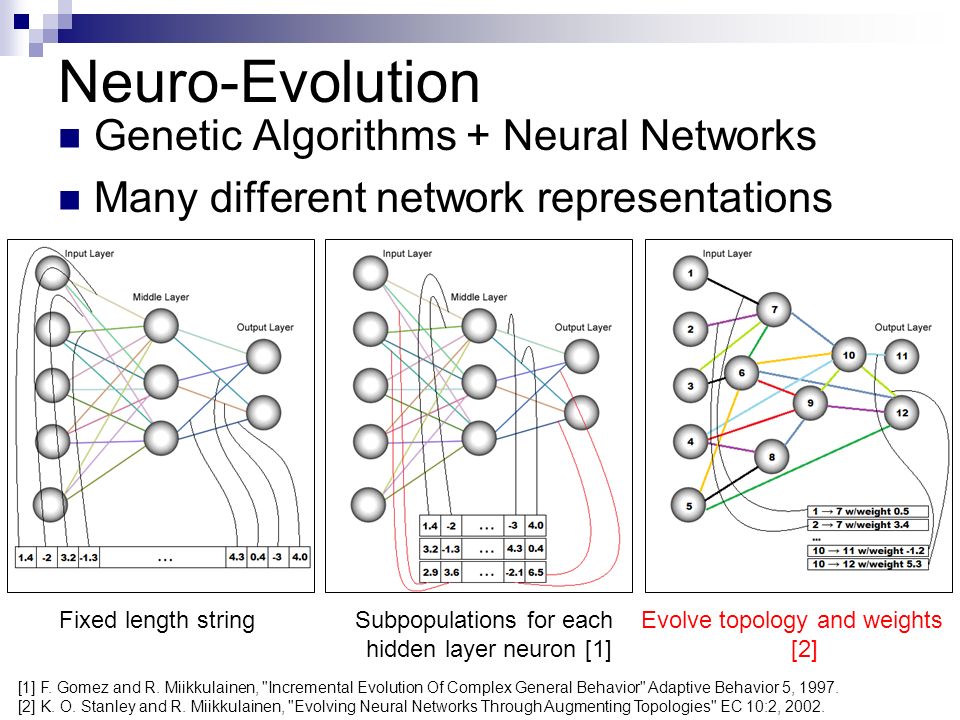
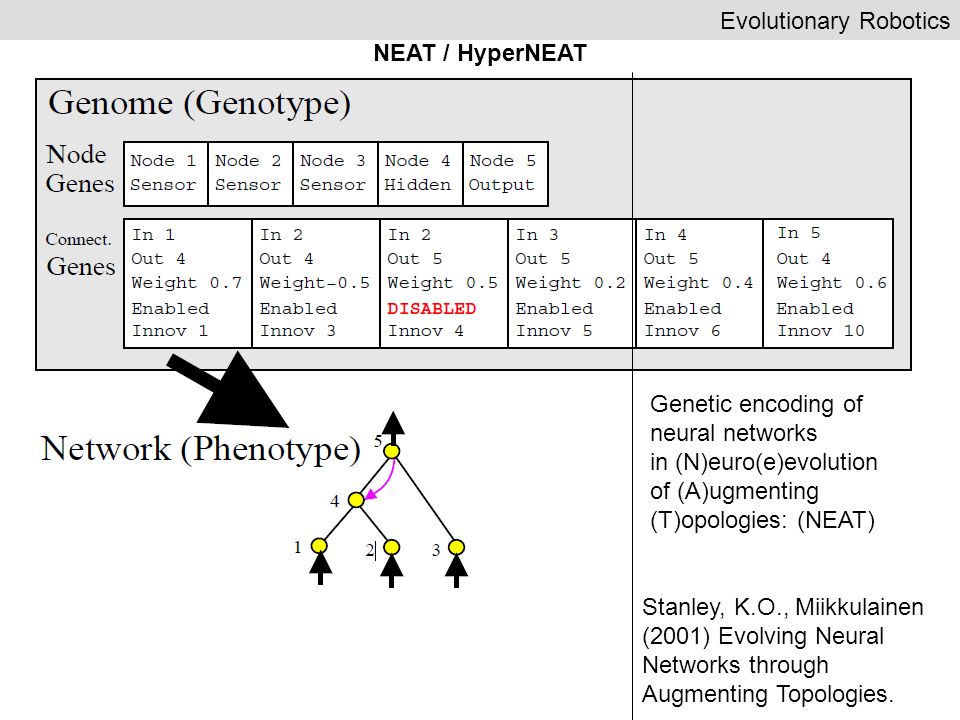
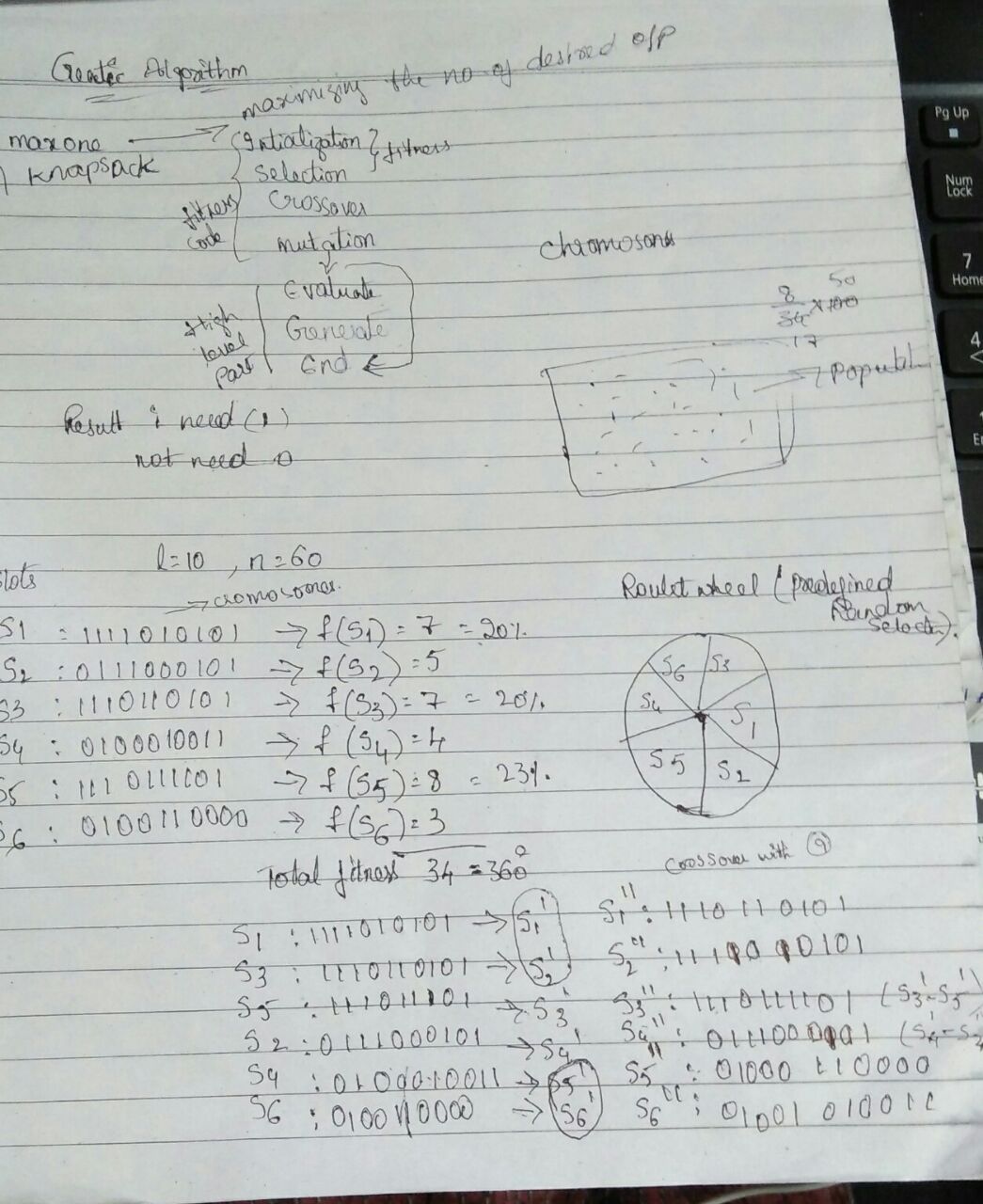
**Neural Evolution**

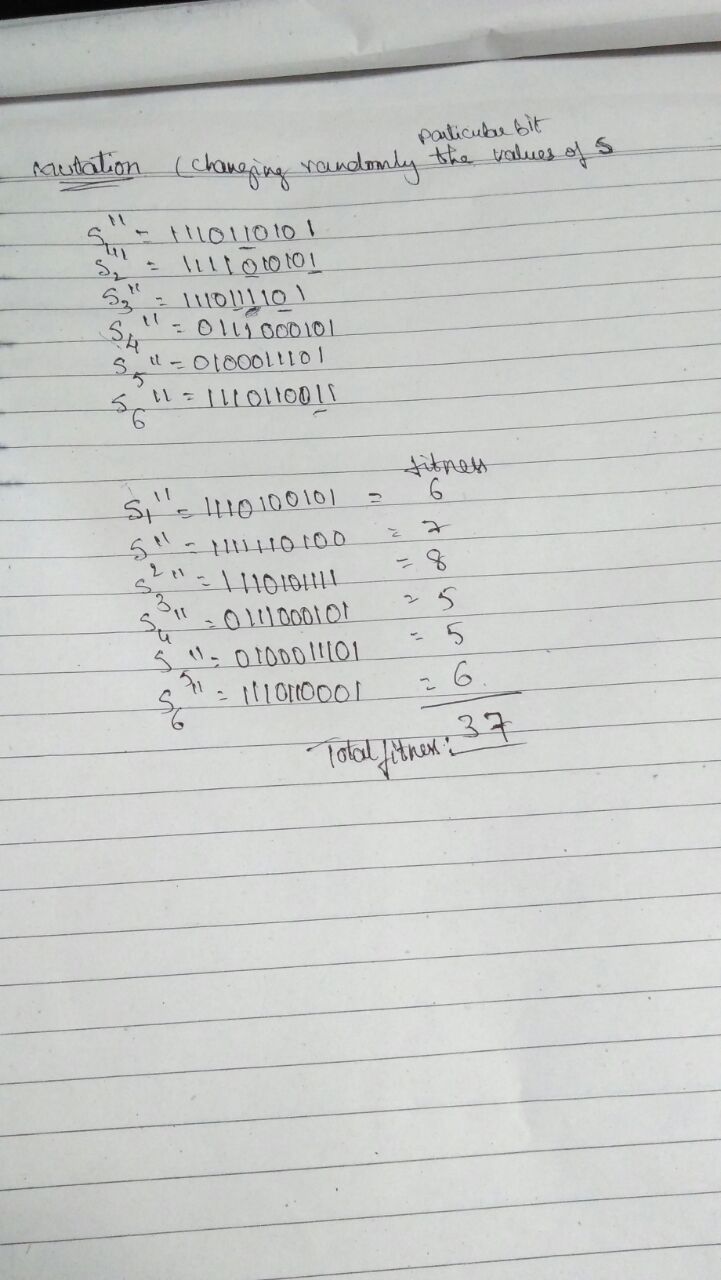
**Short note about Neural Evolution**:

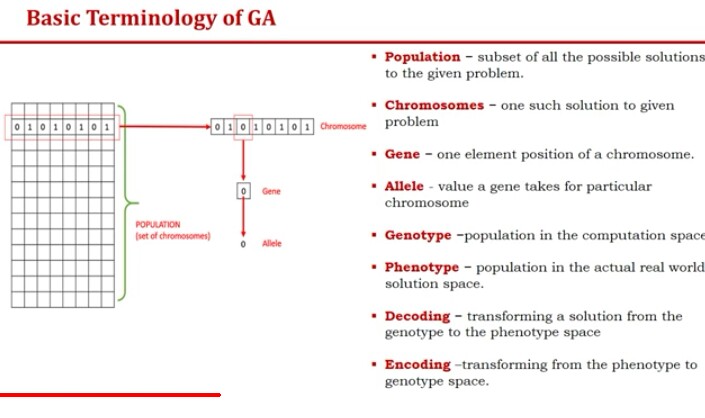
*Neuroevolution, or neuro-evolution, is a form of artificial intelligence that uses evolutionary algorithms to generate artificial neural networks (ANN), parameters, topology and rules. It is most commonly applied in artificial life, general game playing and evolutionary robotics*.

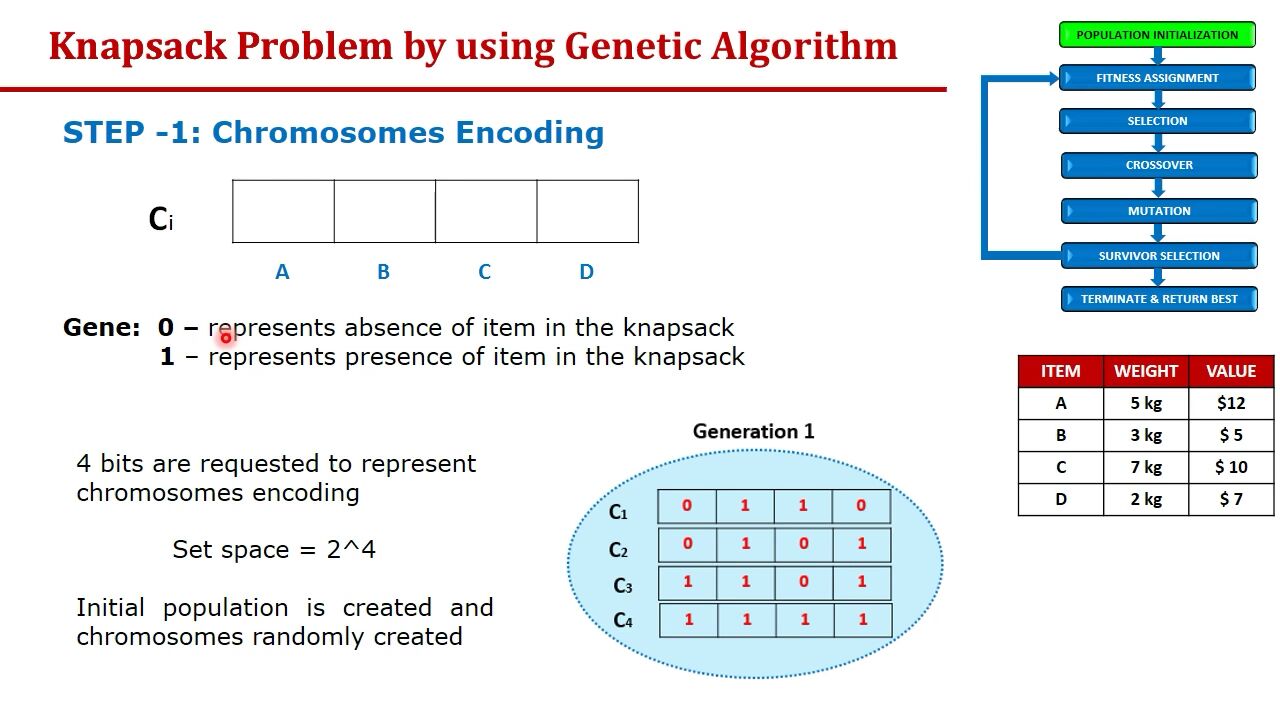


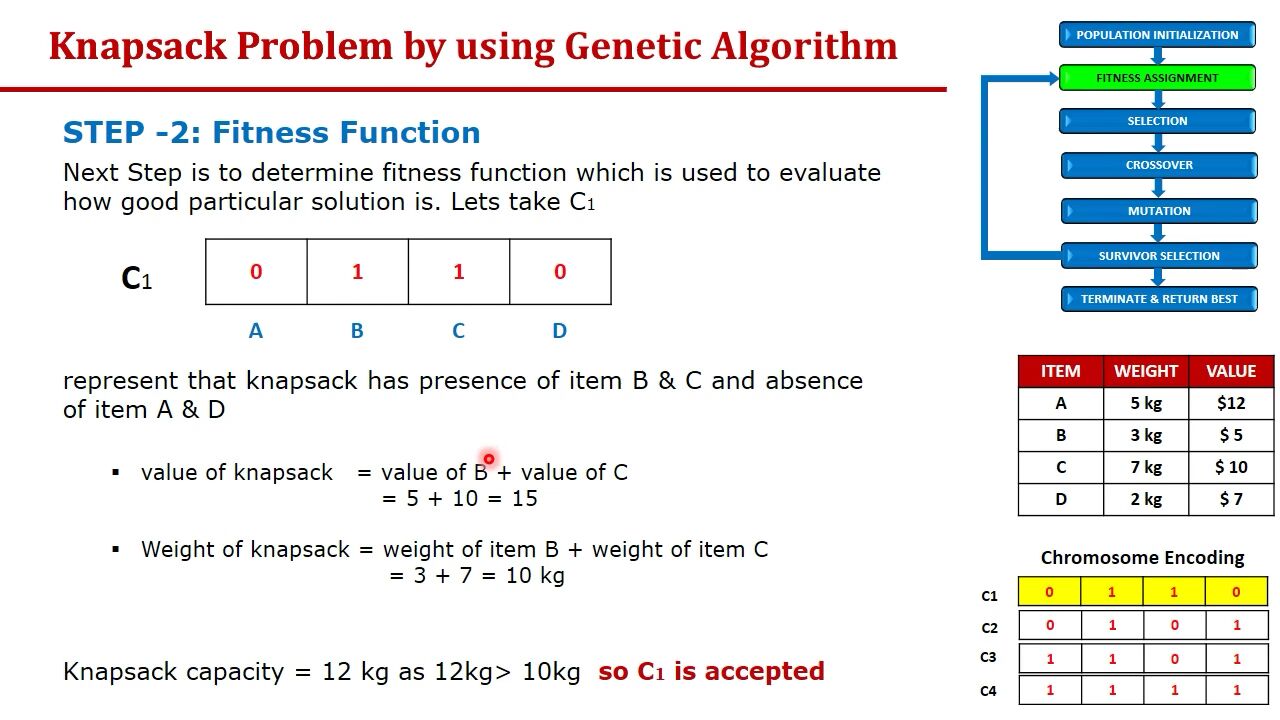


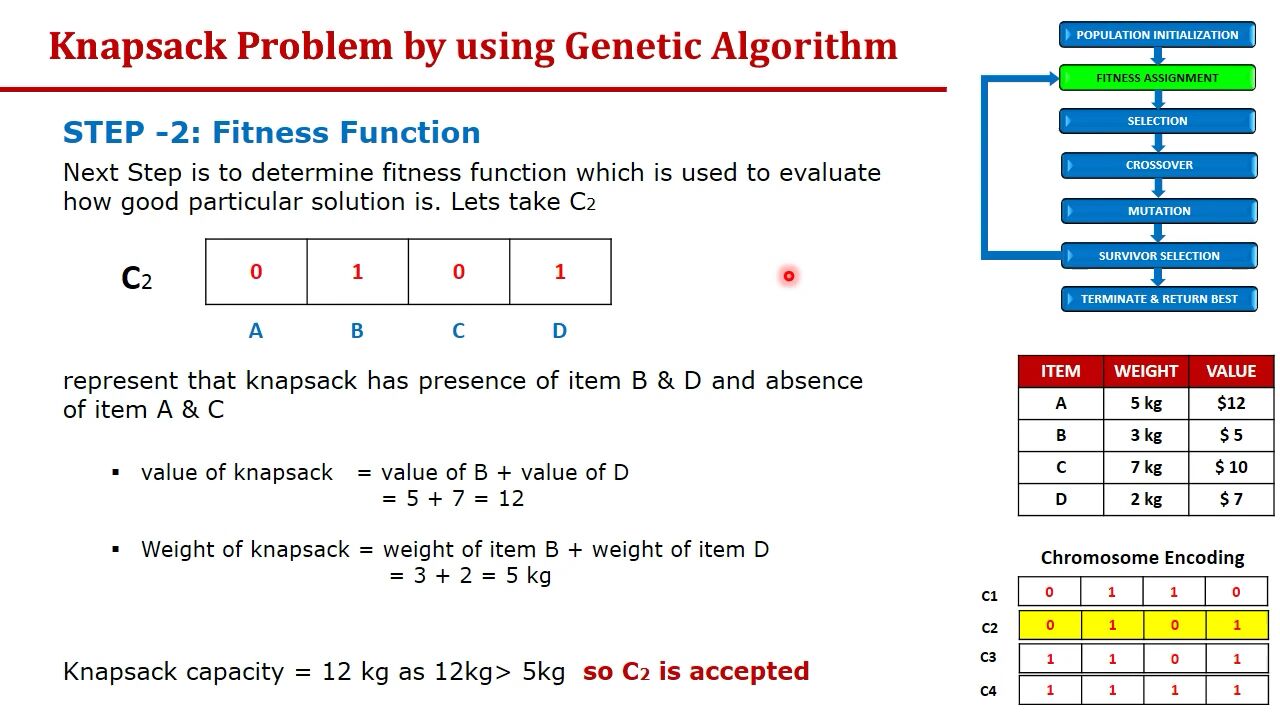


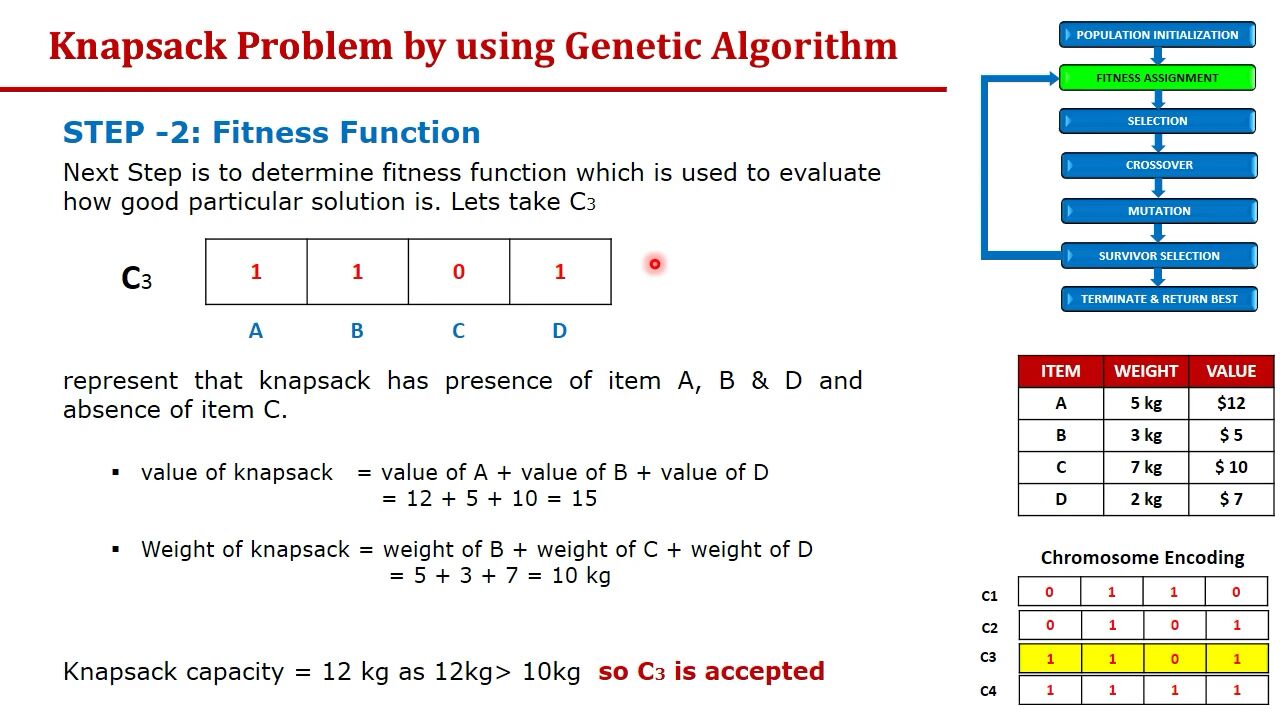


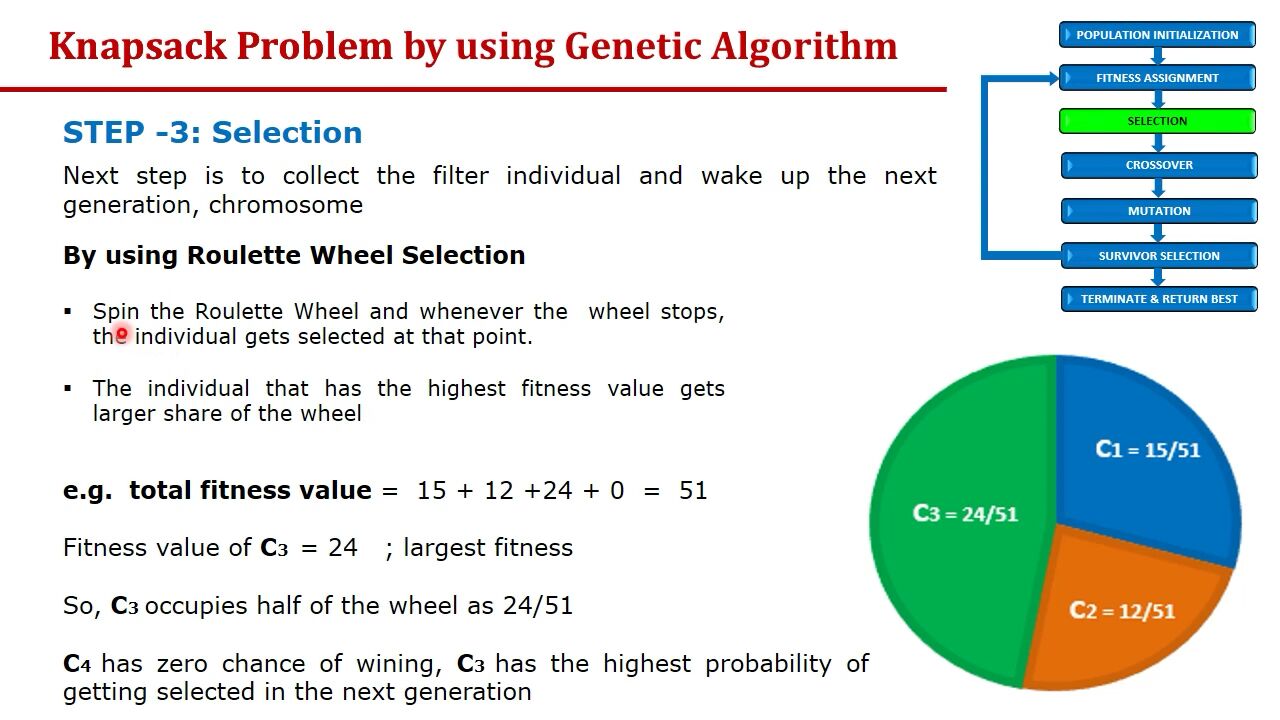


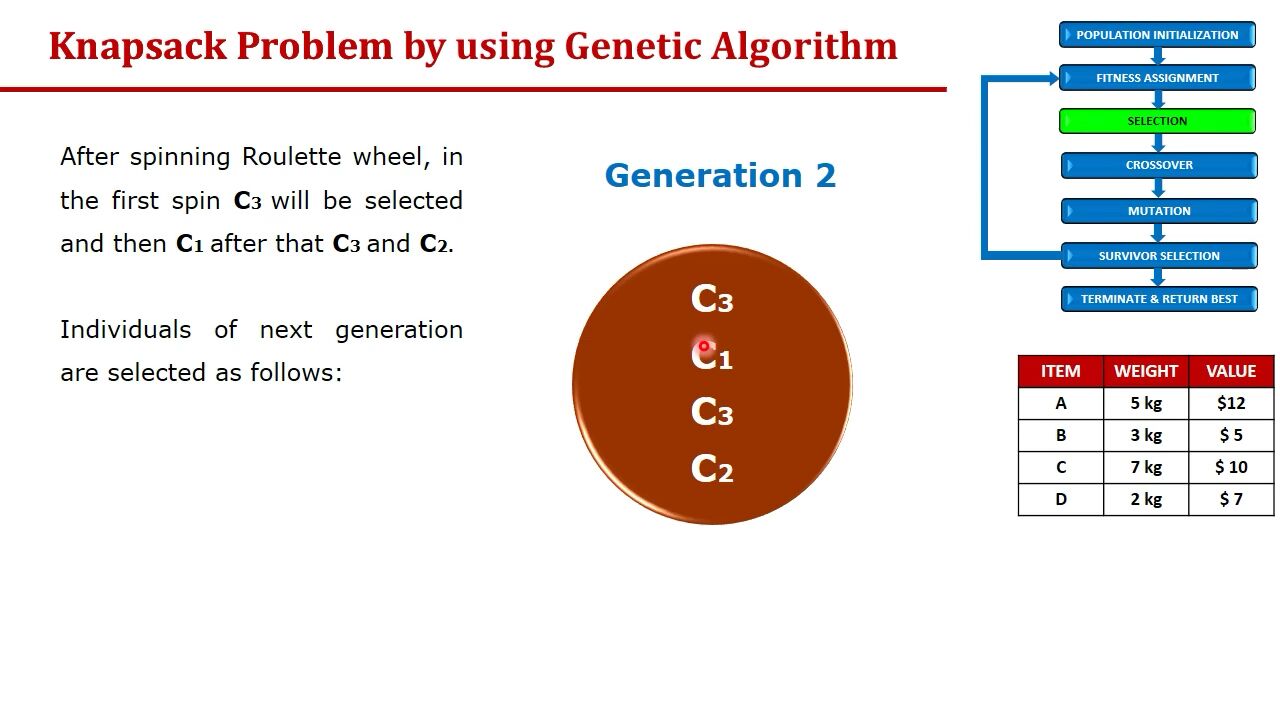


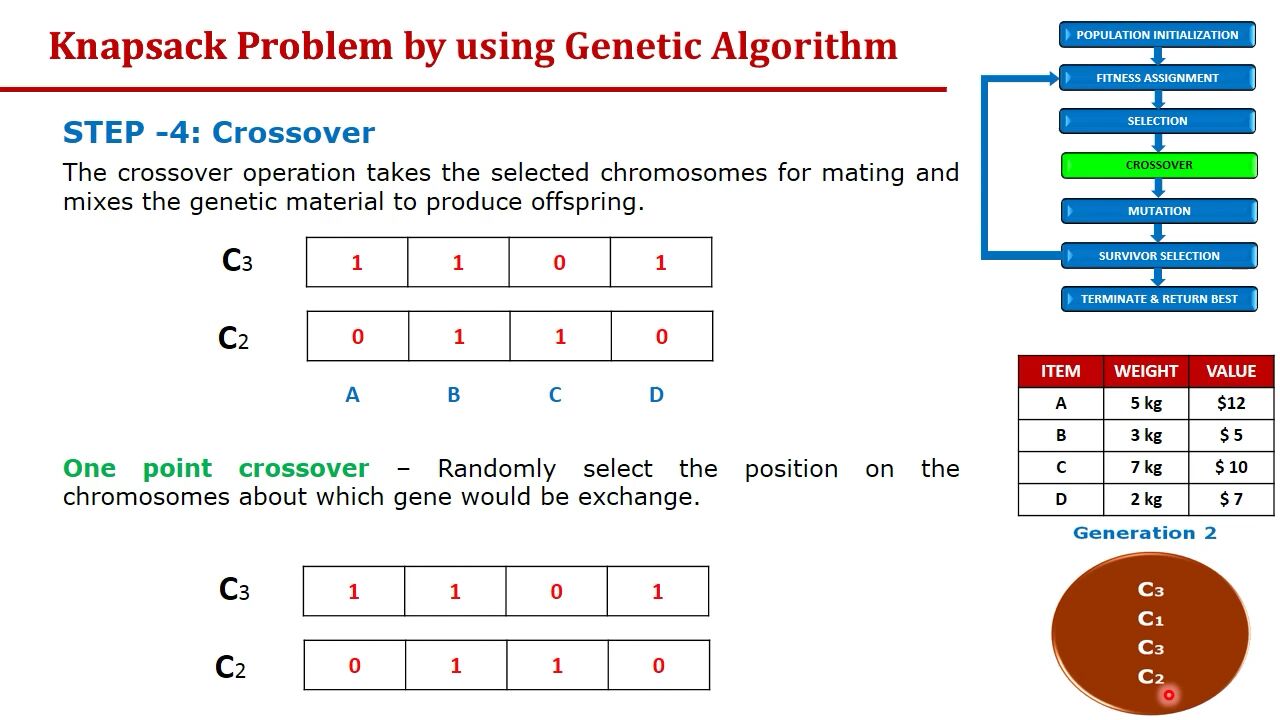


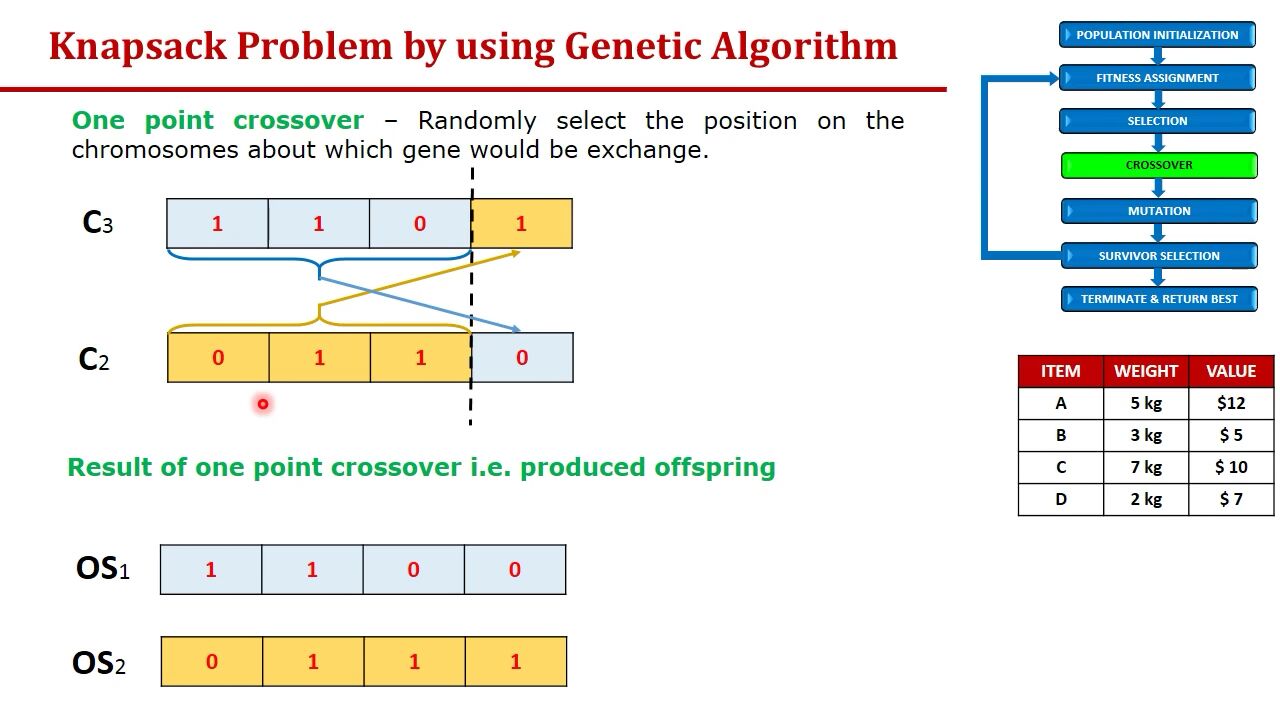


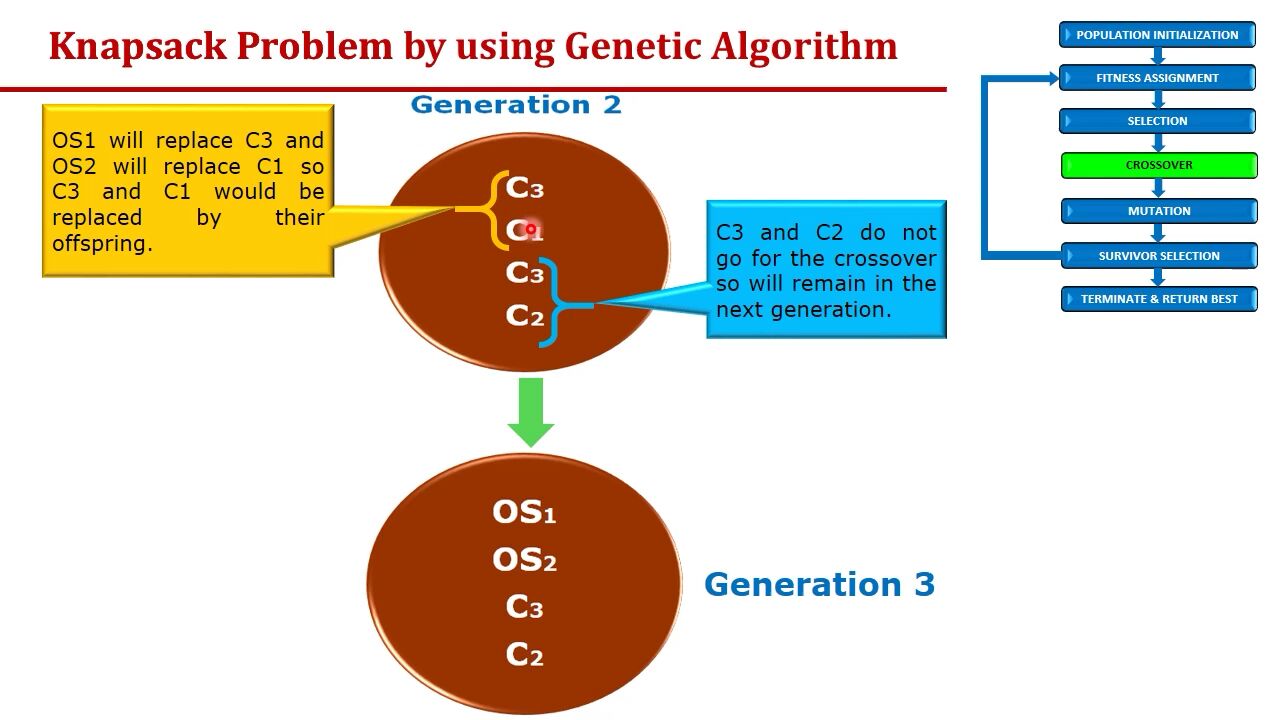


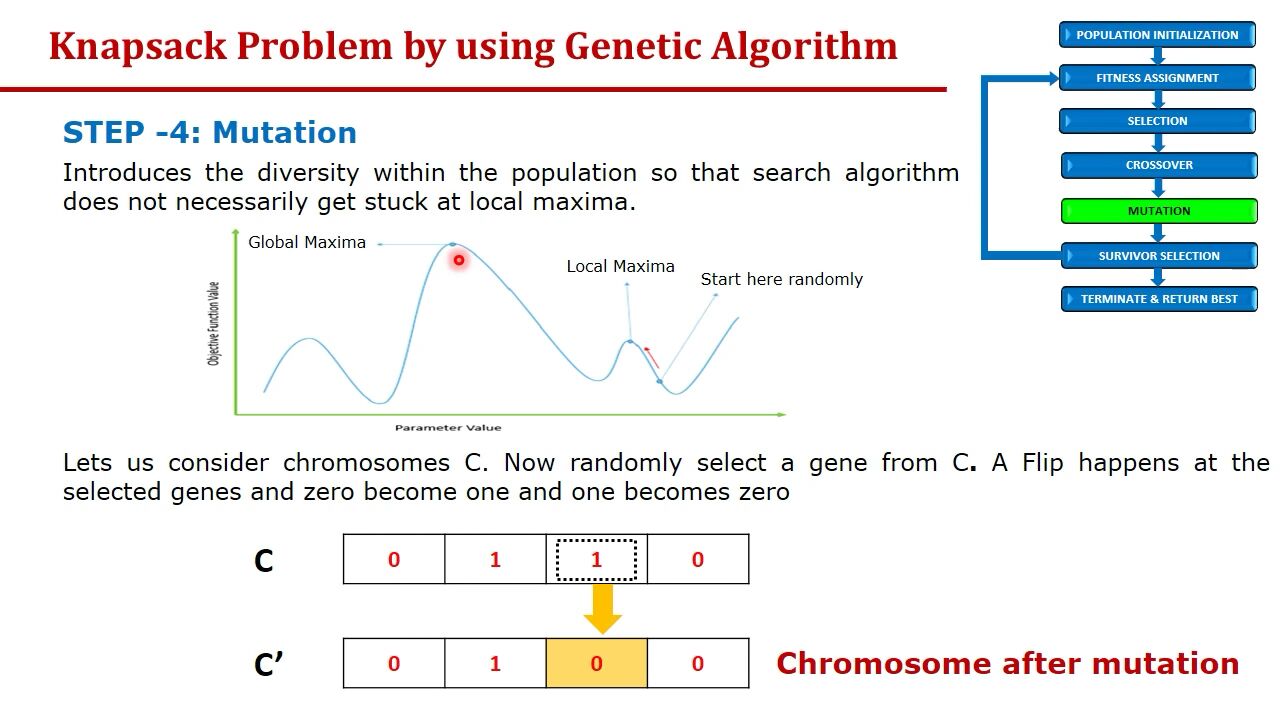


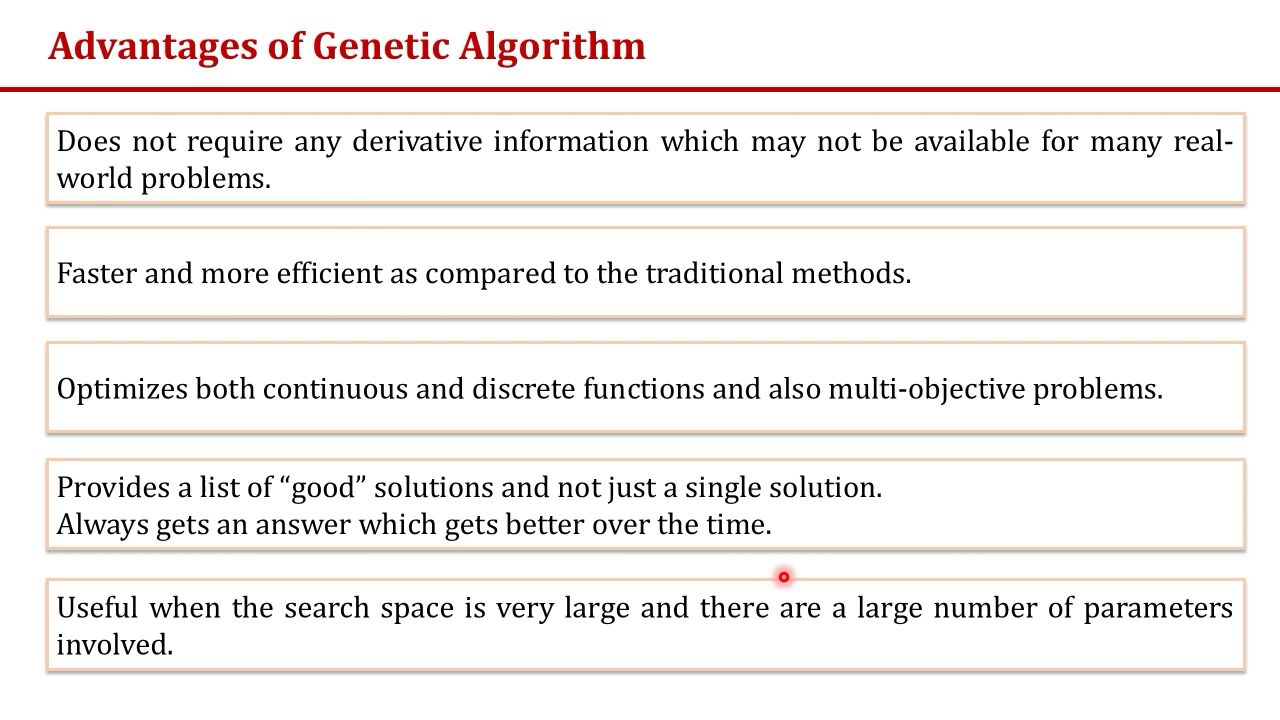












**Use Cases-**

* Market Price and cost modeling.
* Customer Segmentation.
* Insurance claim and fraud Detection.
* Elgibility to apply for credit card
* Chances of getting selected in Interview
* Chances of apply use cases without error.
* To mention the Population Growth.
* To check the Credit Risk in banks.
* Finding the transactions happened in ATM per day.
* Finding the changes in Pollution
* To check the existence of the customer in the bank.
* To check the market strategy.
* To find the price fluctuation of Amazon products.

Python-

import numpy as np

import matplotlib.pyplot as plt

DNA\_SIZE = 10 # DNA length

POP\_SIZE = 100 # population size

CROSS\_RATE = 0.8 # mating probability (DNA crossover)

MUTATION\_RATE = 0.003 # mutation probability

N\_GENERATIONS = 200

X\_BOUND = [0, 5] # x upper and lower bounds

def F(x): return np.sin(10\*x)\*x + np.cos(2\*x)\*x # to find the maximum of this function

# find non-zero fitness for selection

def get\_fitness(pred): return pred + 1e-3 - np.min(pred)

# convert binary DNA to decimal and normalize it to a range(0, 5)

def translateDNA(pop): return pop.dot(2 \*\* np.arange(DNA\_SIZE)[::-1]) / float(2\*\*DNA\_SIZE-1) \* X\_BOUND[1]

def select(pop, fitness): # nature selection wrt pop's fitness

idx = np.random.choice(np.arange(POP\_SIZE), size=POP\_SIZE, replace=True,

p=fitness/fitness.sum())

return pop[idx]

def crossover(parent, pop): # mating process (genes crossover)

if np.random.rand() < CROSS\_RATE:

i\_ = np.random.randint(0, POP\_SIZE, size=1) # select another individual from pop

cross\_points = np.random.randint(0, 2, size=DNA\_SIZE).astype(np.bool) # choose crossover points

parent[cross\_points] = pop[i\_, cross\_points] # mating and produce one child

return parent

def mutate(child):

for point in range(DNA\_SIZE):

if np.random.rand() < MUTATION\_RATE:

child[point] = 1 if child[point] == 0 else 0

return child

pop = np.random.randint(2, size=(POP\_SIZE, DNA\_SIZE)) # initialize the pop DNA

plt.ion() # something about plotting

x = np.linspace(\*X\_BOUND, 200)

plt.plot(x, F(x))

for \_ in range(N\_GENERATIONS):

F\_values = F(translateDNA(pop)) # compute function value by extracting DNA

# something about plotting

if 'sca' in globals(): sca.remove()

sca = plt.scatter(translateDNA(pop), F\_values, s=200, lw=0, c='red', alpha=0.5); plt.pause(0.05)

# GA part (evolution)

fitness = get\_fitness(F\_values)

print("Most fitted DNA: ", pop[np.argmax(fitness), :])

pop = select(pop, fitness)

pop\_copy = pop.copy()

for parent in pop:

child = crossover(parent, pop\_copy)

child = mutate(child)

parent[:] = child # parent is replaced by its child

plt.ioff(); plt.show()