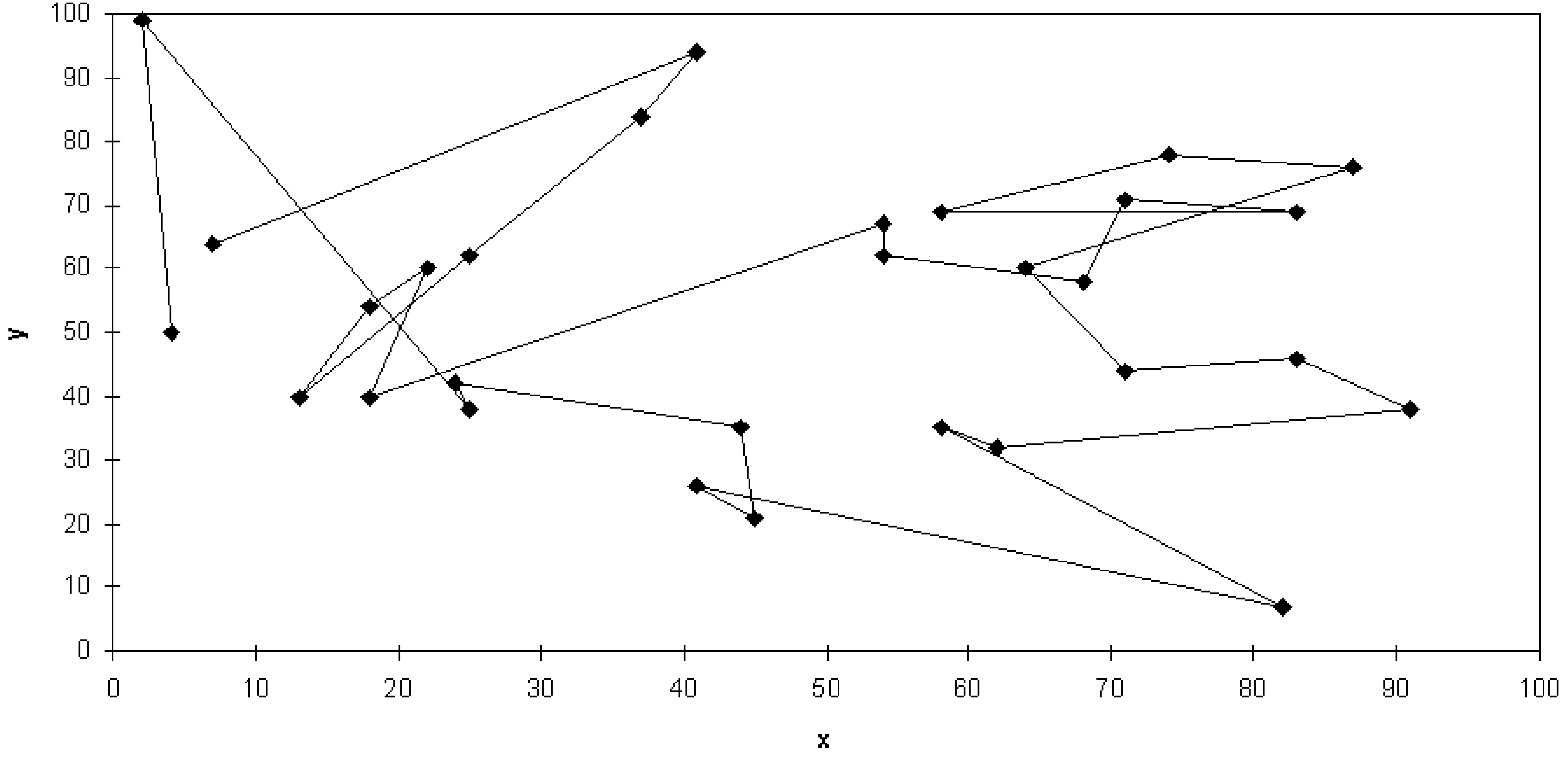
Solution *i* (Distance = 941)



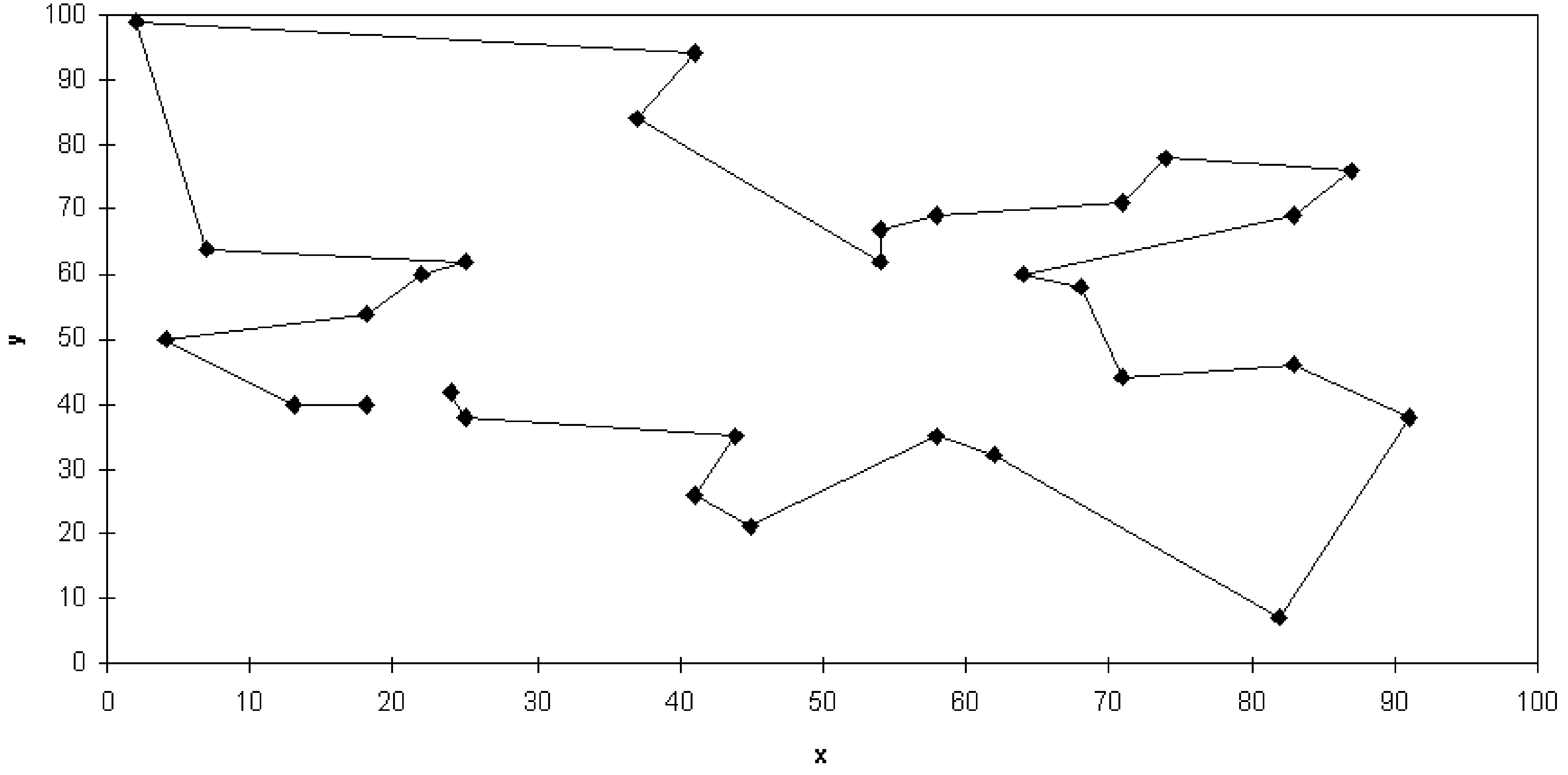
Solution j (Distance = 800)



Solution *k* (Distance = 652)



Best Solution (Distance = 420)



Issues for GA

- **Basic implementation issues:**
  - Representation
  - Population size, mutation rate, ...
  - Selection, deletion policies
  - Crossover, mutation operators
- **Termination Criteria**
- **Performance, scalability**
- **Solution** is only as good as the evaluation function (often hardest part)

- Mutation
  - Choice of mutation rate not critical
- Crossover – often effective
  - Selective choice of crossover point
- N-point crossover
  - 2-points crossover has given better performance
  - 8-points crossover has given best results

Contents for the next lecture

- Constraint Satisfaction Problems
- Games as Search

Any questions?