Natural methods

Mengting Ding

**Summary**

This report aims to compare the pros and cons of applying Simulated Annealing and Genetic Algorithm on bounded knapsack problem. A bounded knapsack problem is an optimization problem with maximizing the value of the items in a bag as the objective function and subjected to the weight limitation.

Simulated annealing and genetic algorithm both are widely used algorithms in solving optimisation problems. In general, simulated annealing excels at giving the global optimal solution though when encountering problems that are complicated it is time consuming. Genetic algorithm is faster than simulated annealing when presented to a more complicated problem, however, due to the randomness the solution might not be the global optimal solution.

**Introduction**

From Wikipedia page of knapsack problem, the objective function and constraints of the bounded knapsack problem are shown below:

Maximize

Subject to (*Knapsack problem* 2021)

**Simulated Annealing and Genetic Algorithms**

*Simulated Annealing*

Variables:

* weight variable is a list of weights of each kind of item
* value variable is a list of value of each kind of item
* num\_item is a list containing the number of each item
* knapsack\_threshold variable is a number indicating the maximum weight
* current\_solution is a list type variable containing the initial solution
* T is the variable for initial temperature
* Terminate\_condition variable

def function SA(weight, value, knapsack\_threshold, T):

update T

randomly generate num\_item

if num\_item \* weight <= knapsack\_threshold:

new\_solution = num\_item \* value

while not Terminate\_condition:

if new\_solution > current\_solution:

current\_solution = new\_solution

else:

if met a smaller probability constraint:

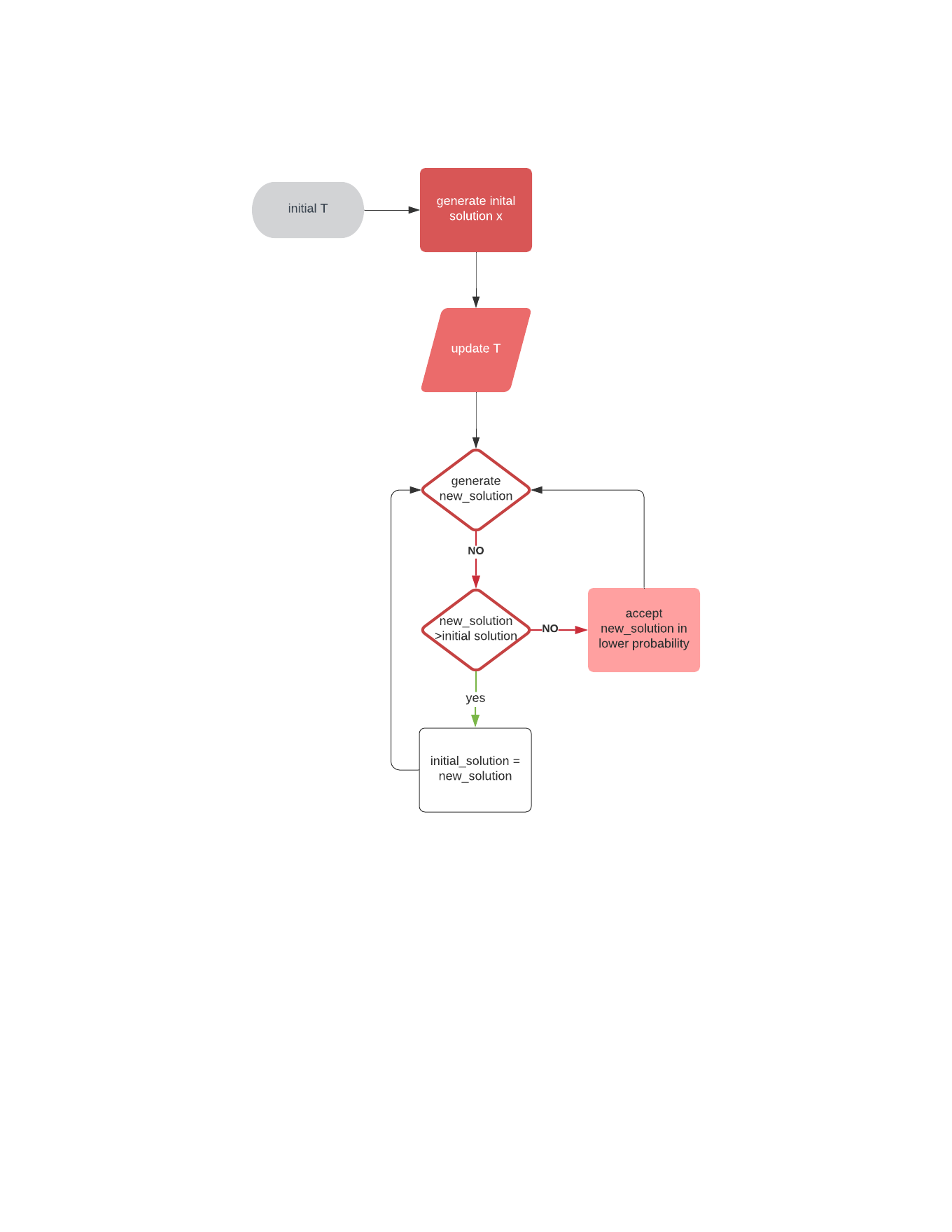
current\_solution = new\_solution

endif

endif

end while

Simulated Annealing flow chat



*Genetic Algorithms*

Variables

* weight variable is a list of weights of each kind of item
* value variable is a list of value of each kind of item
* num\_item is a list containing the number of each item
* knapsack\_threshold variable is a number indicating the maximum weight

def generation():

initial\_population = np.random.randint(n,size=pop\_size)

return initial\_population

def fitness(weight, value, population, knapsack\_threshold):

fit = np.empty(population.shape[0])

obj = popluation \* value

constr = population \* weight

if constr <= knapsack\_threshold:

fit = obj

endif

return fit

def selection(fitness, num\_parents, population):

parents = np.empty((num\_parents, population.shape[1]))

for i in range(num\_parents):

max\_fitness\_idx = np.where(fitness == np.max(fitness))

parents[i,:] = population[max\_fitness\_idx[0][0],:]

fitness[max\_fitness\_idx[0][0]] = -999999

return parents

def crossover(parents, num\_offsprings):

offsprings = np.empty((num\_offsprings,parents.shape[1]))

while parents.shape[0] < num\_offsprings:

offsprings[i,0:crossover\_point]=parents[parents1index, 0:crossover\_point]

offsprings[i,crossover\_point:] = parents[parents2index ,crossover\_point]

end while

return offsprings

def mutation(offsprings,mutation\_rate):

for index in range(len(offsprings)):

if random.random() < mutation\_rate):

swap\_index = int(random.random()\*len(offsprings))

swap1 = offsprings[index]

swap2 = offsprings[swap\_index]

offsprings[index] = swap2

offsprings[swap\_index] = swap1

return offsprings

def GA(weight,value,population,pop\_size,num\_generations, knapsack\_threshold):

for i in range(num\_generations):

fit = fitness(weight, value, population, knapsack\_threshold)

parents = selection(fit, num\_parents, population)

offsprings = crossover(parents,num\_offsprings)

mutants = mutation(offsprings)

population[0:parents.shape[0],:]=parents

population[parents.shape[0]:,:]=mutants

fitness\_last\_gen = fitness(weight,value, population, knapsack\_threshold)

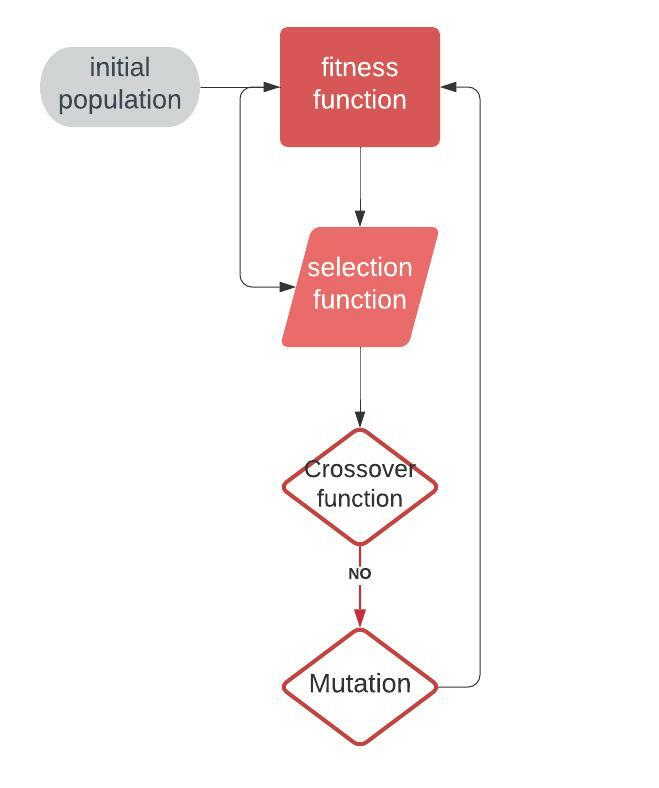
max\_fitness = np.where(fitness\_last\_gen==np.max(fitness\_last\_gen)

parameters.append(population[max\_fitness[0][0],:])

return parameters

(Tiwari, 2019)

Genetic Algorithm flow chat



**Conclusion**

It is hard to choose the functions of each step in Genetic algorithm, it does not guarantee the global optimal solution compared to simulated annealing.

Though simulated annealing is able to give out the best solution,

**Recommendation**

If algorithm users only want to have a good enough solution for their problem, it is appropriate to use genetic algorithm. However, if they want to have the best solution possible it is the best to use simulated annealing though it might take longer time and need higher computational power.

**Appendices**

Reference

Liang, F. (2020, April 21). *Optimization techniques - simulated annealing*. Medium. Retrieved February 8, 2022, from https://towardsdatascience.com/optimization-techniques-simulated-annealing-d6a4785a1de7

McCaffrey, J. D. (2021, December 16). *Knapsack problem using simulated annealing example*. James D. McCaffrey. Retrieved February 8, 2022, from https://jamesmccaffrey.wordpress.com/2021/12/17/knapsack-problem-using-simulated-annealing-example/

Tiwari, S. (2019, April 28). *Genetic algorithm: Part 3 - Knapsack problem*. Medium. Retrieved February 8, 2022, from https://medium.com/koderunners/genetic-algorithm-part-3-knapsack-problem-b59035ddd1d6

Wikimedia Foundation. (2021, December 14). *Knapsack problem*. Wikipedia. Retrieved February 8, 2022, from https://en.wikipedia.org/wiki/Knapsack\_problem