08 Introduction to Genetic Algorithm

Sang Jin Kweon (권상진)

School of Management Engineering
UNIST

Office hours: after class (or by APPT.)
Email: sjkweon@unist.ac.kr
Tel: +82 (52) 217-3146
Web: http://or.unist.ac.kr

TA
Seungok Woo (우승옥)
- Email: wso1017@unist.ac.kr
- Office Hours: by APPT.

Acknowledgement

본 캠프는 한동대학교 공학교육혁신센터의
SW 단기교육과정 개발사업비로 개최되었습니다.

Hicee.handong.edu

Heuristic and Metaheuristic

❑ Heuristic

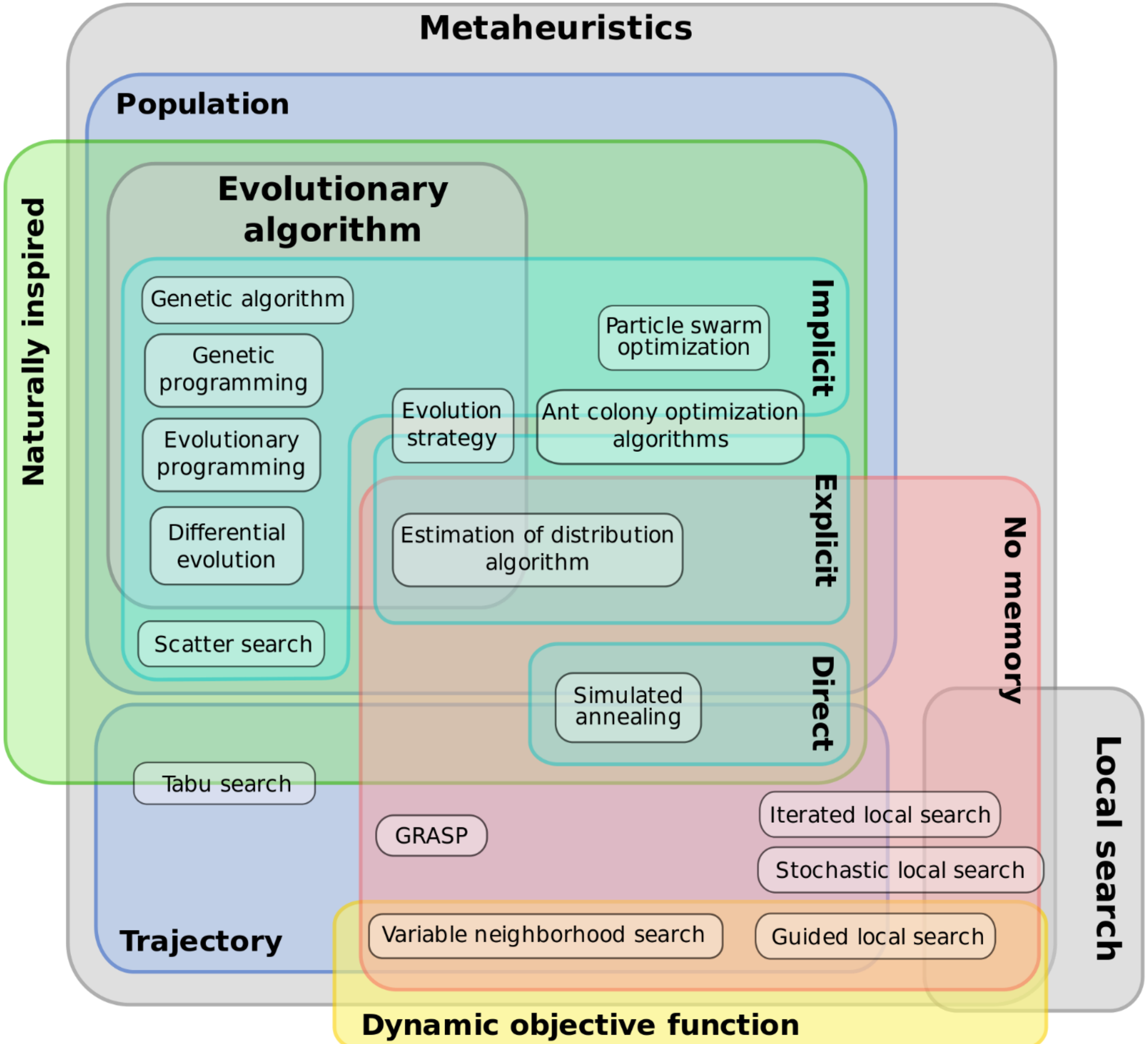
- Is any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect or rational, but which is nevertheless sufficient for reaching an immediate, short-term goal.
- Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution. Heuristics can be mental shortcuts that ease the cognitive load of making a decision.
- Examples that employ heuristics include using trial and error, a rule of thumb or an educated guess.

❑ Metaheuristic

- Is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity.
- Does not guarantee that a globally optimal solution can be found on some problems.
- But, by searching over a large set of feasible solutions in combinatorial optimization, metaheuristics can often find good solutions with less computational effort than optimization algorithms, iterative methods, or simple heuristics.
- As such, metaheuristics are useful approaches for optimization problems.
- Well-known metaheuristics including but not limited to: Genetic Algorithm (GA), Tabu Search (TS), Simulated Annealing (SA).

Classifications of Metaheuristics

- Euler diagram of the different classifications of metaheuristics



Classifications of Search Methods

- Deterministic vs. Stochastic

Search Methods

Deterministic

Numerical Methods

Graph Search Methods

Blind Search

- Depth-first
- Breadth-first
- Dijkstra

Informed Search

- Hill climbing
- Beam search
- Best-first, A*

Stochastic

Trajectory Methods

- TS
- SA

Population-based Methods

Evolutionary Computing

- **GA**

Swarm Intelligence

- PSO
- ACO

What is a Genetic Algorithm (GA)?

- Evolution in biology
- Algorithm
- Pros and cons
- Applications
- Examples
  - The Delivery Scheduling Problem
  - The Traveling Salesman Problem

Evolution in Biology

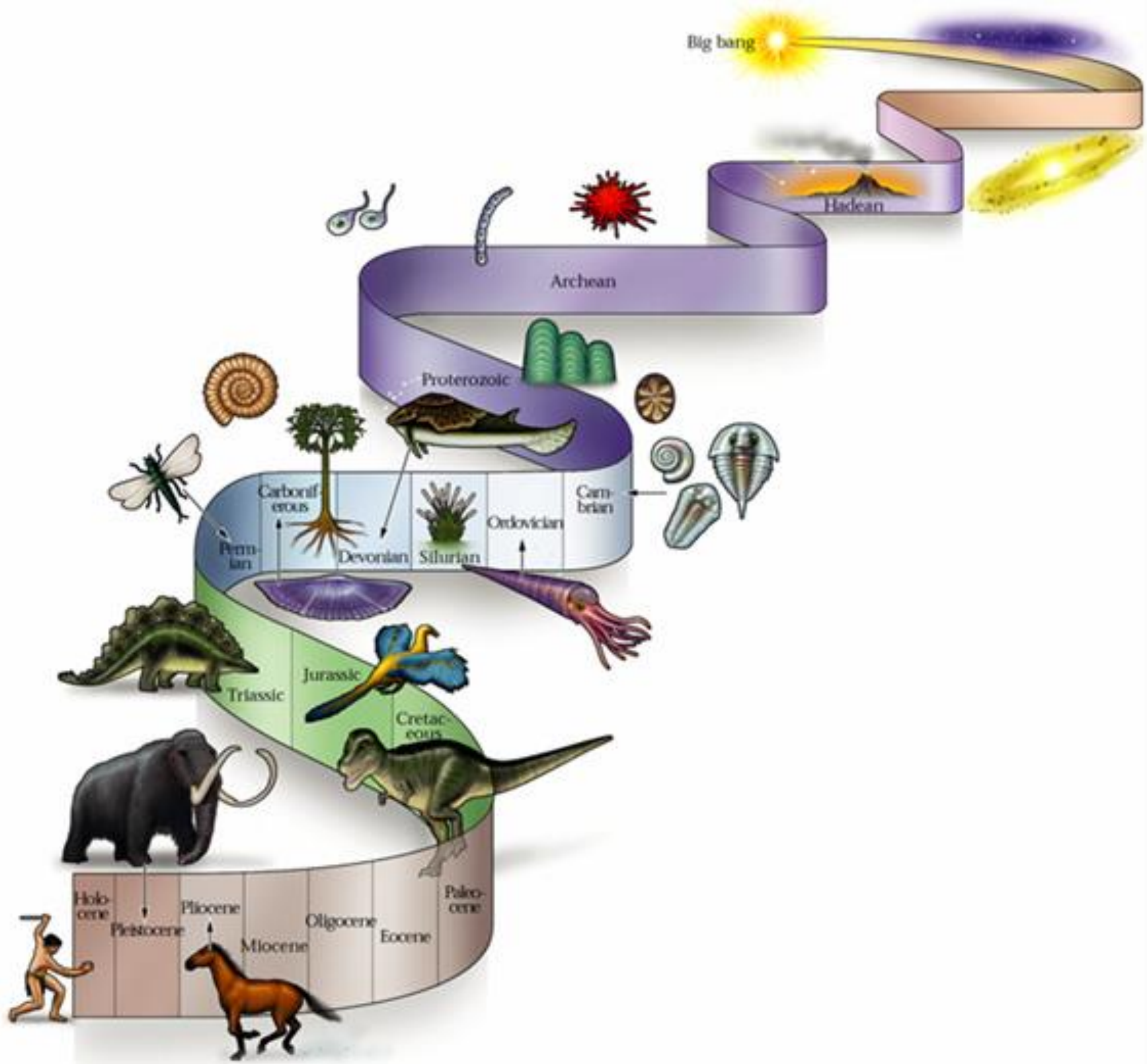
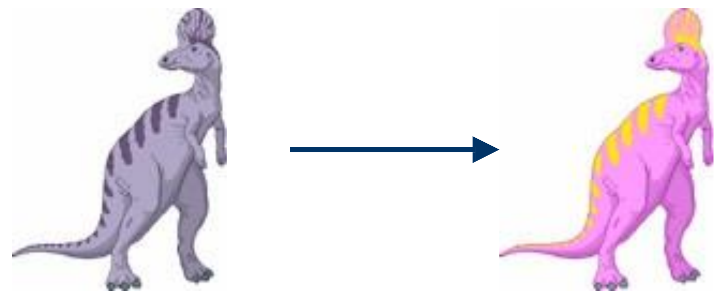


Image from http://www.geo.au.dk/besoegsservice/foredrag/evolution

Evolution in Biology I

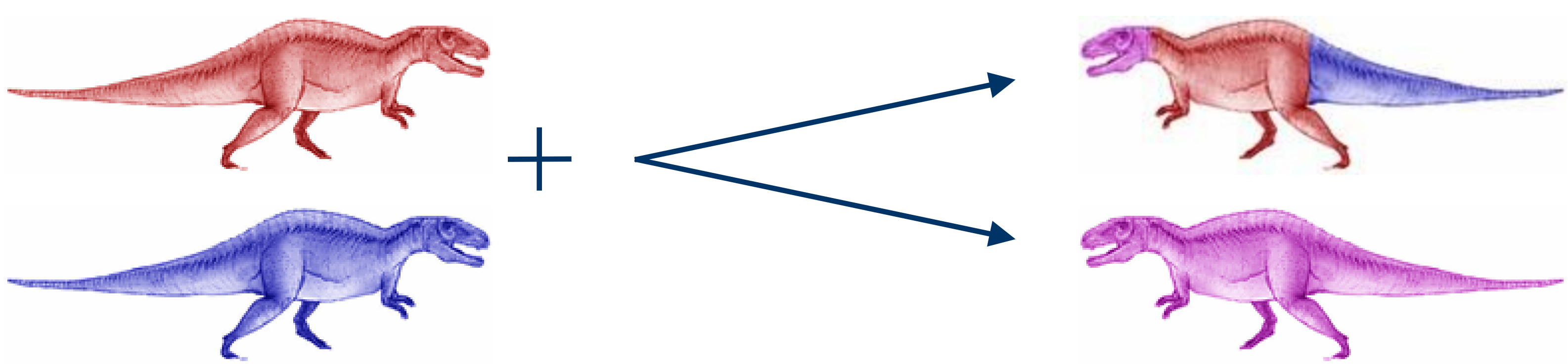
❑ Mutations

- Organisms produce a number of offspring similar to themselves but can have variations due to:
  - Mutations (random changes)



❑ Crossovers

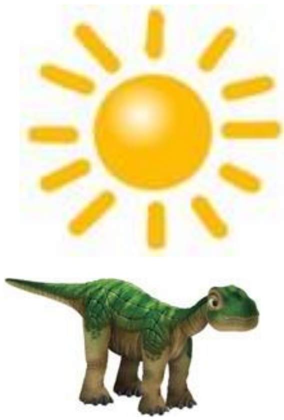
  - Sexual reproduction (offspring have combinations of features inherited from each parent)



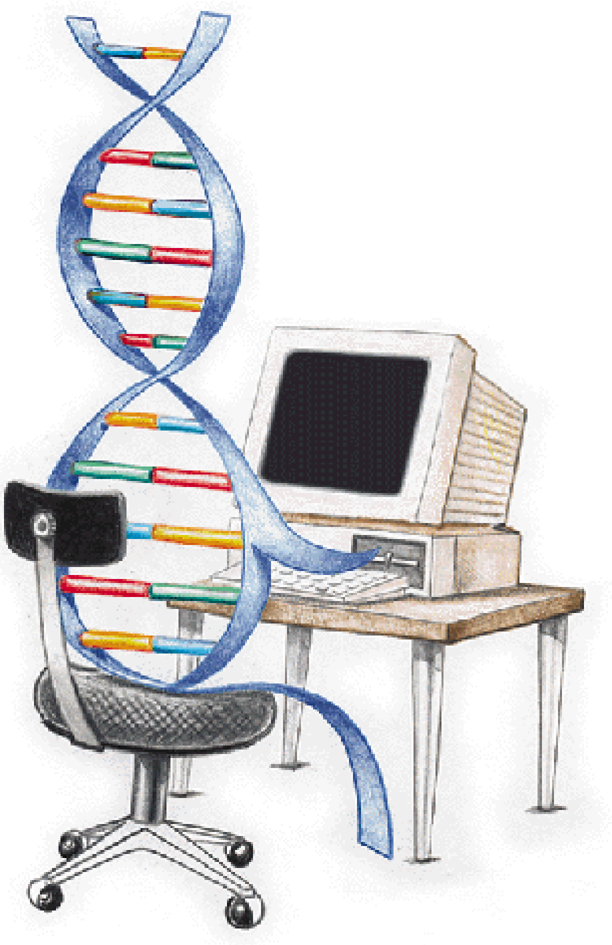
Evolution in Biology II

❑ Selections (Fitness)

- Some offspring survive, and produce next generations, and some don't:
  - The organisms adapted to the environment better have higher chance to survive
  - Over time, the generations become more and more adapted because the fittest organisms survive



The Genetic Algorithms



The Genetic Algorithms (GA)

- Based on the mechanics of biological evolution
- Initially developed by John Holland, University of Michigan (1970's)
  - To understand processes in natural systems
  - To design artificial systems retaining the robustness and adaptation properties of natural systems

John Henry Holland (1929 – 2015)
Professor
Electrical engineering and computer science
The University of Michigan, Ann Arbor

- Holland's original GA is known as the simple genetic algorithm (SGA)
- Provide efficient techniques for optimization and machine learning applications
- Widely used in business, science and engineering

Genetic Algorithms Techniques

- GAs are a particular class of evolutionary algorithms.
  - The techniques common to all GAs are:
    - Inheritance
    - Mutation
    - Selection
    - Crossover (also called recombination)
  - GAs are best used when the objective function is:
    - Discontinuous
    - Highly nonlinear
    - Stochastic
    - Has unreliable or undefined derivatives

Performance of Genetic Algorithms

- GAs can provide solutions for highly complex search spaces
- GAs perform well approximating solutions to all types of problems because they do not make any assumption about the underlying fitness landscape (the shape of the fitness function, or objective function)
- However, GAs can be outperformed by more field-specific algorithms

Biological Terminology of Genetic Algorithms

- Gene – a single encoding of part of the solution space, i.e. either single bits or short blocks of adjacent bits that encode an element of the candidate solution

1

- Chromosome – a string of genes that represents a solution

0	1	0	1	1

- Population – the number of chromosomes available to test

0	1	0	1	1
1	1	1	1	1

1	1	0	1	1
1	0	0	1	1

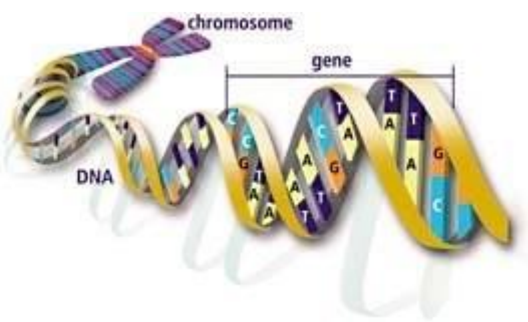
1	1	0	0	1
0	1	0	1	0

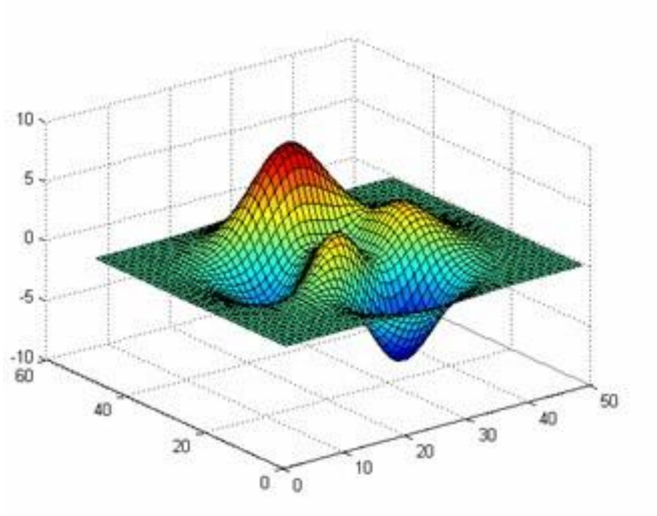
0	1	0	0	0
1	1	0	1	0

Biology vs. Optimization

- Candidate solutions to the optimization problem play the role of individuals in a population (or chromosomes)

- Cost/fitness/objective function determines the environment within which the solutions "live"





Features of Genetic Algorithms

- Not too fast but cover large search space
  - Capable of quickly finding promising regions of the search space but may take a relatively long time to reach the optimal solution. Solution: hybrid algorithms

- Good heuristics for combinatorial problems

- Usually emphasize combining information from good parents (crossover)

- Different GAs use different
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

The Basic Genetic Algorithm

❑ Pseudocode for the basic GA

Initialize population with random candidates,

Evaluate all individuals,

While termination criteria is not met [Fitness]

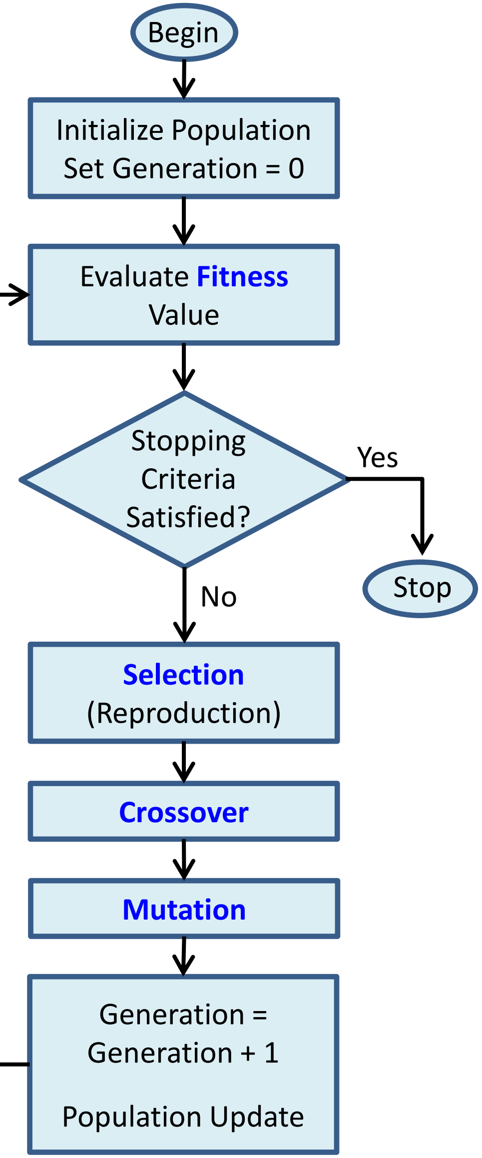
  Select parents, [Selection]

  Apply crossover, [Crossover]

  Mutate offspring, [Mutation]

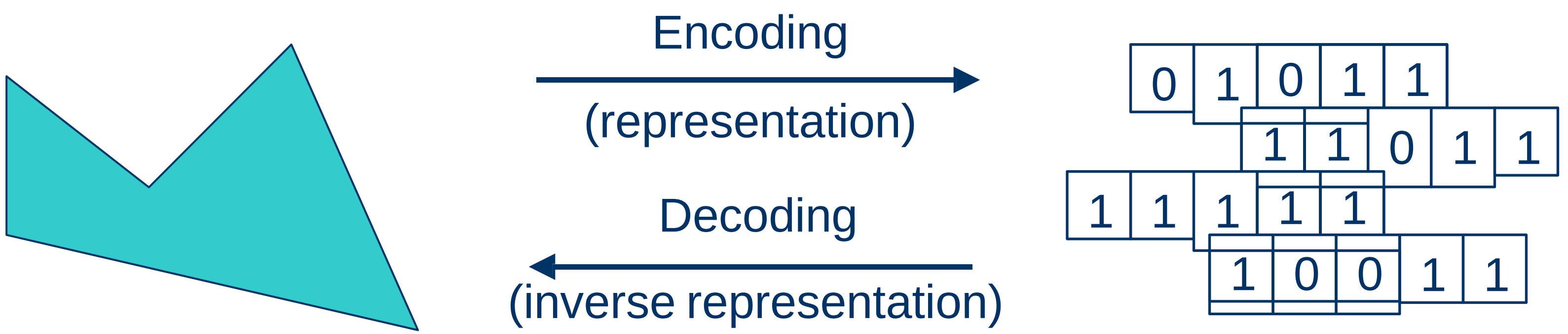
  Replace current generation,

end while



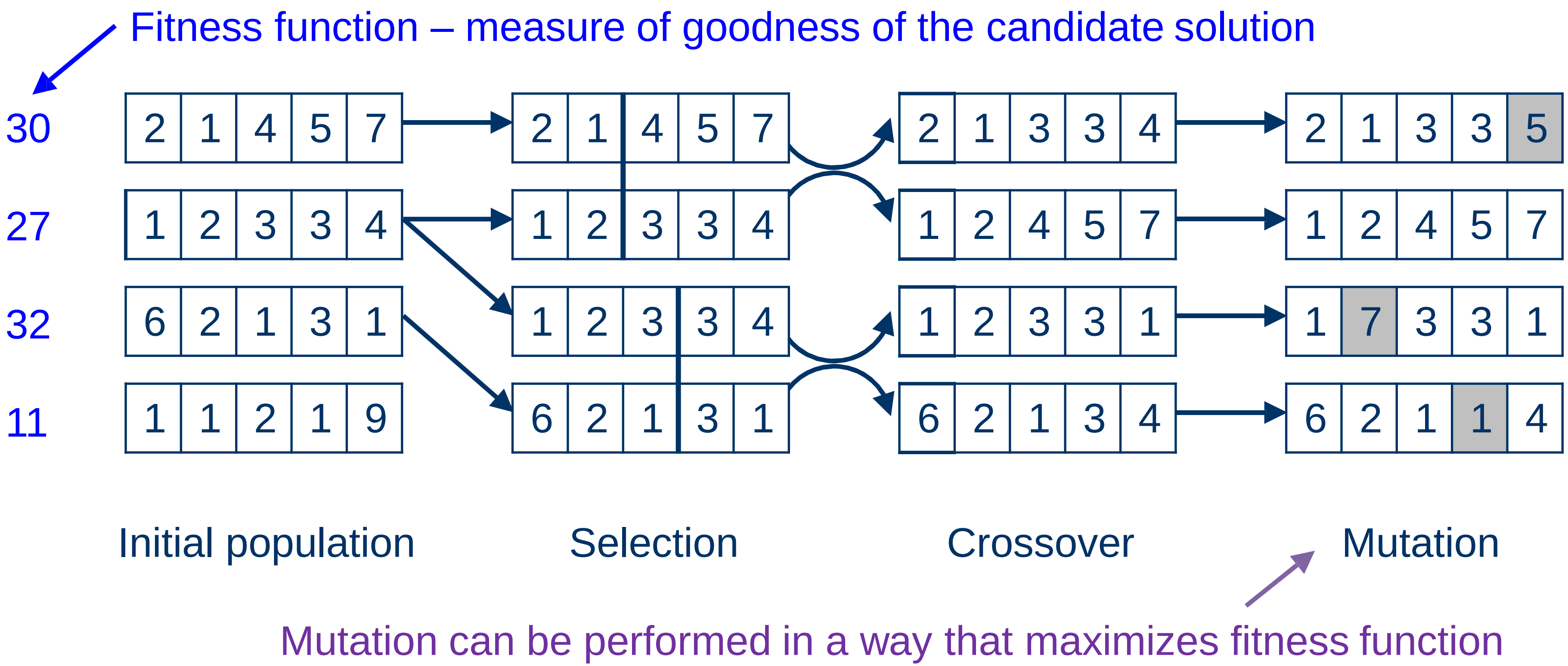
Representation

- Chromosomes can be:
  - Bit strings (0110, 0011, 1101, …)
  - Real numbers (33.2, -12.11, 5.32, …)
  - Permutations of element (1234, 3241, 4312, …)
  - Lists of rules (R1, R2, R3, … Rn …)
  - Program elements (genetic programming)
  - Any other data structure



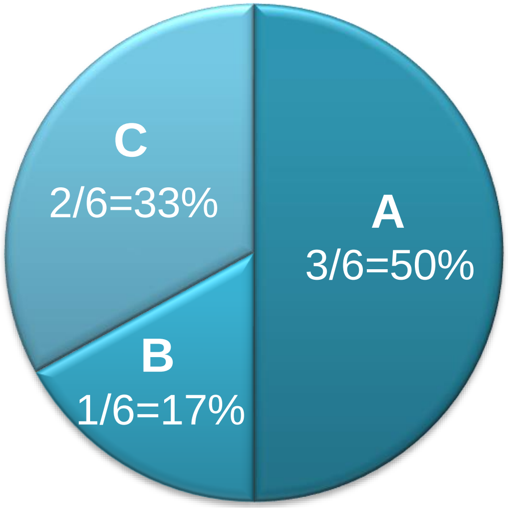
Selection

- Parents with better fitness have better chances to produce offspring



An Example of Selection: Roulette Wheel

- The most commonly used selection methods include: Roulette Wheel Selection, Elitist Selection, cutoff Selection, Rank Selection, Tournament Selection, and Boltzmann Selection.
- Better solutions get higher chance to become parents for next generation solutions.
- Roulette wheel technique:
  - Assign each individual part of the wheel
  - Spin wheel N times to select N individuals



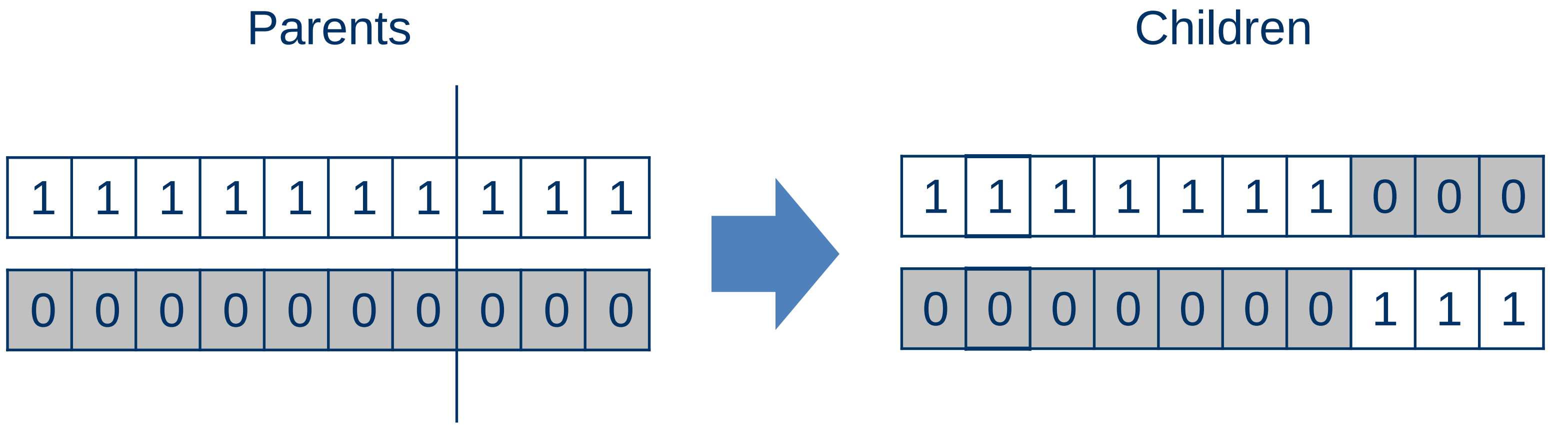
fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

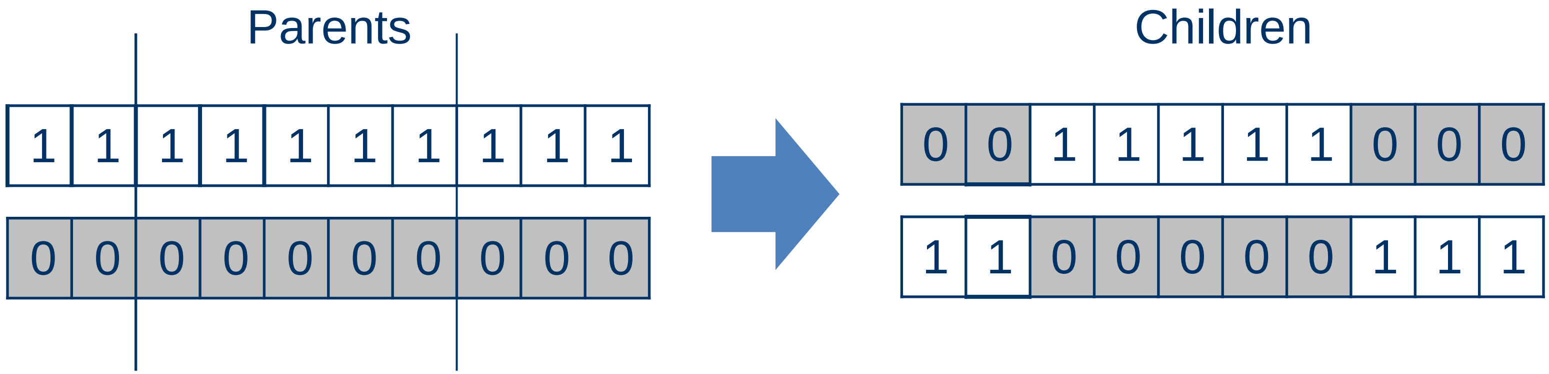
1-Point Crossover

- Choose a random point
- Split parents at this crossover point
- Create children by exchanging tails
- Probability of crossover is typically in range (0.6, 0.9)

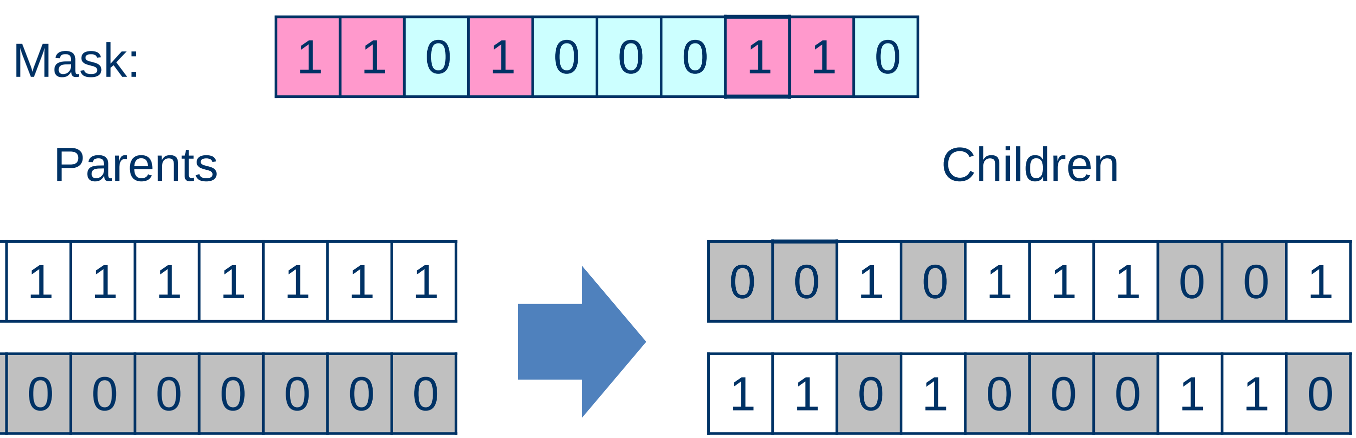


Other Crossover Types

- Two-point crossover



- Uniform crossover: randomly generated mask



Mutation

- Alter each gene independently
- Mutation probability is typically in range (1/population_size, 1/chromosome_length)

Parent

1	1	1	1	1	1	1	1	1	1

Child

1	1	0	1	1	1	1	0	1	0

- Choose mutation with the best fit

Termination (Stopping Criteria)

- This generational process is repeated until a termination condition has been reached.
- Common terminating conditions are:
  - A solution is found that satisfies minimum criteria
  - Fixed number of generations reached
  - Allocated budget (computation time/money) reached
  - The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
  - Manual inspection
  - Combinations of the above

Benefits of Genetic Algorithms

- Concept is easy to understand
- Modular – separate from application (representation); building blocks can be used in hybrid applications
- Supports multi-objective optimization
- Good for "noisy" environment
- Always results in an answer, which becomes better and better with time
- Can easily run in parallel
- The fitness function can be changed from iteration to iteration, which allows incorporating new data in the model if it becomes available

Issues with Genetic Algorithms

- Choosing parameters:
  - Population size
  - Crossover and mutation probabilities
  - Selection, deletion policies
  - Crossover, mutation operators, etc.
  - Termination criteria
- Performance:
  - Can be too slow but covers a large search space
  - Is only as good as the fitness function

Applications of Genetic Algorithms

- **Optimization** – numerical and combinatorial optimization problems, e.g. traveling salesman, routing, graph colouring and partitioning
- **Robotics** – trajectory planning
- **Machine learning** – designing neural networks, classification and prediction, e.g. prediction of weather or protein structure
- **Signal processing** – filter design
- **Design** – semiconductor layout, aircraft design, communication networks
- **Automatic programming** – evolve computer programs for specific tasks, design cellular automata and sorting networks
- **Economics** – development of bidding strategies, emergence of economics markets
- **Immune systems** – model somatic mutations
- **Ecology** – model symbiosis, resource flow
- **Population genetics** – “Under what condition will a gene for recombination be evolutionarily viable?”

Genetic Algorithm Example 1:
The Delivery Scheduling Problem

Problem Statement

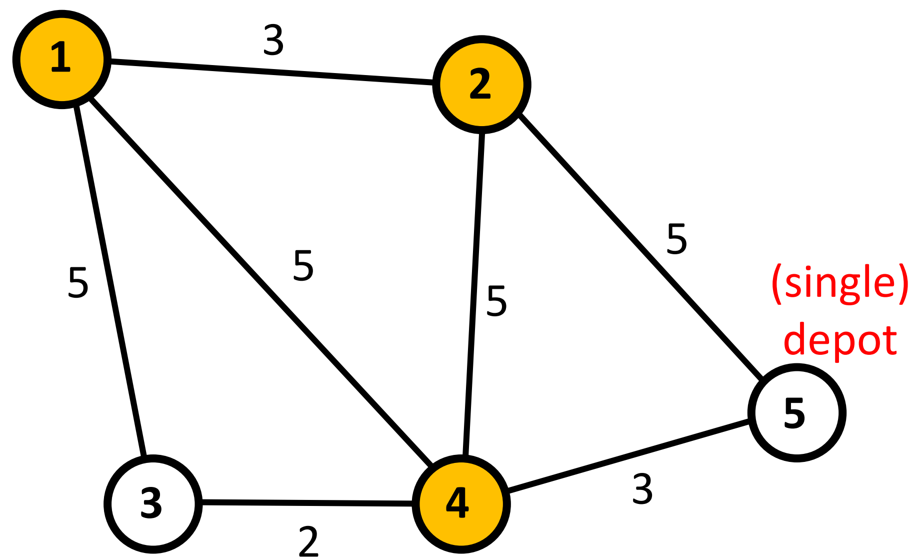
- For a given problem setting below, the gas delivery schedule to gas stations by trucks should be determined for finite time period $t \in T=\{1, 2, 3, 4\}$, **so as to minimize the total cost** (= inventory holding costs at stations + delivery costs by trucks) while no shortage is allowed.

usually trade-off relationship b/t them

- There is a single depot for gas deliveries at Node 5.
- Gas stations open at Nodes 1, 2, and 4.
- Every station has a capacity of 300 drums.
- At the beginning of time period, every station has its full capacity of 300 drums.
- No shortages are allowed (i.e., gas must be replenished to stations before shortage).

- Time windows for gas demand (in drums) at stations are given, as below.

	$t=1$	$t=2$	$t=3$	$t=4$
Station 1	95	105	55	50
Station 2	120	148	134	76
Station 4	115	136	166	91



Simplified Version of the MILP Formulation

- Notations

parameters

h_i^t = inventory holding cost at Station i at the end of time t

d_{ij}^t = delivery cost Depot i (Depot 5) to Station j in time t;

variables

v_i^t = amount of gas remained at Station i at the end of time t;

$$x_{i,j}^{q,t} = \begin{cases} 1 & \text{, if truck } q \text{ delivers gas from Depot } i \text{ (Depot 5) to Station } j \text{ at the end of time } t; \\ 0 & \text{, otherwise.} \end{cases}$$

- This problem can be formulated as a mixed integer linear programming (MILP) problem.

Part of MILP Formulation (simplified version for lecture)

$$\text{Min.} \sum_{i \in N} \sum_{t \in T} h_i^t v_i^t + \sum_{i,j \in N_0} \sum_{q \in Q} \sum_{t \in T} d_{ij}^t x_{ij}^{qt}$$

$$\begin{aligned} \text{s.t.} \quad & v_i^t \le 300, && \forall i \in N, \forall t \in T \\ & v_i^t \ge 0, && \forall i \in N, \forall t \in T \\ & x_{i,j}^{q,t} \in \{0,1\}, && \forall i,j \in N, \forall t \in T, \forall q \in Q \end{aligned}$$

$1 per item per time at i — **Inventory holding cost**

$100 × Distance b/t i and j — **Delivery cost**

Original Version of the MILP Formulation

$$\min \sum_{i \in N'} \sum_{t \in T} h_i^t v_i^t + \sum_{i,j \in N_0} \sum_{q \in Q} \sum_{t \in T} d_{ij} x_{ij}^{qt}$$

s.t.

$$\sum_{i \in K_{jk}^p} z_i \geq z^p \qquad \forall a_{ij} \in A_p, \forall p \in P$$

$$\sum_{p \in P} f^{pt} z^p \geq F \sum_{p \in P} f^{pt} \qquad \forall t \in T$$

$$z_i \geq \sum_{h \in H_p} z_i^{ph} \qquad \forall i \in N, \forall p \in P$$

$$\sum_{h \in H_p} z^{ph} = z^p \qquad \forall p \in P, \forall t \in T$$

$$z_i^{ph} = z^{ph} \qquad \forall i \in N_{hp}, \forall h \in H_p, \forall p \in P$$

$$v_i^t = C z_i - \sum_{p \in P} \sum_{h \in H_p} f^{pt} c_i^{ph} z_i^{ph} \qquad t = 0, \forall i \in N$$

$$v_i^t = v_i^{t-1} + \sum_{q \in Q} u_i^{qt} - \sum_{p \in P} \sum_{h \in H_p} f^{pt} c_i^{ph} z_i^{ph} \qquad t \neq 0, \forall i \in N, \forall t \in T$$

$$u_i^{qt} \geq C y_i^{qt} - v_i^{t-1} \qquad \forall i \in N, \forall q \in Q, \forall t \in T$$

Original Version of MILP Formulation

$$u_i^{qt} \leq C - v_i^{t-1} \qquad \forall i \in N, \forall q \in Q, \forall t \in T$$

$$\sum_{i \in N'} u_i^{qt} \leq L \qquad \forall q \in Q, \forall t \in T$$

$$\sum_{j \in N_0} x_{ij}^{qt} = \sum_{j \in N_0} x_{ji}^{qt} \qquad \forall i \in N, \forall q \in Q, \forall t \in T$$

$$\sum_{j \in N_0} x_{ij}^{qt} = y_i^{qt} \qquad \forall i \in N, \forall q \in Q, \forall t \in T$$

$$w_i^{qt} - w_j^{qt} + L x_{ij}^{qt} \leq L - u_j^{qt} \qquad \forall i, j \in N, \forall q \in Q, \forall t \in T$$

$$u_i^{qt} \leq w_i^{qt} \leq L \qquad \forall i \in N, \forall q \in Q, \forall t \in T$$

$$z_i, z_i^p, z^p, z_i^{ph} \in \{0, 1\} \qquad \forall i \in N, \forall p \in P, \forall i \in N$$

$$0 \leq v_i^t \leq C z_i \qquad \forall i \in N, \forall t \in T$$

$$0 \leq u_i^{qt} \leq L y_i^{qt} \qquad \forall i \in N, \forall q \in Q, \forall t \in T$$

$$x_{ij}^{qt} \in \{0, 1\} \qquad \forall i, j \in N_0, \forall q \in Q, \forall t \in T$$

$$y_i^{qt} \in \{0, 1\} \qquad \forall i \in N, \forall q \in Q, \forall t \in T$$

Step 1: Population Initialization

- Solution i:

Station 1				Station 2				Station 4			
$x_{5,1}^{q,1}$	$x_{5,1}^{q,2}$	$x_{5,1}^{q,3}$	$x_{5,1}^{q,4}$	$x_{5,2}^{q,1}$	$x_{5,2}^{q,2}$	$x_{5,2}^{q,3}$	$x_{5,2}^{q,4}$	$x_{5,4}^{q,1}$	$x_{5,4}^{q,2}$	$x_{5,4}^{q,3}$	$x_{5,4}^{q,4}$
t=1	t=2	t=3	t=4	t=1	t=2	t=3	t=4	t=1	t=2	t=3	t=4

- Genes in each chromosome: $x_{i,j}^{q,t}$

$$x_{i,j}^{q,t} = \begin{cases} 1, \text{ if truck } q \text{ delivers gas from node } i \text{ (Depot 5) to Station } j \text{ at the end of time } t; \\ 0, \text{ otherwise.} \end{cases}$$

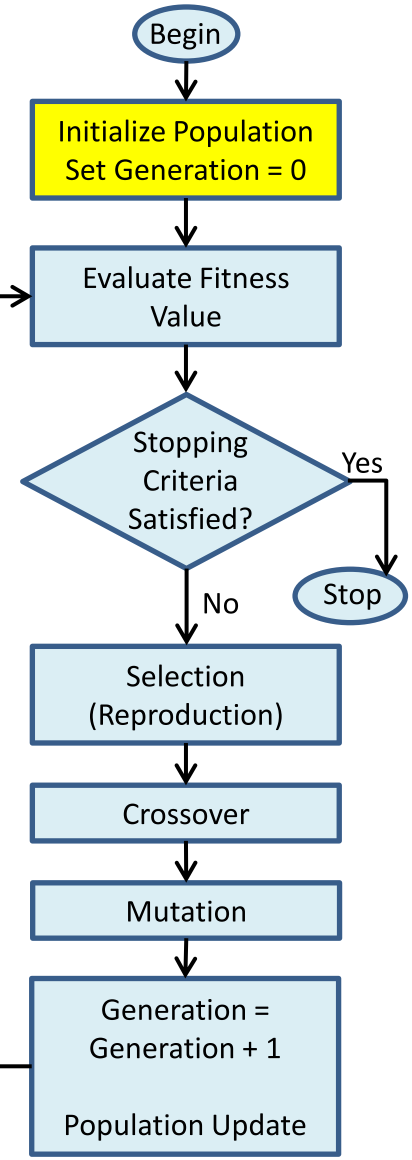
- 12 bits need for this example to represent chromosome encoding

→ Set Space = $2^{12} = 4096$

- Initial population (chromosomes) are **randomly** created as follows:

Generation 0

Solution 1	0	0	0	0	0	0	1	0	0	0	1	0
Solution 2	0	1	1	1	0	1	0	1	0	0	0	1
Solution 3	0	0	1	0	0	1	0	1	0	1	0	1
Solution 4	0	1	0	0	0	1	0	0	0	1	0	0



Step 2: Fitness Function

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i\in N}\sum_{t\in T} h_i^t v_i^t + \sum_{i,j\in N_0}\sum_{q\in Q}\sum_{t\in T} d_{ij}^t x_{ij}^{qt}$$

$$\delta \underbrace{\sum_{i\in N}\sum_{t\in T} g\left(v_i^t\right)}$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g\left(v_i^t\right) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Solution 1

0	0	0	0	0	0	1	0	0	0	1	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	100	45	-5
2	180	32	-102	224
			300	
4	185	49	-117	209
			300	

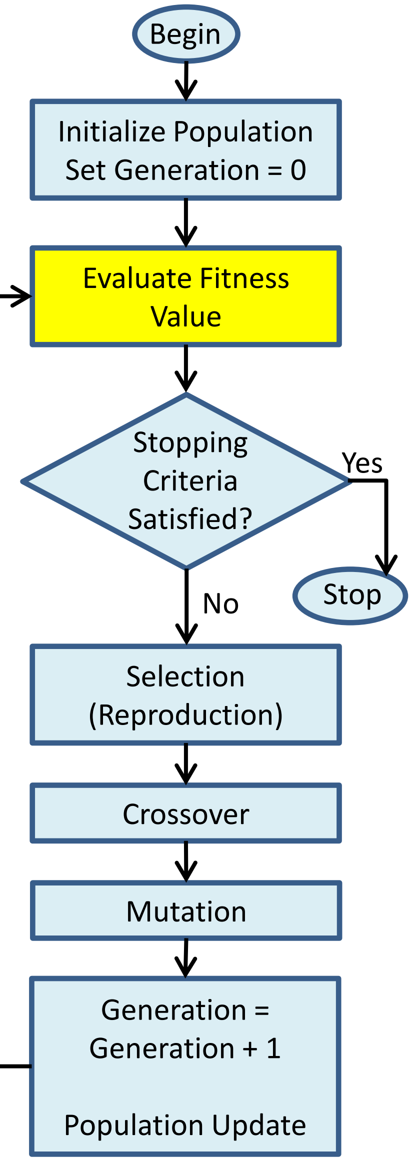
Inventory holding cost

Delivery cost

objective function = 1,229 + 800 = **2,029**

violation = **22,400,000**

Fitness of **Solution 1** = **22,402,029**



Step 2: Fitness Function

- ❑ Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i\in N}\sum_{t\in T} h_i^t v_i^t + \sum_{i,j\in N_0}\sum_{q\in Q}\sum_{t\in T} d_{ij}^t x_{ij}^{qt}$$

$$\delta \sum_{i\in N}\sum_{t\in T} g(v_i^t)$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Solution 2

0	1	1	1	0	1	0	1	0	0	0	1

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	100	245	250
		200	**55**	**50**
2	180	32	166	90
		268		**210**
4	185	49	-117	-91
				300

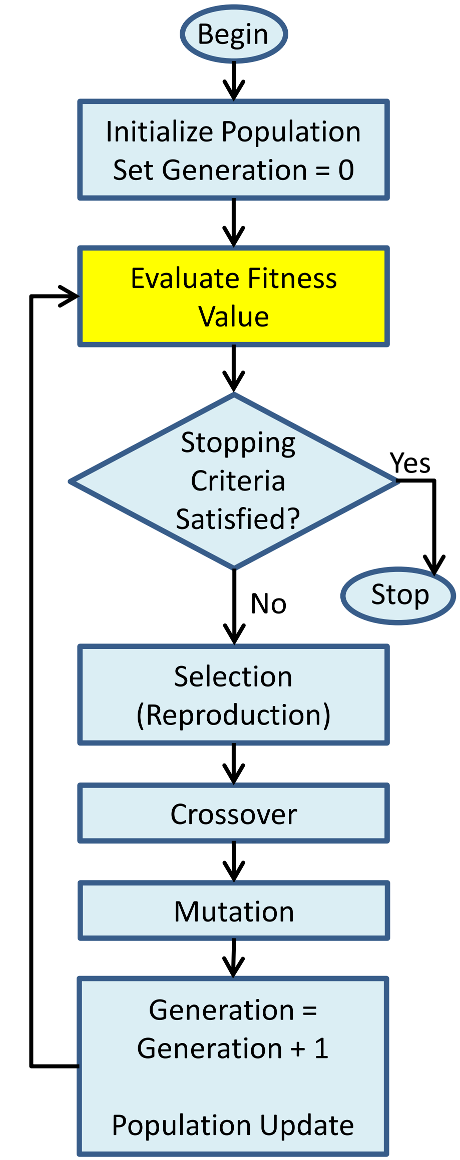
Inventory holding cost

Delivery cost

objective function = 1,502 + 3,700 = **5,202**

violation = **20,800,000**

Fitness of **Solution 2** = **20,805,202**



Step 2: Fitness Function

❑ Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i \in N} \sum_{t \in T} h_i^t v_i^t + \sum_{i,j \in N_0} \sum_{q \in Q} \sum_{t \in T} d_{ij}^t x_{ij}^{qt} \qquad \delta \sum_{i \in N} \sum_{t \in T} g(v_i^t)$$

h_i^t: $1 per item per time at i

d_{ij}^t: $100 × Distance

δ: Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Solution 3

0	0	1	0	0	1	0	1	0	1	0	1

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	100	45	250
			255	
2	180	32	166	90
		268		**210**
4	185	49	134	43
		251		**300**

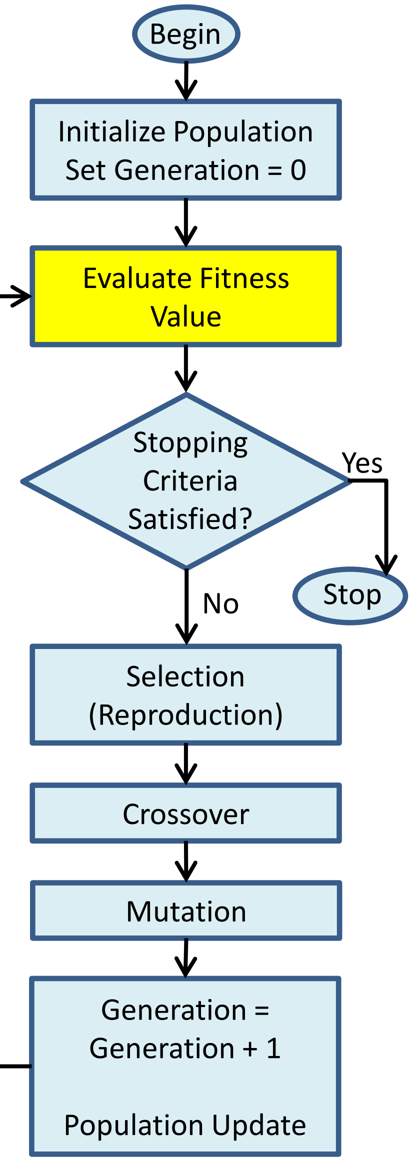
Inventory holding cost

Delivery cost

objective function = 1,479 + 2,400 = **3,879**

violation = **0**

Fitness of **Solution 3** = **3,879**



Step 2: Fitness Function

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i\in N}\sum_{t\in T} h_i^t v_i^t + \sum_{i,j\in N_0}\sum_{q\in Q}\sum_{t\in T} d_{ij}^t x_{ij}^{qt}$$

$$\delta \sum_{i\in N}\sum_{t\in T} g(v_i^t)$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Solution 4

0	1	0	0	0	1	0	0	0	1	0	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	100	245	195
		200		
2	180	32	166	90
		268		
4	185	49	134	43
		251		

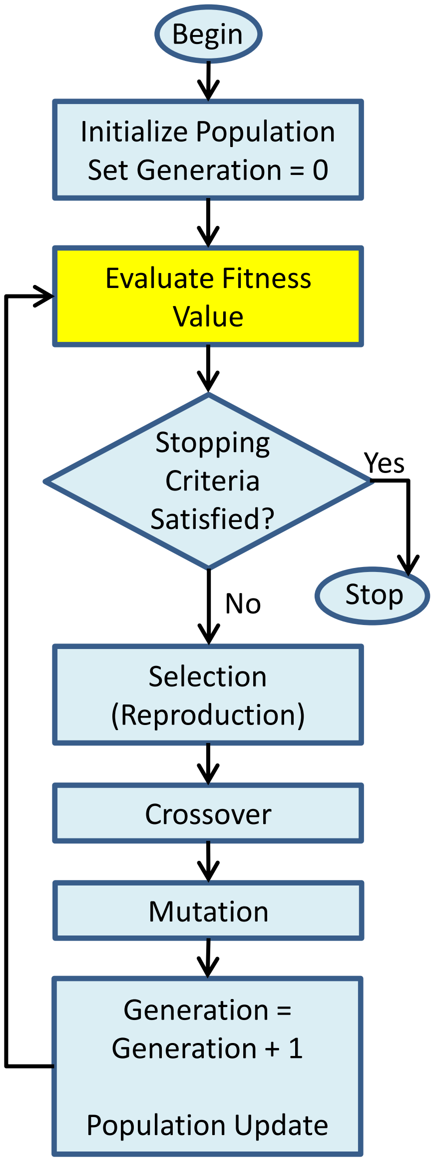
Inventory holding cost

Delivery cost

objective function = 1,624 + 1,600 = **3,224**

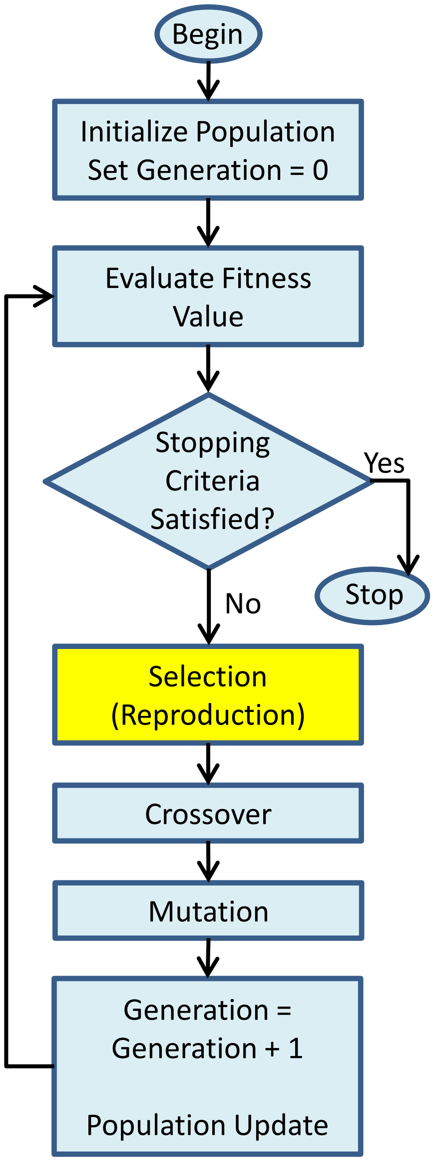
violation = **0**

Fitness of **Solution 4** = **3,224**



Step 3: Tournament Selection

- A selection technique selects individuals from the population to insert individual into **mating pool**.
- Individuals from **the mating pool** are used to generate new offspring, with the resulting offspring forming the basis of the next generation.
- A selection technique in a GAs is simply a process that favors the selection of better individuals in the population for the mating pool.
- In this example, **a tournament selection operator** is used as reproduction operator because **the tournament selection** has better or equivalent convergence and computational time complexity properties when compared to any other reproduction operator that exists in the literature.
- In **the tournament selection**, **tournaments** are played between k solutions (k is tournament size) and the better solution is chosen and placed in **the mating pool** .
- k other solutions are picked again and another slot in **the mating pool** is filled with the better solution.
- The user specifies the size of the tournament set as a percentage of the total population.



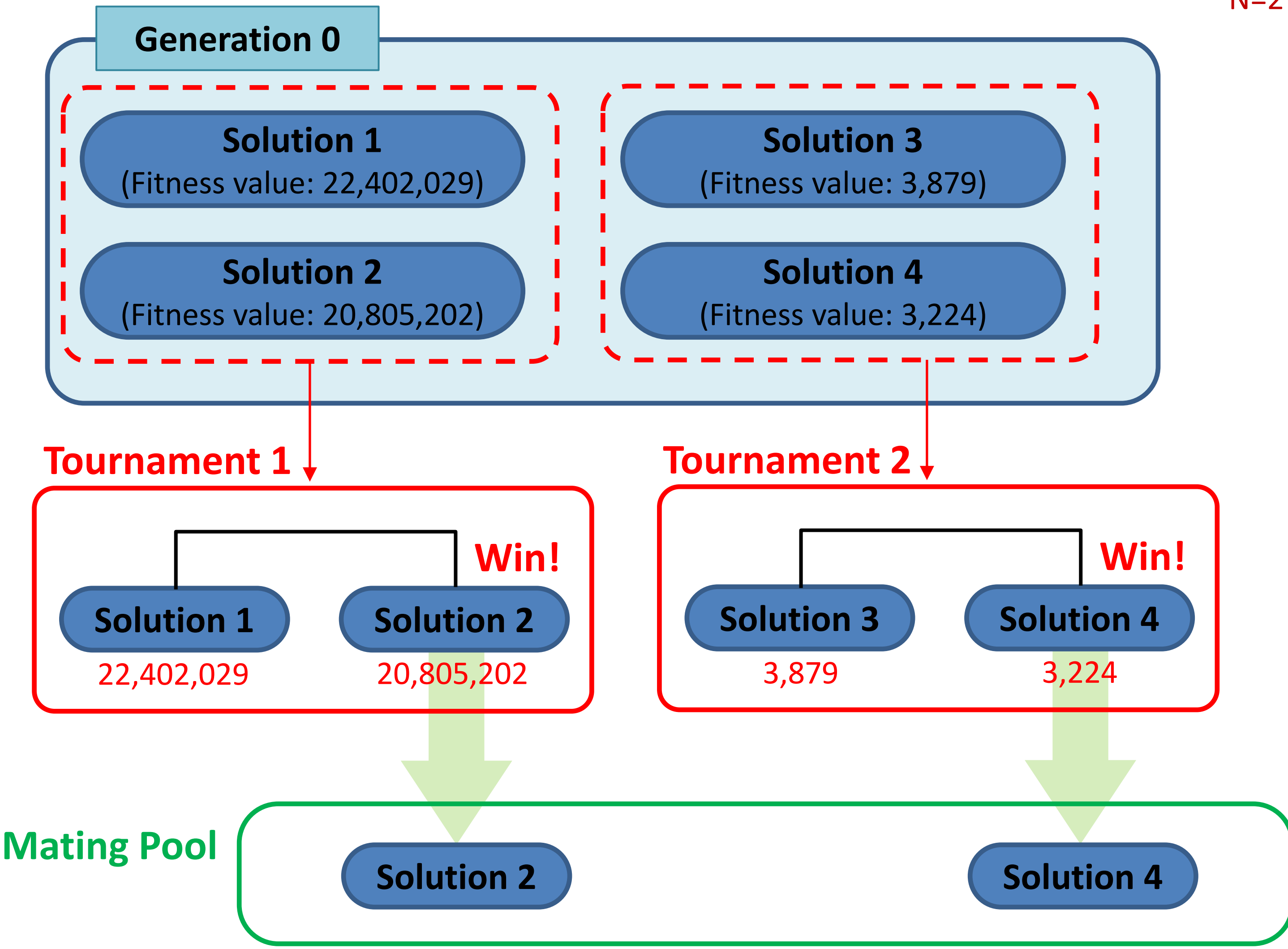
Step 3: Tournament Selection

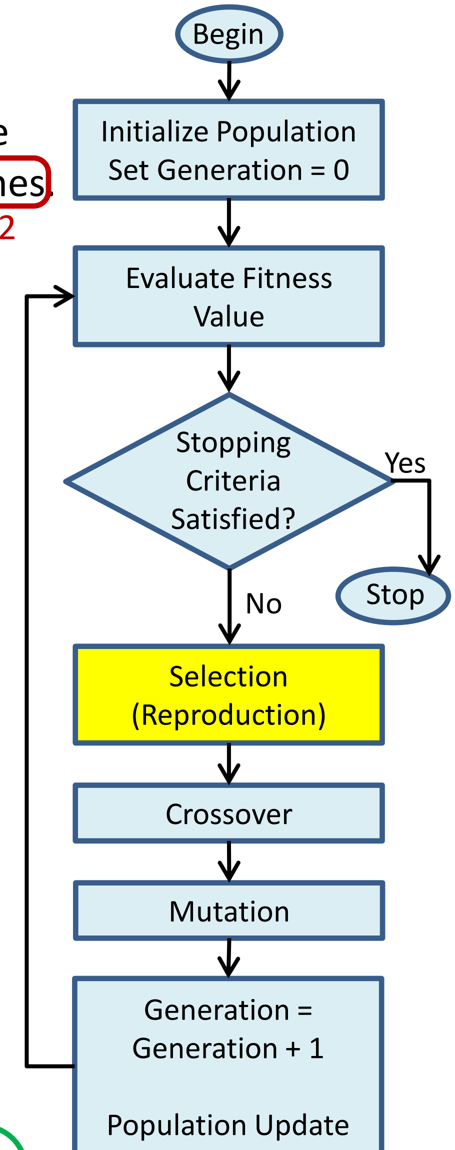
❑ (Binary) Tournament Selection

k=2

Choose some number k of individuals **randomly** from the population and copy the best individual from this group into the intermediate population, and repeat N times.

N=2





Step 4: Uniform Crossover

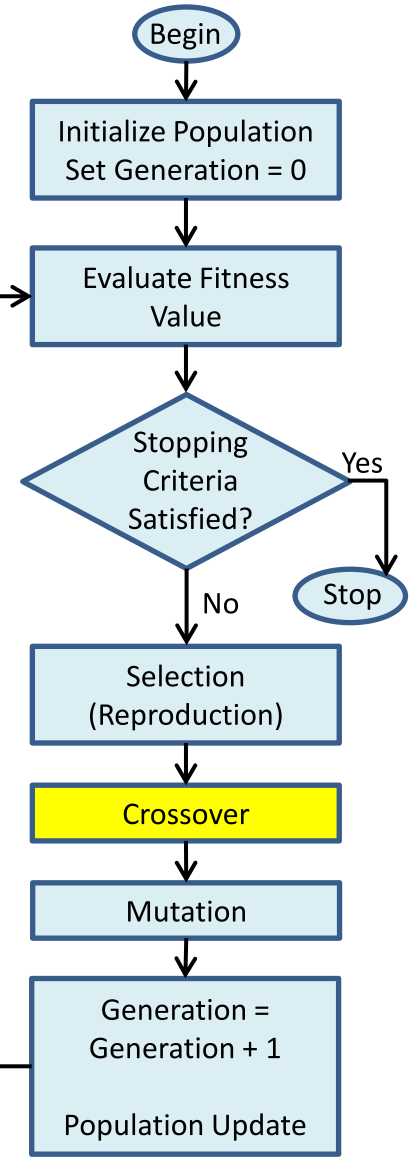
- A crossover operator takes the selected individuals (chromosomes) for mating and mixes the gene to produce offspring.
- In this example, a Uniform crossover is used, so each bit (i.e., gene) is chosen from either parent with equal probability. (Other mixing ratios can be used, resulting in offspring which inherit more genetic information from one parent than the other.)

1st Uniform Crossover

(Generation 0) Solution 2	0	1	1	1	0	1	0	1	0	0	0	1
(Generation 0) Solution 4	0	1	0	0	0	1	0	0	0	1	0	0
(Generation 1) Offspring 1	0	1	0	1	0	1	0	0	0	1	0	0

2nd Uniform Crossover

(Generation 0) Solution 2	0	1	1	1	0	1	0	1	0	0	0	1
(Generation 0) Solution 4	0	1	0	0	0	1	0	0	0	1	0	0
(Generation 1) Offspring 2	0	1	1	0	0	1	0	0	0	0	0	1



Step 5: Uniform Mutation

- A mutation operator introduces the diversity within the population so that search algorithm does not necessarily get stuck at local maxima.
- In this example, a Uniform mutation is used, so this operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene.

Uniform Mutation for Offspring 1

(Generation 1) Offspring 1
Before Mutation operator

0	1	0	1	0	1	0	0	0	1	0	0

(Generation 1) Offspring 1
After Mutation operator

1	1	0	1	0	1	0	0	0	1	0	0

The mutation operator only flipped the value of this bit.

Uniform Mutation for Offspring 2

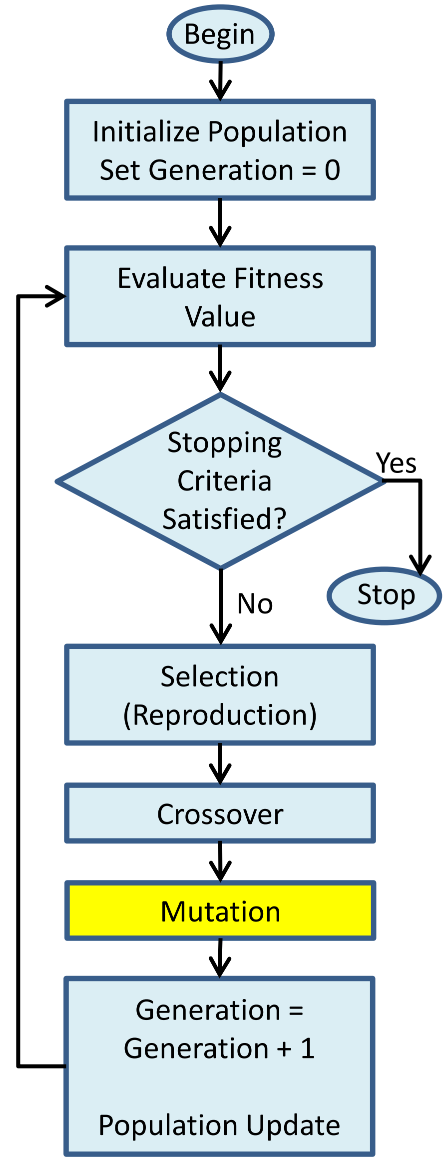
(Generation 1) Offspring 2
Before Mutation operator

0	1	1	0	0	1	0	0	0	0	0	1

(Generation 1) Offspring 2
After Mutation operator

0	1	1	0	0	1	0	0	0	0	0	1

The mutation operator did NOT flip the value of any bit.

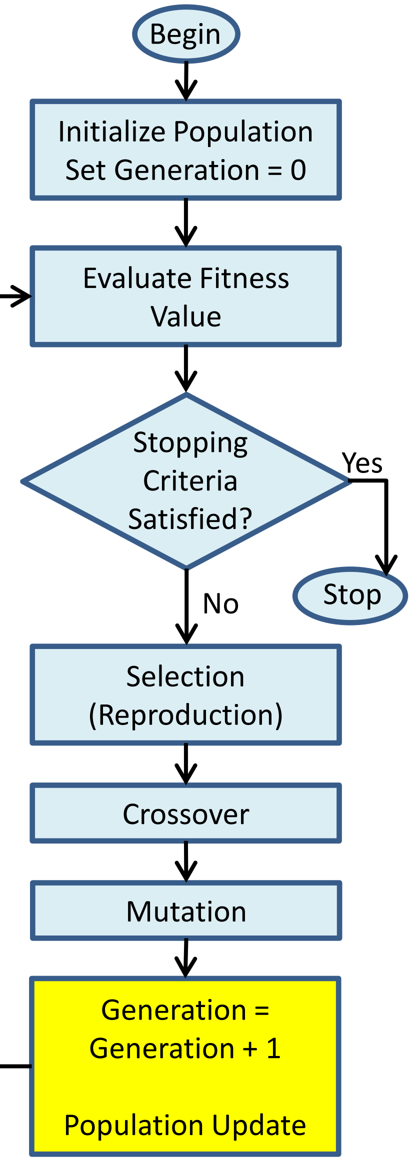


Step 6: Population Update

- Generation = Generation + 1 ➡ **Generation 1**
- Population (chromosomes) for **Generation 1** is updated as follows (Note that population can be updated in other ways depending on the user's preferences):

Generation 1

~~Solution 1~~ **(Eliminated)**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~
Solution 2	0	1	1	1	0	1	0	1	0	0	0	1
~~Solution 3~~ **(Eliminated)**	~~0~~	~~0~~	~~1~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~0~~	~~1~~	~~0~~	~~1~~
Solution 4	0	1	0	0	0	1	0	0	0	1	0	0
Offspring 1 **(After Mutation)**	1	1	0	1	0	1	0	0	0	1	0	0
Offspring 2 **(After Mutation)**	0	1	1	0	0	1	0	0	0	0	0	1



Step 2: Fitness Function (Generation 1)

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i \in N} \sum_{t \in T} h_i^t v_i^t + \sum_{i,j \in N_0} \sum_{q \in Q} \sum_{t \in T} d_{ij}^t x_{ij}^{qt} \qquad \delta \sum_{i \in N} \sum_{t \in T} g(v_i^t)$$

h_i^t: $1 per item per time at i

d_{ij}^t: $100 × Distance

δ: Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Solution 2

0	1	1	1	0	1	0	1	0	0	0	1

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

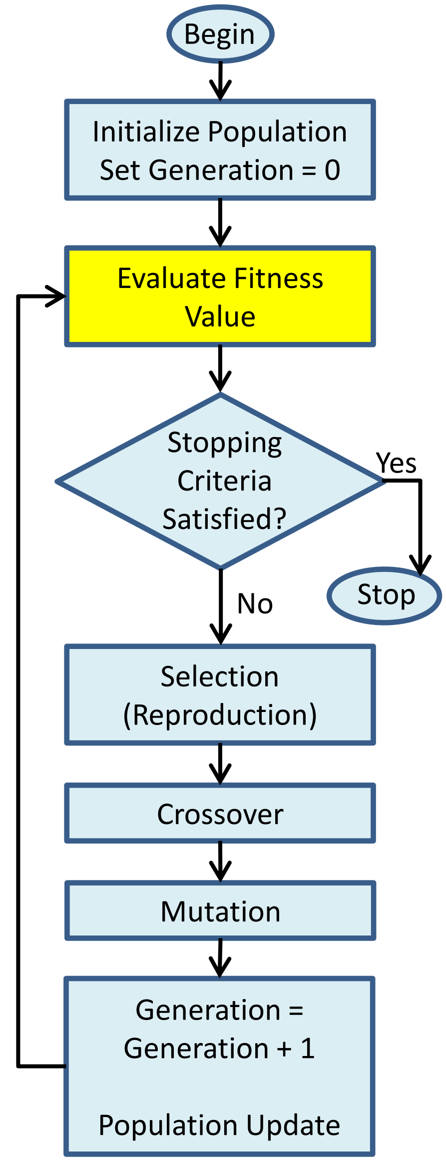
	1	2	3	4
1	205	100	245	250
		200	**55**	**50**
2	180	32	166	90
		268		**210**
4	185	49	-117	-91
				300

Inventory holding cost — Delivery cost

objective function = 1,502 + 3,700 = **5,202**

violation = **20,800,000**

Fitness of **Solution 2** = **20,805,202**



Step 2: Fitness Function (Generation 1)

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

objective function:

$$\text{Min.} \sum_{i \in N} \sum_{t \in T} h_i^t v_i^t + \sum_{i,j \in N_0} \sum_{q \in Q} \sum_{t \in T} d_{ij}^t x_{ij}^{qt}$$

h_i^t: $1 per item per time at i

d_{ij}^t: $100 × Distance

violation of constraints:

$$\delta \sum_{i \in N} \sum_{t \in T} g(v_i^t)$$

δ: Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Solution 4

0	1	0	0	0	1	0	0	0	1	0	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

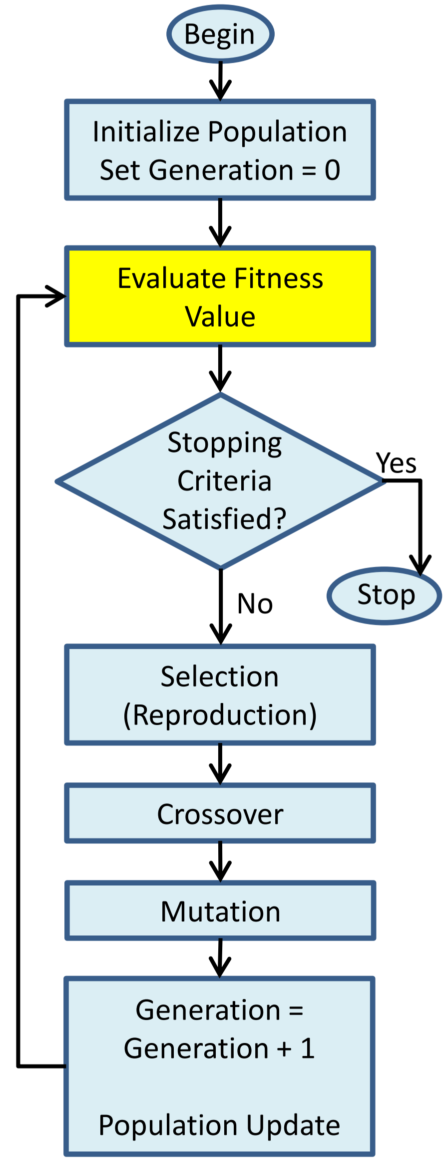
	1	2	3	4
1	205	100	245	195
		200		
2	180	32	166	90
		268		
4	185	49	134	43
		251		

Inventory holding cost   Delivery cost

objective function = 1,624 + 1,600 = **3,224**

violation = **0**

Fitness of **Solution 4** = **3,224**



Step 2: Fitness Function (Generation 1)

❑ Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i\in N}\sum_{t\in T} h_i^t v_i^t + \sum_{i,j\in N_0}\sum_{q\in Q}\sum_{t\in T} d_{ij}^t x_{ij}^{qt} \qquad \delta \sum_{i\in N}\sum_{t\in T} g(v_i^t)$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Offspring 1

1	1	0	1	0	1	0	0	0	1	0	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	195	245	195
	95	**105**		**105**
2	180	32	166	90
		268		
4	185	49	134	43
		251		

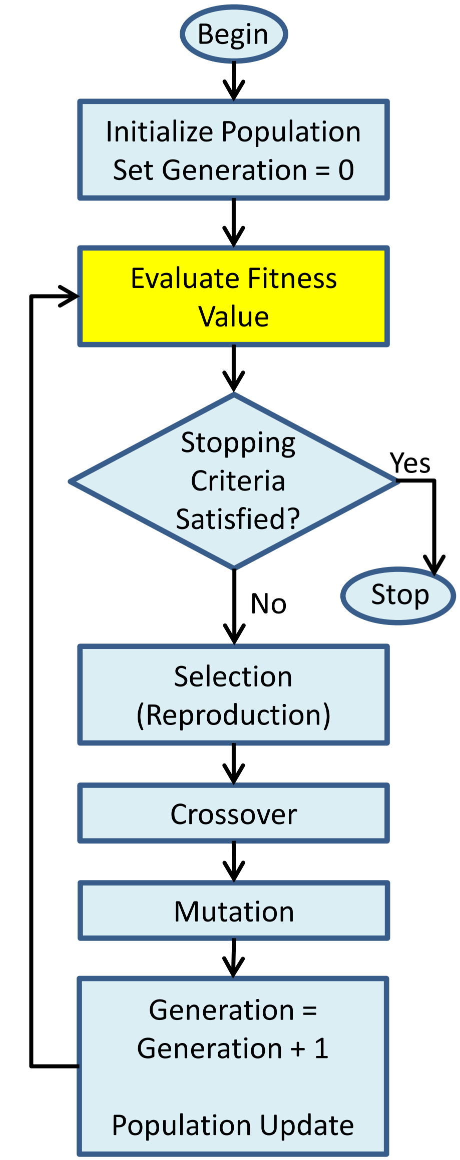
Inventory holding cost

Delivery cost

objective function = 1,719 + 3,200 = **4,919**

violation = **0**

Fitness of **Offspring 1** = **4,919**



Step 2: Fitness Function (Generation 1)

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i \in N} \sum_{t \in T} h_i^t v_i^t + \sum_{i,j \in N_0} \sum_{q \in Q} \sum_{t \in T} d_{ij}^t x_{ij}^{qt}$$

$$\delta \sum_{i \in N} \sum_{t \in T} g(v_i^t)$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Offspring 2

0	1	1	0	0	1	0	0	0	0	0	1

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	100	245	250
		200	**55**	
2	180	32	166	90
		268		
4	185	49	-117	-91
				300

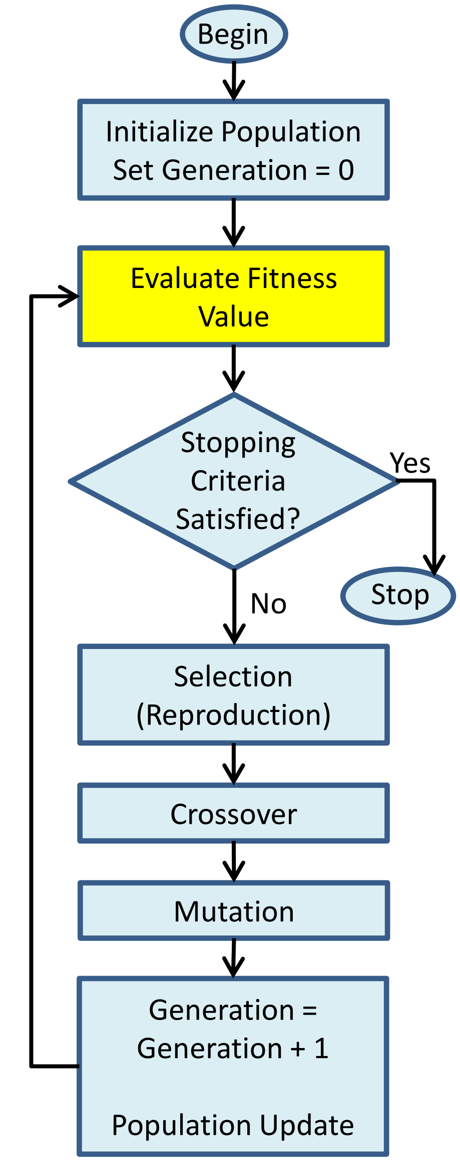
Inventory holding cost

Delivery cost

objective function = 1,502 + 2,400 = **3,902**

violation = **20,800,000**

Fitness of **Offspring 2** = **20,803,902**



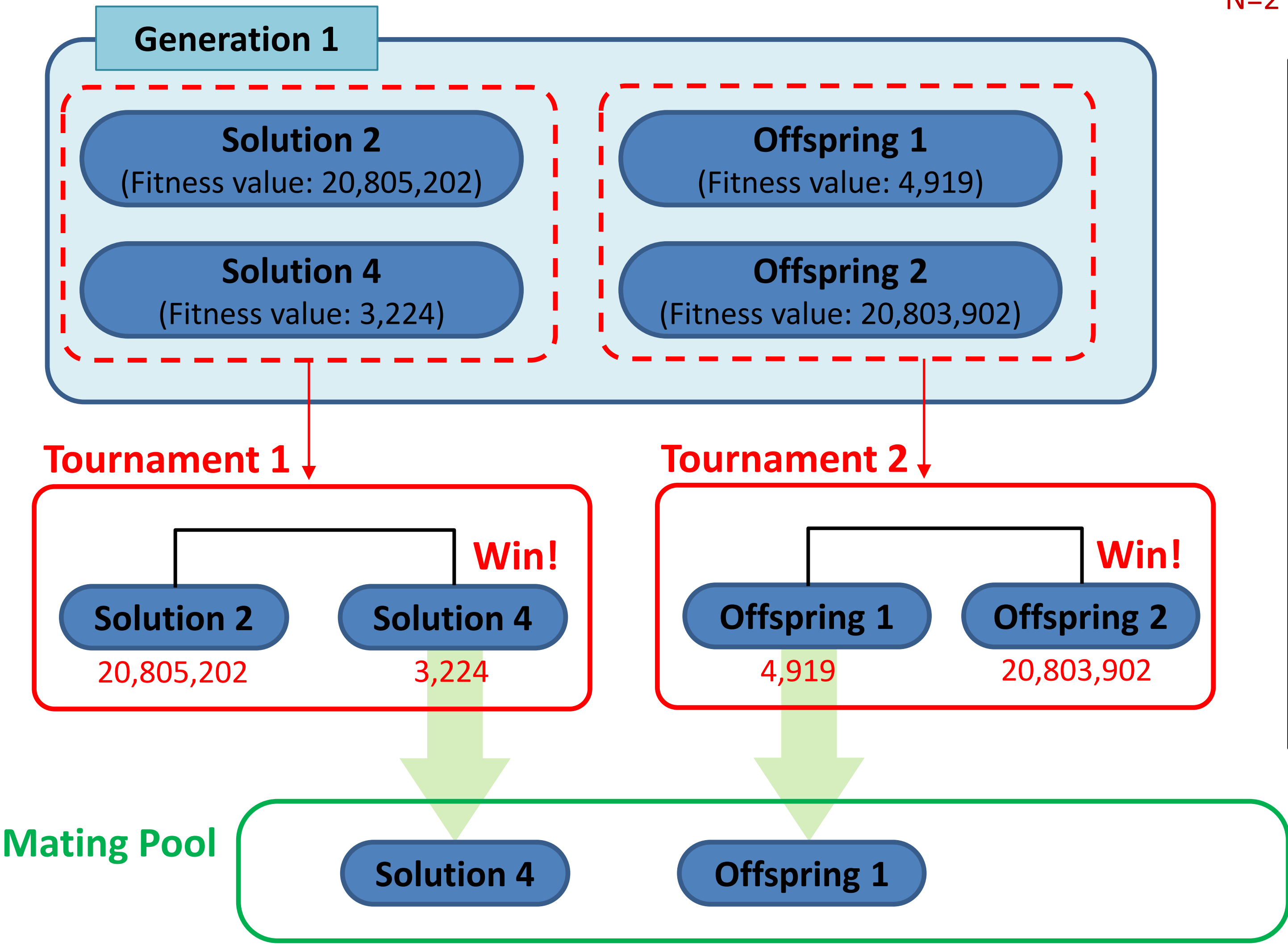
Step 3: Tournament Selection (Generation 1)

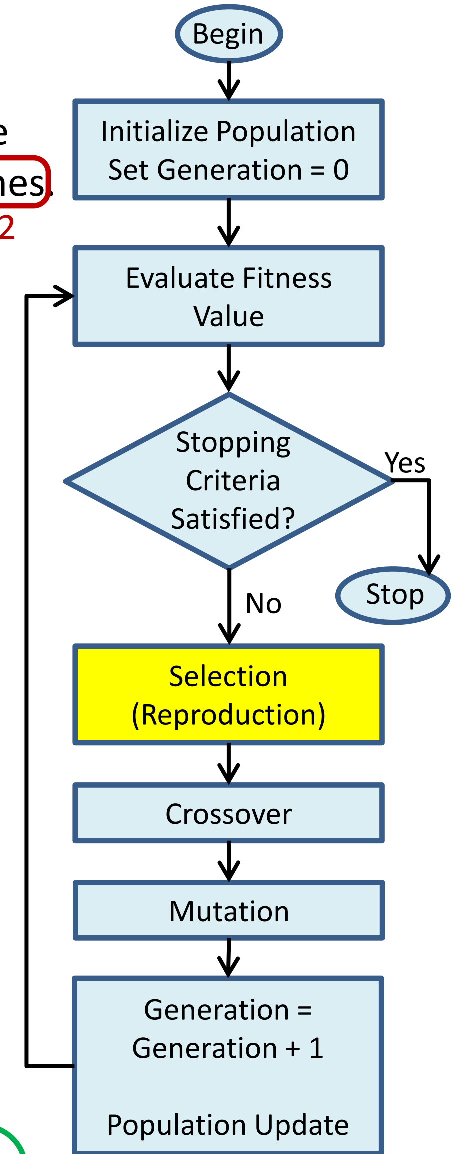
- (Binary) Tournament Selection

k=2

Choose some number k of individuals **randomly** from the population and copy the best individual from this group into the intermediate population, and repeat N times.

N=2





Step 4: Uniform Crossover (Generation 1)

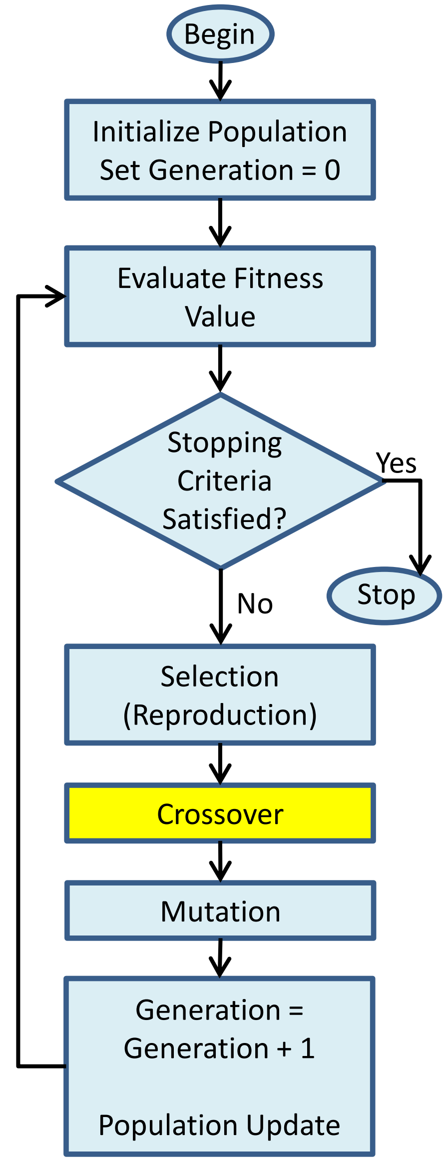
- A crossover operator takes the selected individuals (chromosomes) for mating and mixes the gene to produce offspring.
- In this example, a Uniform crossover is used, so each bit (i.e., gene) is chosen from either parent with equal probability. (Other mixing ratios can be used, resulting in offspring which inherit more genetic information from one parent than the other.)

1st Uniform Crossover

(Generation 1) Solution 4	0	1	0	0	0	1	0	0	0	1	0	0
(Generation 1) Offspring 1	1	1	0	1	0	1	0	0	0	1	0	0
(Generation 2) Offspring 3	1	1	0	0	0	1	0	0	0	1	0	0

2nd Uniform Crossover

(Generation 1) Solution 4	0	1	0	0	0	1	0	0	0	1	0	0
(Generation 1) Offspring 1	1	1	0	1	0	1	0	0	0	1	0	0
(Generation 2) Offspring 4	1	1	0	1	0	1	0	0	0	1	0	0



Step 5: Uniform Mutation (Generation 1)

- A mutation operator introduces the diversity within the population so that search algorithm does not necessarily get stuck at local maxima.
- In this example, a Uniform mutation is used, so this operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene.

Uniform Mutation for Offspring 3

(Generation 2) Offspring 3
Before Mutation operator

1	1	0	0	0	1	0	0	0	1	0	0

(Generation 2) Offspring 3
After Mutation operator

1	0	0	0	0	1	0	0	0	1	0	0

The mutation operator only flipped the value of this bit.

Uniform Mutation for Offspring 4

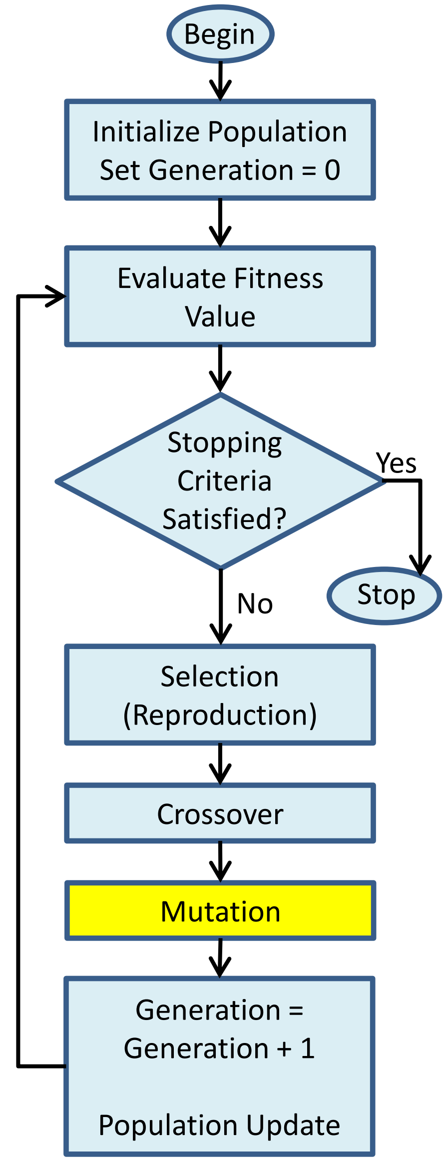
(Generation 2) Offspring 4
Before Mutation operator

1	1	0	1	0	1	0	0	0	1	0	0

(Generation 2) Offspring 4
After Mutation operator

1	1	0	0	0	1	0	0	0	1	0	0

The mutation operator only flipped the value of this bit.

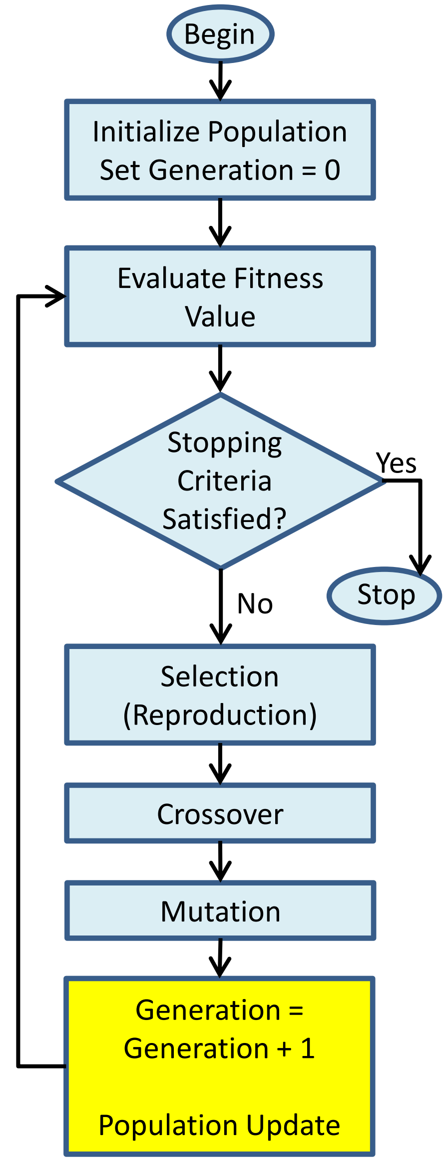


Step 6: Population Update (Generation 1)

- Generation = Generation + 1 ➡ **Generation 2**
- Population (chromosomes) for **Generation 2** is updated as follows (Note that population can be updated in other ways depending on the user's preferences):

Generation 2

~~Solution 2~~	~~0~~	~~1~~	~~1~~	~~1~~	~~0~~	~~1~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~1~~	**(Eliminated)**
Solution 4	0	1	0	0	0	1	0	0	0	1	0	0	
Offspring 1 **(After Mutation)**	1	1	0	1	0	1	0	0	0	1	0	0	
~~Offspring 2~~ **(After Mutation)**	~~0~~	~~1~~	~~1~~	~~0~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	**(Eliminated)**
Offspring 3 **(After Mutation)**	1	0	0	0	0	1	0	0	0	1	0	0	
Offspring 4 **(After Mutation)**	1	1	0	0	0	1	0	0	0	1	0	0	



Step 2: Fitness Function (Generation 2)

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i\in N}\sum_{t\in T} h_i^t v_i^t + \sum_{i,j\in N_0}\sum_{q\in Q}\sum_{t\in T} d_{ij}^t x_{ij}^{qt}$$

$$\delta \sum_{i\in N}\sum_{t\in T} g(v_i^t)$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Solution 4

0	1	0	0	0	1	0	0	0	1	0	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	100	245	195
		200		
2	180	32	166	90
		268		
4	185	49	134	43
		251		

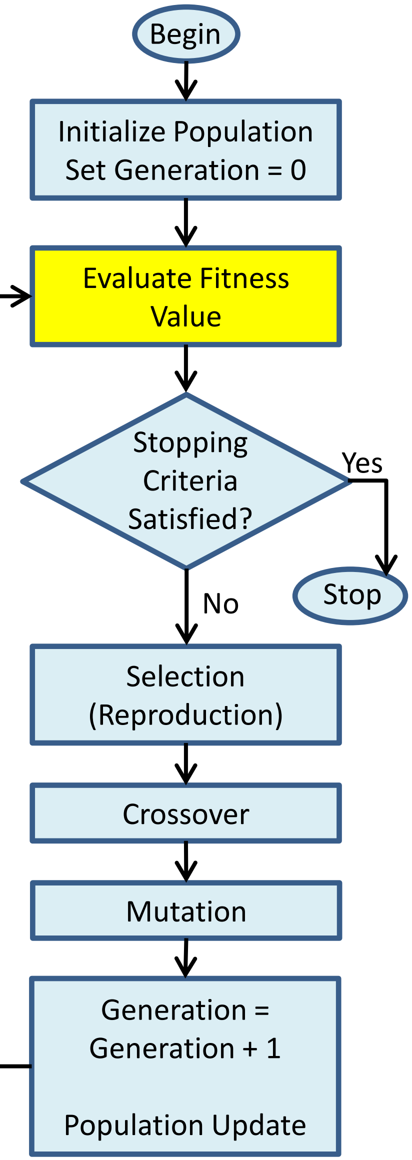
Inventory holding cost

Delivery cost

objective function = 1,624 + 1,600 = **3,224**

violation = **0**

Fitness of **Solution 4** = **3,224**



Step 2: Fitness Function (Generation 2)

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i\in N}\sum_{t\in T} h_i^t v_i^t + \sum_{i,j\in N_0}\sum_{q\in Q}\sum_{t\in T} d_{ij}^t x_{ij}^{qt} \qquad \delta \sum_{i\in N}\sum_{t\in T} g(v_i^t)$$

h_i^t: $1 per item per time at i

d_{ij}^t: $100 × Distance

δ: Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Offspring 1

1	1	0	1	0	1	0	0	0	1	0	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

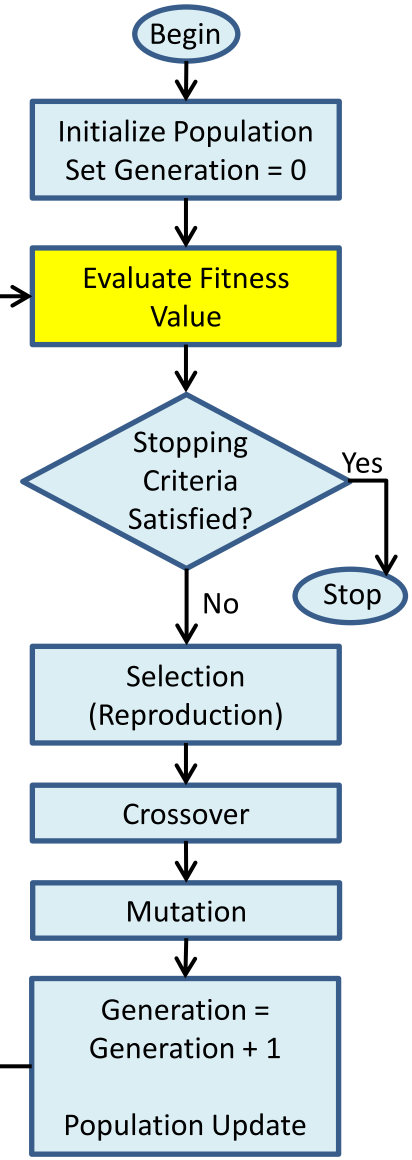
	1	2	3	4
1	205	195	245	195
	95	**105**		**105**
2	180	32	166	90
		268		
4	185	49	134	43
		251		

Inventory holding cost   Delivery cost

objective function = 1,719 + 3,200 = **4,919**

violation = **0**

Fitness of **Offspring 1** = **4,919**



Step 2: Fitness Function (Generation 2)

- Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i\in N}\sum_{t\in T} h_i^t v_i^t + \sum_{i,j\in N_0}\sum_{q\in Q}\sum_{t\in T} d_{ij}^t x_{ij}^{qt}$$

$$\delta \sum_{i\in N}\sum_{t\in T} g(v_i^t)$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Offspring 3

1	0	0	0	0	1	0	0	0	1	0	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	195	140	90
	95			
2	180	32	166	90
		268		
4	185	49	134	43
		251		

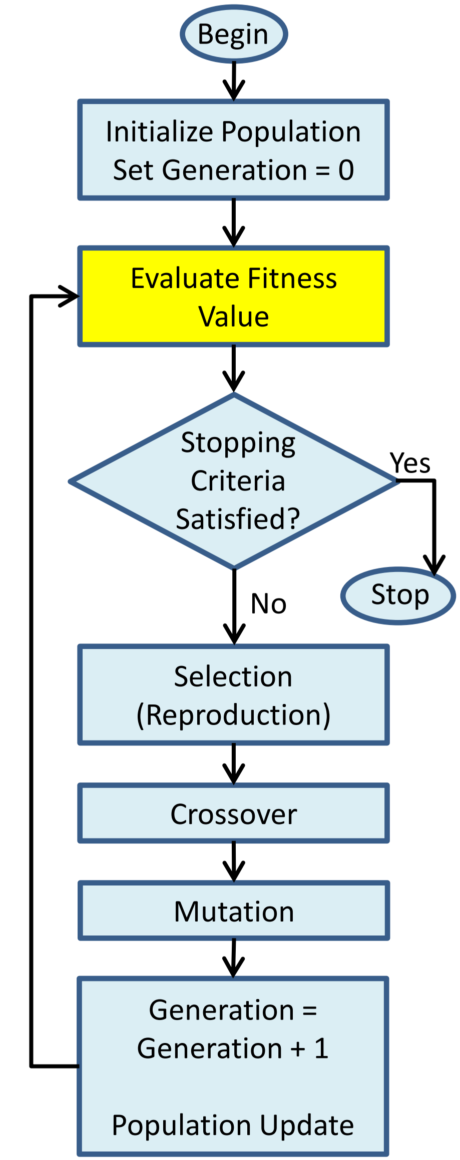
Inventory holding cost

Delivery cost

objective function = 1,509 + 1,600 = **3,109**

violation = **0**

Fitness of **Offspring 3** = **3,109**



Step 2: Fitness Function (Generation 2)

❑ Fitness Function

Fitness of Solution i = objective function + violation of constraints

$$\text{Min.} \sum_{i \in N} \sum_{t \in T} h_i^t v_i^t + \sum_{i,j \in N_0} \sum_{q \in Q} \sum_{t \in T} d_{ij}^t x_{ij}^{qt} \qquad \delta \sum_{i \in N} \sum_{t \in T} g(v_i^t)$$

$1 per item per time at i

$100 × Distance

Penalty parameter = 100,000

$$g(v_i^t) = \begin{cases} v_i^t, & \text{if } v_i^t < 0 \\ 0, & \text{otherwise} \end{cases}, \forall i \in N, \forall t \in T$$

Offspring 4

1	1	0	0	0	1	0	0	0	1	0	0

Fuel Demand at Stations 1,2,4

	1	2	3	4
1	95	105	55	50
2	120	148	134	76
4	115	136	166	91

Inventory remained at Stations 1,2,4

	1	2	3	4
1	205	195	245	195
	95	**105**		
2	180	32	166	90
		268		
4	185	49	134	43
		251		

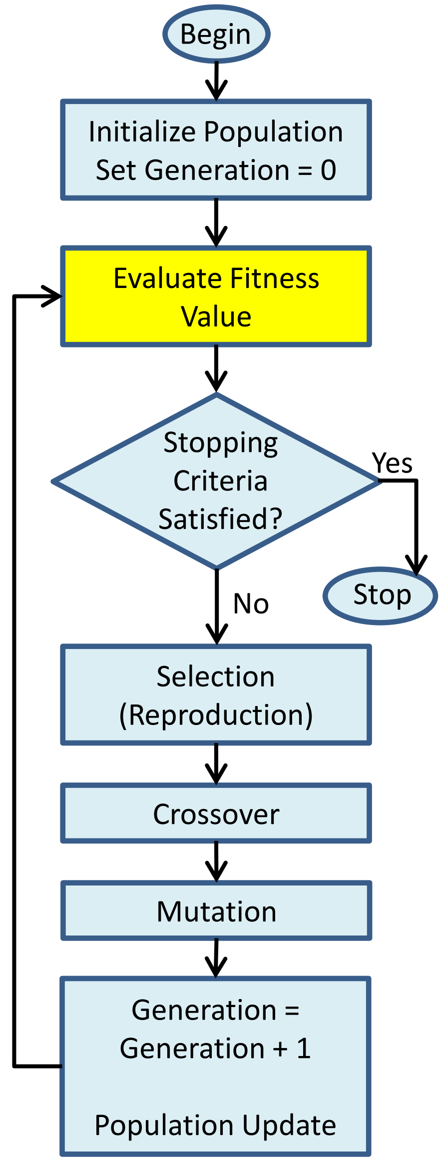
Inventory holding cost

Delivery cost

objective function = 1,719 + 2,400 = **4,119**

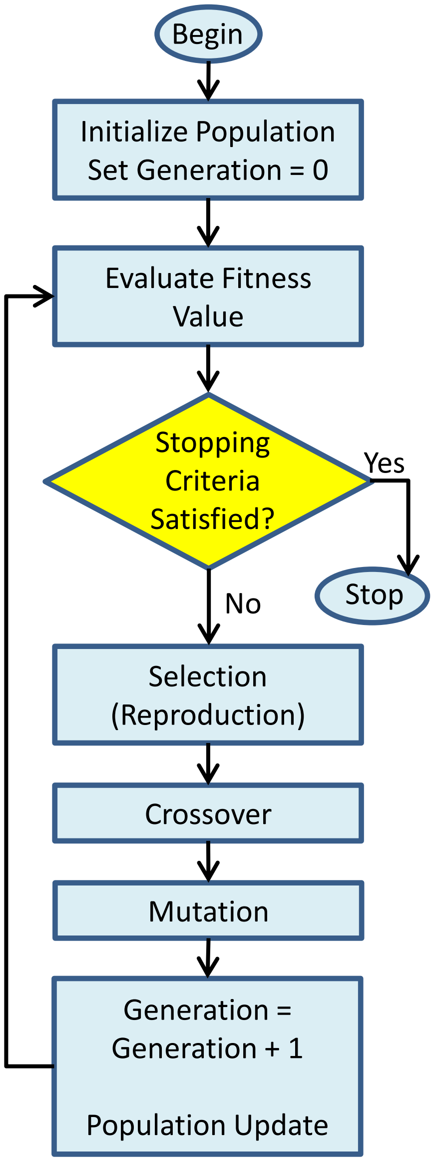
violation = **0**

Fitness of **Offspring 4** = **4,119**



Step 2.1: Stopping Criteria (Generation 2)

- There exist three popular stopping criteria
  - **Stopping criterion 1:** maximum number of iterations (generations)
    - When the generation reaches to the predefined maximum number of generations, it stops and provides the best solution in the last generation.
  - **Stopping criterion 2:** solution improvement
    - A GA proceeds until the best solution during the evolution process doesn't change to a better value for a predefined value of generations. This predefined value can be 20% or 30% of the generation number which the best solution has found so far. For example, the algorithm reaches to a value of 200 at generation 50, and then this value doesn't change for next 15 generations (30% of 50), then the algorithm stops.
  - **Stopping criterion 3:** predetermined value
    - When the fitness value has reached a certain predetermined value, it stops.



Step 2.1: Stopping Criteria (Generation 2)

- In this example, we end a GA with Offspring 3, which is a real optimal solution.

Generation 0

Solution 1
(Fitness value: 22,402,029)

Solution 3
(Fitness value: 3,879)

Solution 2
(Fitness value: 20,805,202)

Solution 4
(Fitness value: 3,224)

Generation 1

Solution 2
(Fitness value: 20,805,202)

Offspring 1
(Fitness value: 4,919)

Solution 4
(Fitness value: 3,224)

Offspring 2
(Fitness value: 20,803,902)

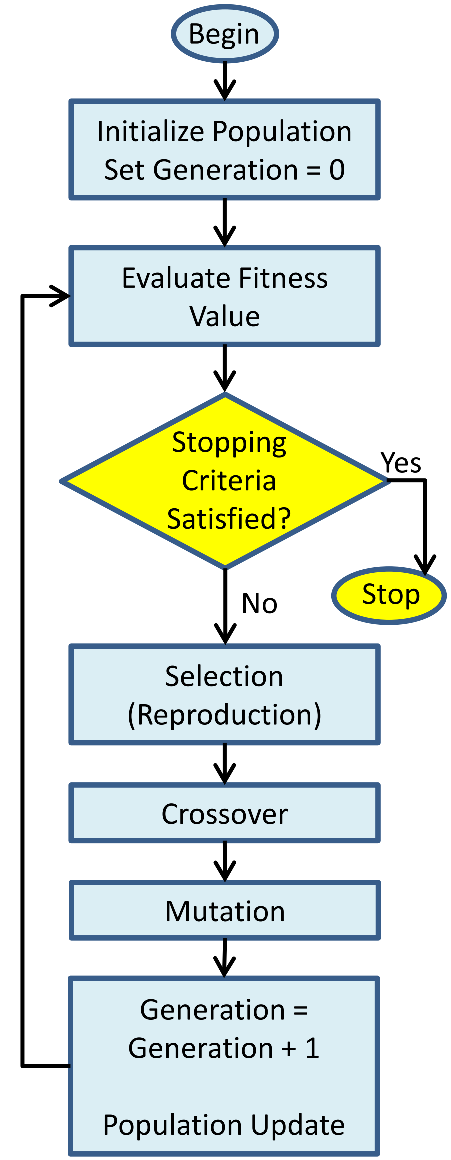
Generation 2

Solution 4
(Fitness value: 3,224)

Offspring 3
(Fitness value: 3,109)

Offspring 1
(Fitness value: 4,919)

Offspring 4
(Fitness value: 4,119)



Genetic Algorithm Example 2:
The Traveling Salesman Problem

Traveling Salesman Problem (TSP)

A traveling salesman must visit each of n cities before returning home. He knows the distance between each of the cities and wishes to minimize the total distance traveled while visiting all of the cities.

Problem: In what order should he visit the cities?

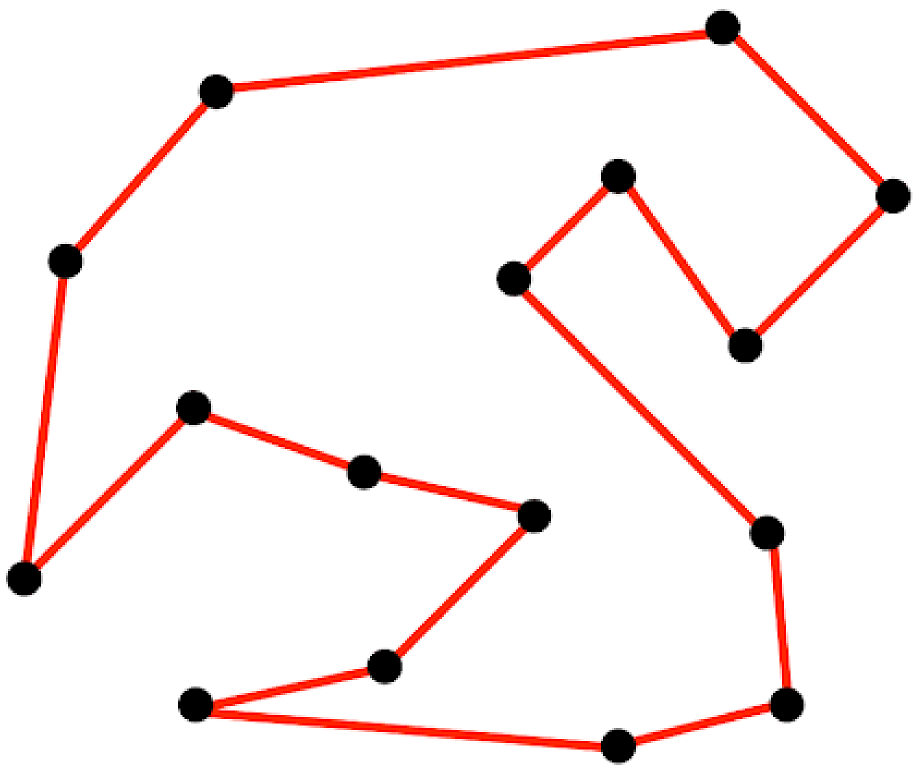
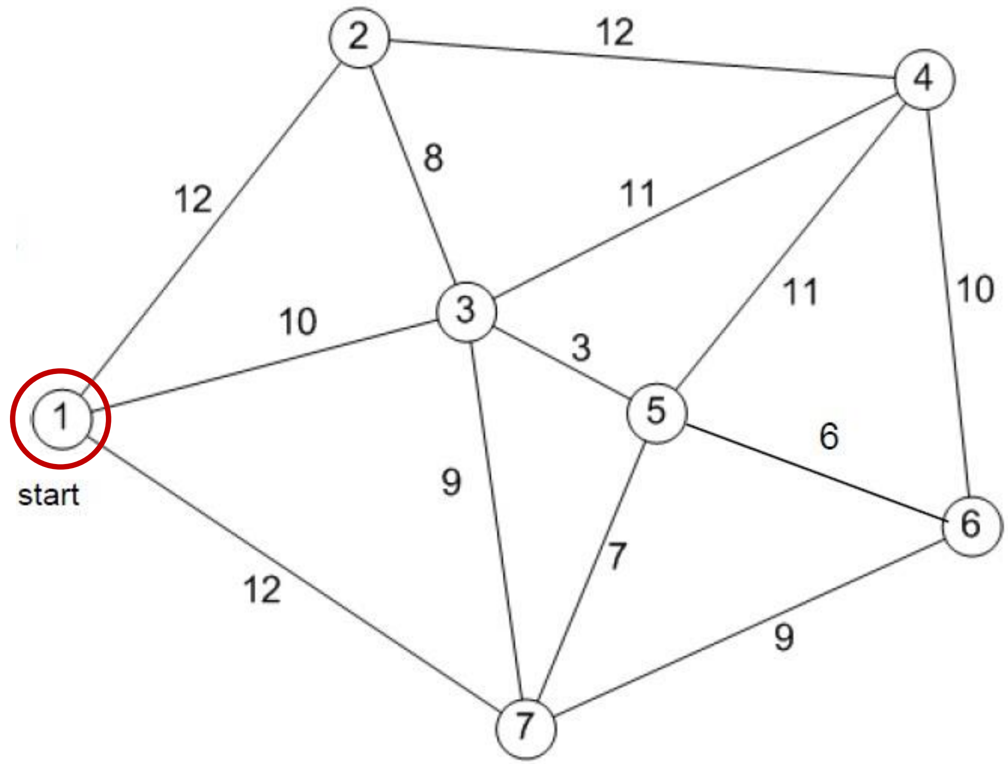


Illustration of a TSP solution

Example of TSP

- Starting from City 1, the salesman must travel to all cities once before returning home.
- The distance between each city is given, and is assumed to be the same in both direction (Symmetric TSP). If this assumption is relaxed, it is called Asymmetric TSP.
- Only the links shown are to be used.
- **Objective:** Minimize the total distance to be traveled.



TSP is NP-hard since it is **O(n!)** to investigate all the possible cases (TSP is one of famous combinatorial optimization problem).

For example, 10 Cities = 3,628,800 cases
20 Cities = 2,432,902,008,176,640,000 cases
30 Cities = 265,252,859,812,191,058,636,308,480,000,000 cases

Formulation for TSP

Let c_{ij} = distance from city i to city j, for all city pairs (i, j),

$N = \{1, 2, \dots, n\}$: set of cities.

Variables: $x_{ij} = \begin{cases} 1 & \text{if he goes directly from city i to city j}, \\ 0 & \text{otherwise}, \end{cases}$ for all city pairs (i, j).

Constraints:

✓ He leaves city i exactly once: $\sum_{j=1,\, j\neq i}^{n} x_{ij} = 1, \quad i = 1, \dots, n.$

✓ He arrives at city j exactly once: $\sum_{i=1,\, i\neq j}^{n} x_{ij} = 1, \quad j = 1, \dots, n.$

Requires high complexity!

✓ Sub-tour elimination constraints:

$$\sum_{i\in S} \sum_{j\notin S} x_{ij} \geq 1, \qquad \text{for all } S \subset N,\ 1 \leq |S| \leq n-1,$$

or

$$\sum_{i\in S} \sum_{j\in S\setminus\{i\}} x_{ij} \leq |S| - 1, \qquad \text{for all } S \subset N,\ 2 \leq |S| \leq n-1.$$

Binary Linear Programming Formulation for TSP

Formulation:

$$\text{minimize} \quad z = \sum_{i=1}^{n} \sum_{j=1,\, j\neq i}^{n} c_{ij}\, x_{ij}\ ,$$

$$\text{subject to} \quad \sum_{j=1,\, j\neq i}^{n} x_{ij} = 1, \qquad i = 1, \dots, n,$$

$$\sum_{i=1,\, i\neq j}^{n} x_{ij} = 1, \qquad j = 1, \dots, n,$$

$$\sum_{i\in S} \sum_{j\in S\setminus\{i\}} x_{ij} \leq |S| - 1, \quad \text{for all } S \subset N,\ 2 \leq |S| \leq n-1,$$

$$x_{ij} \in \{0, 1\}, \qquad \text{for all city pairs } (i, j).$$

Requires high complexity!

Representation for TSP

- Representation is an ordered list of city numbers:

  1. State College
  2. Pittsburgh
  3. Salt Lake City
  4. Seattle
  5. LA
  6. Miami
  7. DC
  8. NYC

- Possible city lists as candidate solutions:
  - CityList1 (3 5 7 2 1 6 4 8)
  - CityList2 (2 5 7 6 8 1 3 4)

Crossover for TSP

Parent1	(3 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)

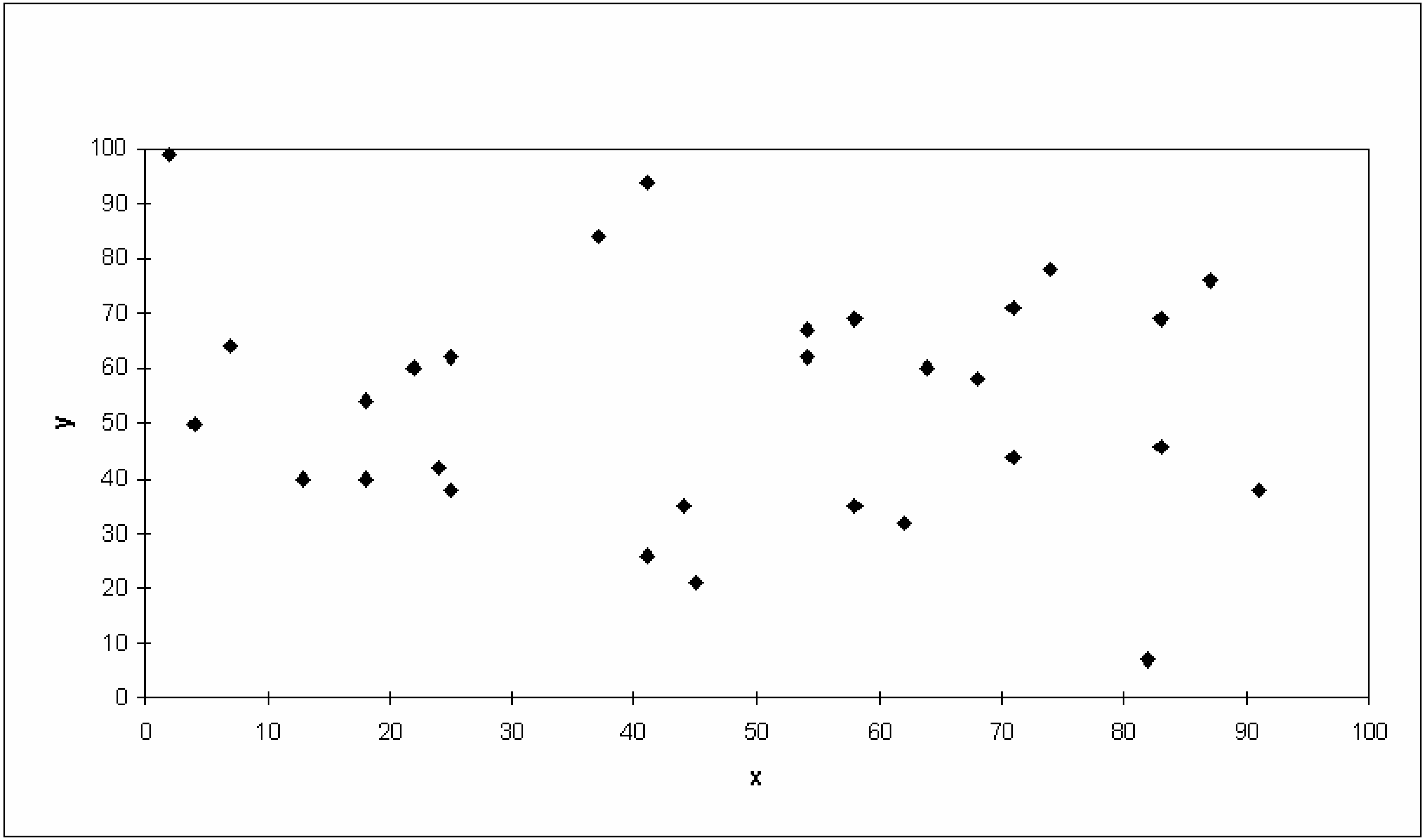
Mutation for TSP

Mutation involves reordering of the list:

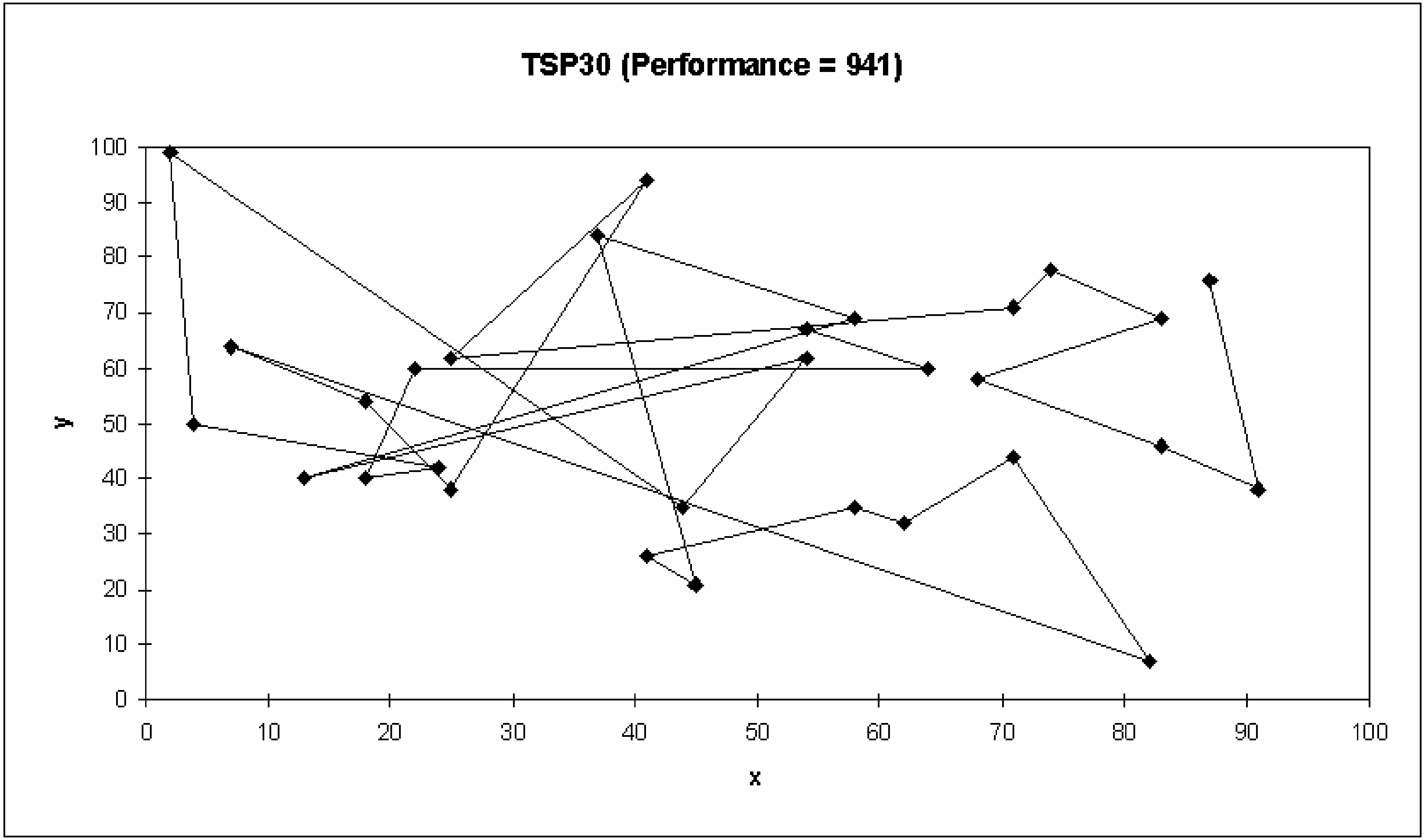
Before: (5 8 7 2 1 6 3 4)

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

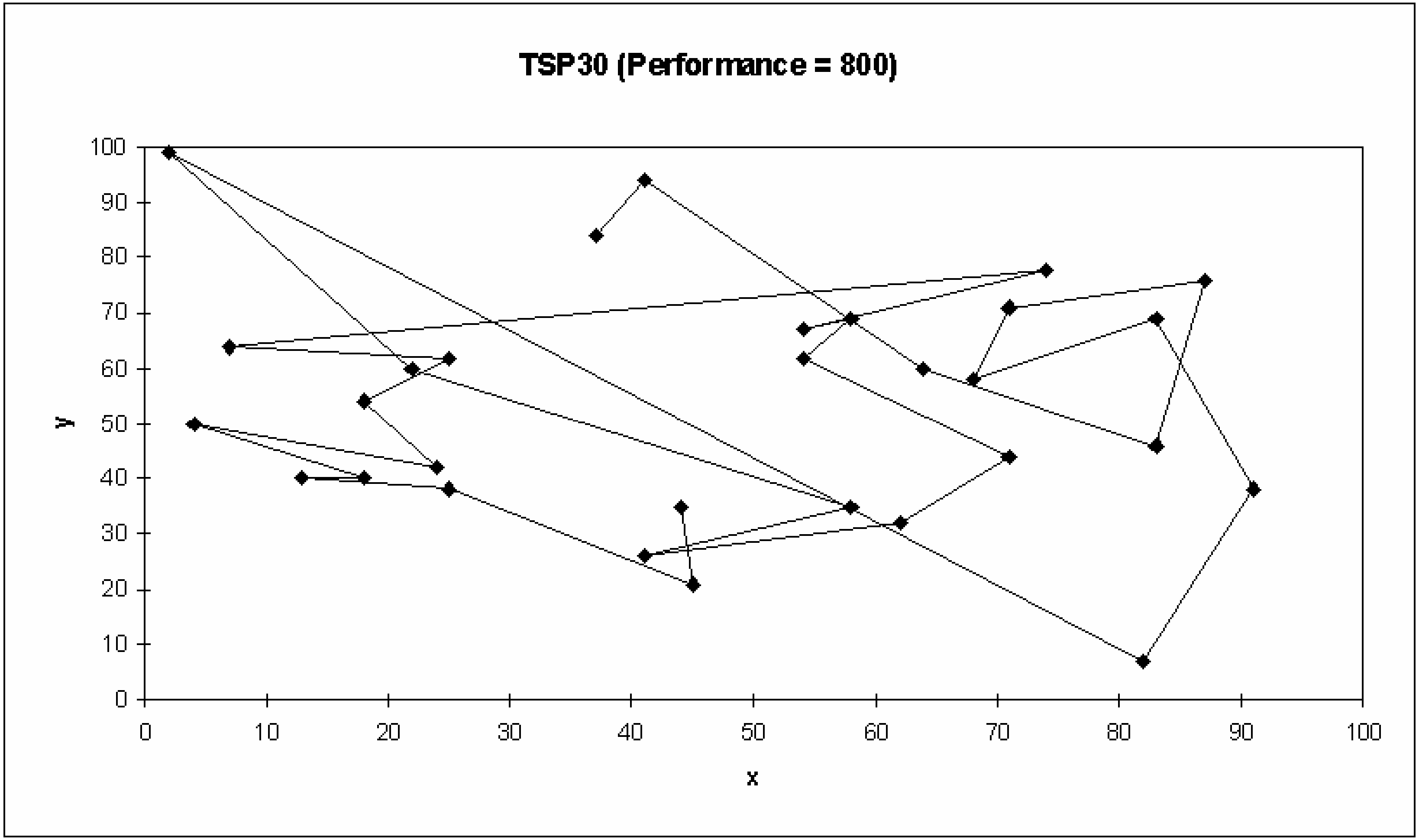
TSP Example: 30 Cities



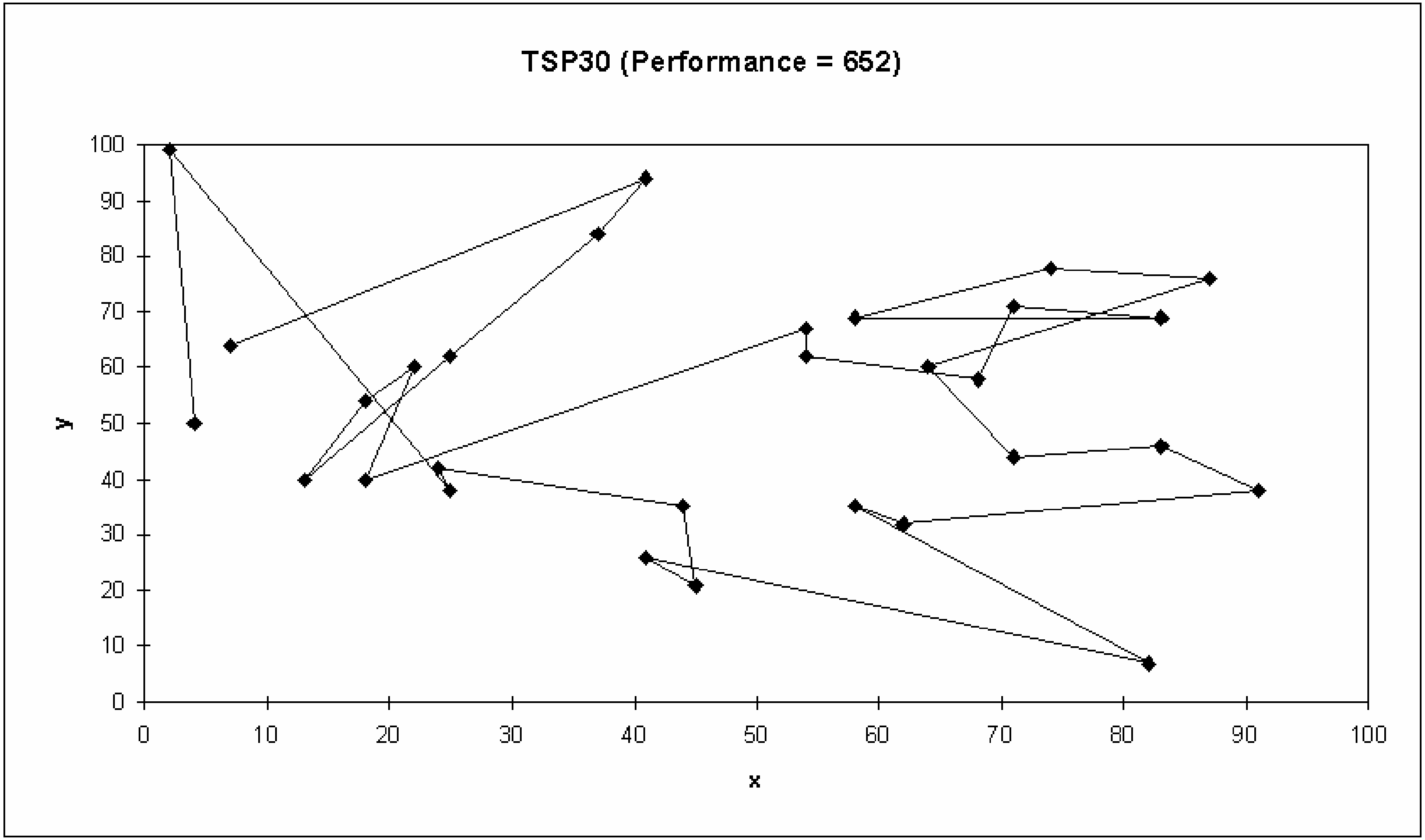
Solution i (Distance = 941)



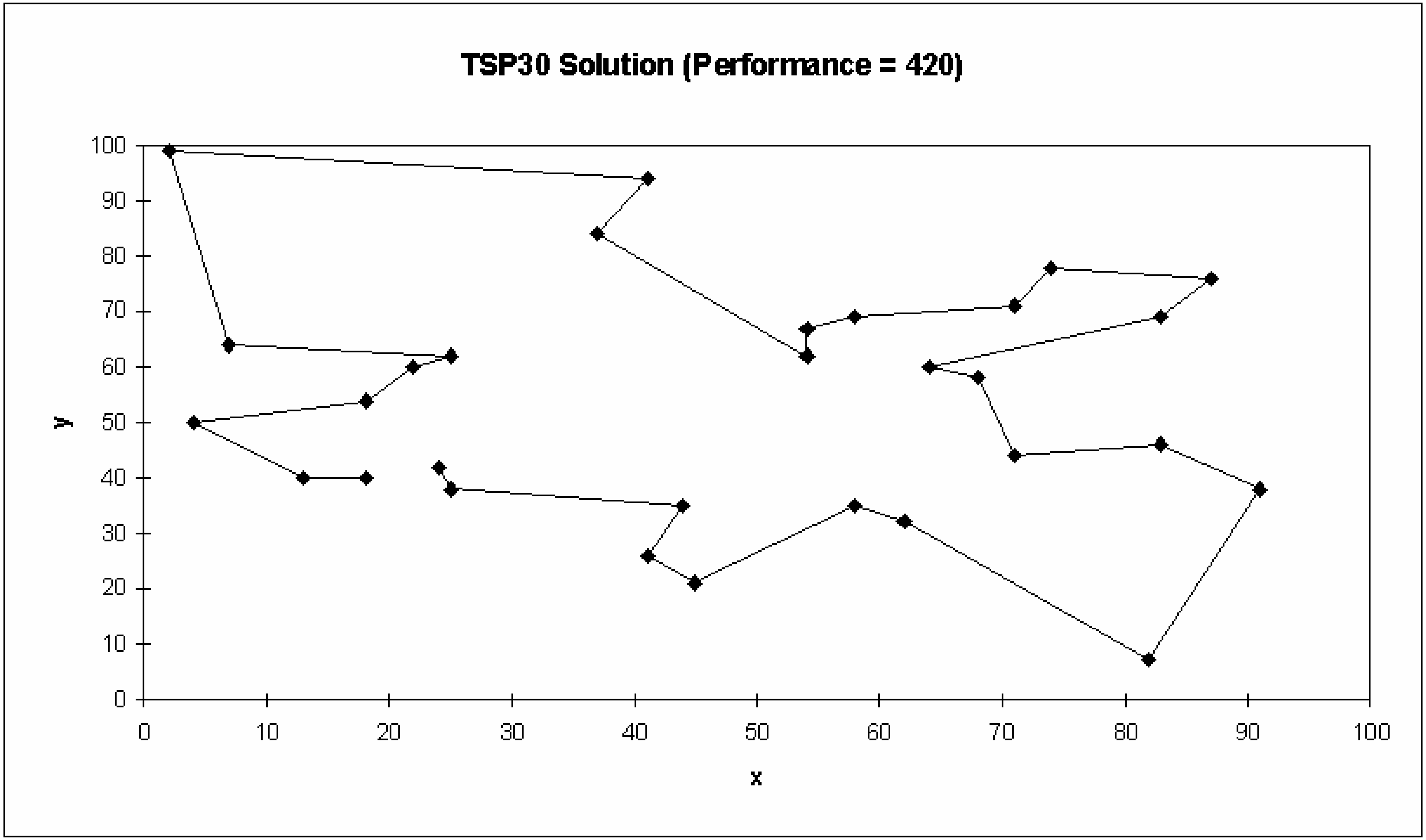
Solution j (Distance = 800)



Solution k (Distance = 652)



Best Solution (Distance = 420)



Acknowledgement

본 캠프는 한동대학교 공학교육혁신센터의
SW 단기교육과정 개발사업비로 개최되었습니다.

Hicee.handong.edu