Universitat Autònoma de Barcelona

Knapsack Problem
An Implementation Using Simulated Annealing and Python

Alfredo Hernández Ruth Kristianingsih

10th February 2018

Contents

1 Introduction

The knapsack problem is a classic and widely studied computational problem in combinatorial optimization. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. Because the knapsack problem is a very general problem in combinatorial optimization, it has applications in almost every field. For instance, in economics, the knapsack problem is analogous to a simple consumption model given a budget constraint. In other words, we choose from a list of objects to buy, each with a certain utility, subject to the budget constraint. One of the most interesting applications is in solving truck loading problem. We have to load a truck with some goods. The values and the weights of these goods are listed here[1], and the truck can carry a maximum weight of 600 Kg. We want to maximize the total value of the taken goods.

values	weights
18	17
11	20
15	15
18	12
15	13
7	20
17	13
15	13
22	16
16	15
18	14
10	15
13	21
15	13
16	20
15	19
12	7
13	11
12	16
11	12
14	14
17	17
12	18
15	16
7	17
16	25
16	27
12	15
16	7
13	18
16	13
12	13
10	24
11	18
13	17
15	11
16	18
15	13
11	16
14	14
21	16
11	21
13	17
10	14
14	18
18	16
17	15
14	11
11	23
15	19
–	–

Table 1: Values and Weights of Goods to Optimize

[1] `http://mat.uab.cat/~alseda/MasterOpt/KnapsackData.dat`

2 Theoretical Background

2.1 Knapsack Problem

The knapsack problem is a classical 0–1 combinatorial optimization problem that can be applied to various fields such as economics. A set of n items, each item $i(i = 1, \ldots, n)$ has assigned a value (for instance price) v_i and an item weight value w_i for each item $i(i = 1, \ldots, n)$. The problem is to identify a subset of all items that leads to the highest total profit and does not exceed the resource upper bound W . Then, the knapsack can be formulated as:

$$\text{maximize } f = \sum_{i=1}^{n} v_i x_i, \text{ where } x_i = \begin{cases} 1 & \text{if the item is in the bag} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

subject to the constraints

$$\sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\} \tag{2}$$

The variable x_i is an indicator of item i. If x_i is set to 1, it means item i is selected, or 0 means item i is not selected for $i = 1, \ldots, n$. (1) represents the total value or profit of selection items and (2) constraint.

2.2 Simulated Annealing

Simulated annealing is so named because of its analogy to the process of physical annealing with solids, in which a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration (i.e., its minimum lattice energy state), and thus is free of crystal defects. If the cooling schedule is sufficiently slow, the final configuration results in a solid with such superior structural integrity. Simulated annealing establishes the connection between this type of thermodynamic behavior and the search for global minimum for a discrete optimization problem. Furthermore, it provides an algorithmic means for exploiting such a connection.

At each iteration of a simulated annealing algorithm applied to a discrete optimization problem, the objective function generates values for two solutions (the current solution and a newly selected solution) are compared. Improving solutions are always accepted, while a fraction of non-improving (inferior) solutions are accepted in the hope of escaping local optima in search of global optima. The probability of accepting non-improving solutions depends on a temperature parameter, which is typically non-increasing with each iteration of the algorithm.

Steps of the Algorithm

The usual steps for Simulated Annealing are presented below:

(i) *Initialize*: start with a random initial placement. Initialize a very high "temperature".

(ii) *Move*: perturb the placement through a defined move.

(iii) *Calculate Score*: Calculate the change in the score due to the move made.

(iv) *Choose*: depending on the change in score, accept or reject the move, where probability of acceptance depends on the current "temperature". For this point we will use the Classical Metropolis Method.

(v) *Update and Repeat*: update the temperature value by lowering the temperature following a cooling schedule. Go back to (ii).

The process is done when the "Freezing Point" is reached.

Acceptance Rule

In thermodynamical systems, it is usual to use the Classical Metropolis Method to compute the probability of changing from one state to another [1]. In simple words, the method consists on doing a random change in the system and accept the change in accordance to the following acceptance rule:

$$\operatorname{acc}(\vec{x} \to \vec{y}) = \begin{cases} e^{-\beta[H(\vec{y})-H(\vec{x})]} & H(\vec{y}) > H(\vec{x}) \\ 1 & H(\vec{y}) < H(\vec{x}) \end{cases} \tag{3}$$

where $H(\vec{x})$ is the energy of the state $\vec{x}$, $\beta = 1/(k_B T)$, and T is the temperature.

Notice that this has the same physical interpretation we would expect in a thermodynamical system:

- Limit $T \to 0$ ($\beta \to \infty$): the system only accepts changes that minimize the energy:

$$\operatorname{acc}(\vec{x} \to \vec{y}) = \begin{cases} 0 & H(\vec{y}) > H(\vec{x}) \\ 1 & H(\vec{y}) < H(\vec{x}) \end{cases}$$

- Límit $T \to \infty$ ($\beta \to 0$): the system accepts all changes, which maximizes the entropy.

In our problem, the *energy* can be interpreted as follows:

$$H(\vec{x}) = \sum_{i=1}^{n} v_i x_i \tag{4}$$

Cooling Schedule

In the process of annealing a metal, if the heating temperature is sufficiently high to ensure random state and the cooling process is slow enough to ensure thermal equilibrium, then the atoms will place themselves in a pattern that corresponds to the global energy minimum of a perfect crystal.

For this reason, we need to choose very carefully our Cooling Schedule. We have decided to use a curve that decreases slowly the temperature until the particles arrange themselves in the ground state of the solid. We use the following formula:

$$T_i = \frac{T_0 \cdot \tanh(-\ln(i) + 10) + 1)}{2} \tag{5}$$

The idea behind it is to let the algorithm quickly discard bad configurations in the very beginning (high temperature), and explore different configurations close to the optimal solution in the final steps (low temperatures). The plot of the our cooling schedule can be seen in fig. 1.

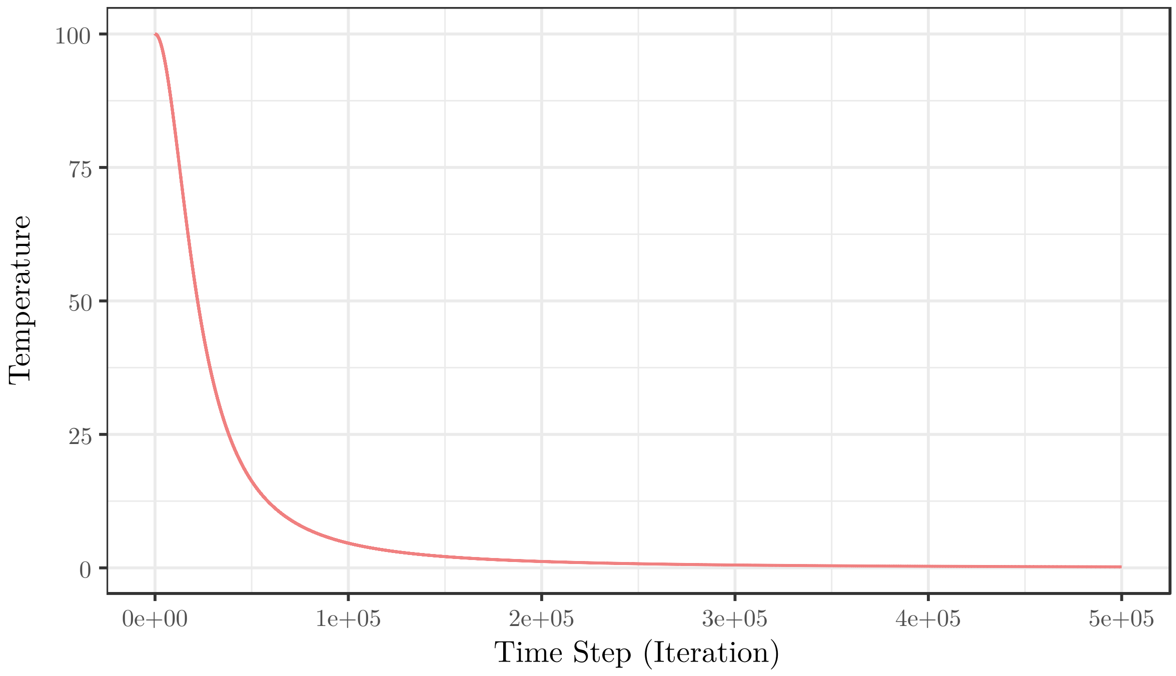

Figure 1: Evolution of the Temperature Following a Cooling Schedule

3 Implementation of the Program

3.1 Preprocessing the Data

Since the data provided includes a header, we decided the get rid of it using AWK, and change the format to store it in a `.csv` file:

```
BEGIN{
  FS = "␣"
  OFS = ","
}
NR > 1 && NF == 2 {
  print $1, $2
}
```

We just run it as usual:

```
awk -f clean_data.awk KnapsackData.dat > data.csv
```

3.2 Algorithm in Python

The algorithm in Python 3.5 is programmed mainly using the NumPy library.

First of all, we use Pandas to read and process the file with the data:

```
data = pd.read_csv('data.dat')
values = np.array(data["values"])
weights = np.array(data["weights"])
num_items = len(data)
```

Two important parameters in the algorithm are the total weight and the total value of the items in the truck. For this reason, we have defined the `get_weight()` and `get_value()` functions:

```
def get_weight(individual, weights):
  return int(np.sum(individual * weights))
```

```
def get_value(individual, values, weights):
  total_weight = get_weight(individual, weights)
  if (total_weight <= 600):
    total_value = np.sum(individual * values)
  else:
    total_value = - pow(total_weight, 100)
  return int(total_value)
```

Where the `individual` is the array $\vec{x}$ array of zeroes and ones used as an indicator of the present items in the truck.

We use a simple Monte Carlo method to mutate the elements x_i of the individual:

```
def random_bool():
  return np.random.randint(2)
```

For the Simulated Annealing we define a function which is just a fully fledged version of the algorithm we described in § 2.2:

```
def simulated_annealing(individual, MAX_ITER):
  for iter in range(0, MAX_ITER):
    # Temperature cooling schedule
    temp[iter] = MAX_TEMP * 0.5 * (tanh(-log(iter+1) + 10) + 1 )

    # Exploring the neighbourhood
    individual_new = np.copy(individual)
    for m in range(0, 5):
      index = np.random.randint(num_items)
      individual_new[index] = random_bool()

    # Total value of items inside the truck
    value_old = get_value(individual, values, weights)
    value_new = get_value(individual_new, values, weights)

    # Acceptance probabilities
    if (value_old < value_new):
      prob[iter] = 1
    else:
      prob[iter] = exp(-(value_old - value_new) / temp[iter])

    # Acceptance rule
    if (float(np.random.random(1)) < prob[iter]):
      individual = np.copy(individual_new)

    # Update total value and weight
    sol_values[iter] = get_value(individual, values, weights)
    sol_weights[iter] = get_weight(individual, weights)
  return individual
```

To run the algorithm, and store the final solution we just call the function we defined above:

```
solution = simulated_annealing(individual, MAX_ITER)
```

To run the algorithm, we just need to run the following:

```
python knapsack_sa.py
```

4 Results and Discussion

In table 2, we can see one optimal solution found using our algorithm. The vector $\vec{x}$ shows the indicator of item i accepted inside the trucks, for instance since the value x_1 is 1, and x_2 is 0, hence item 1 is accepted and item 2 is not accepted inside truck. In table 2 is also shown the total value that we want to maximize. The result of the maximum value is $f = 598$ with the total weight is exactly 600 kg which is the upper bound or maximum weight W of goods which the truck can carry.

Variable	Optimal value

$\vec{x}$	`[1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0`
	`1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1]`

Total Value	598
Total Weight	600

Table 2: Optimal Solution for the Knapsack Problem

Fig. 2 and fig. 3 show the evolution of the total value and total weight of the computed, respectively. As it can be seen, the convergence to the optimal solution is quite fast in the initial iterations of the algorithm ($\lesssim$ 200 000 iterations) and then the slow decreasing of the temperature allows the algorithm to find the optimal combination of items in the solution space. One could say that after iteration 400 000 the system reaches the freezing point; although this is not a problem, since the solution has already been found.

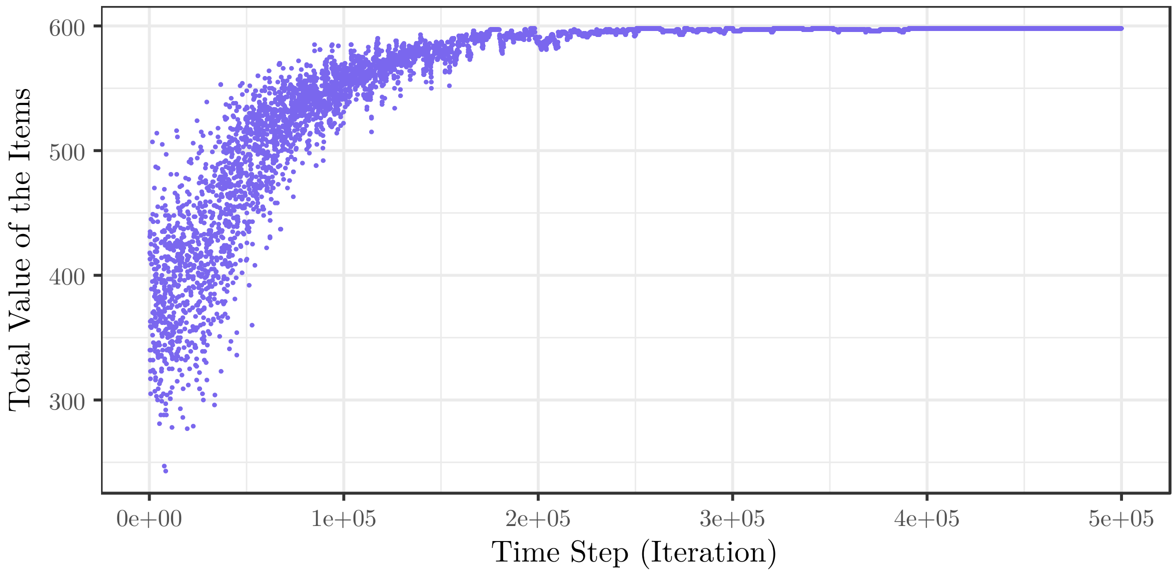

Figure 2: Evolution of the Total Value of the Items in the Truck

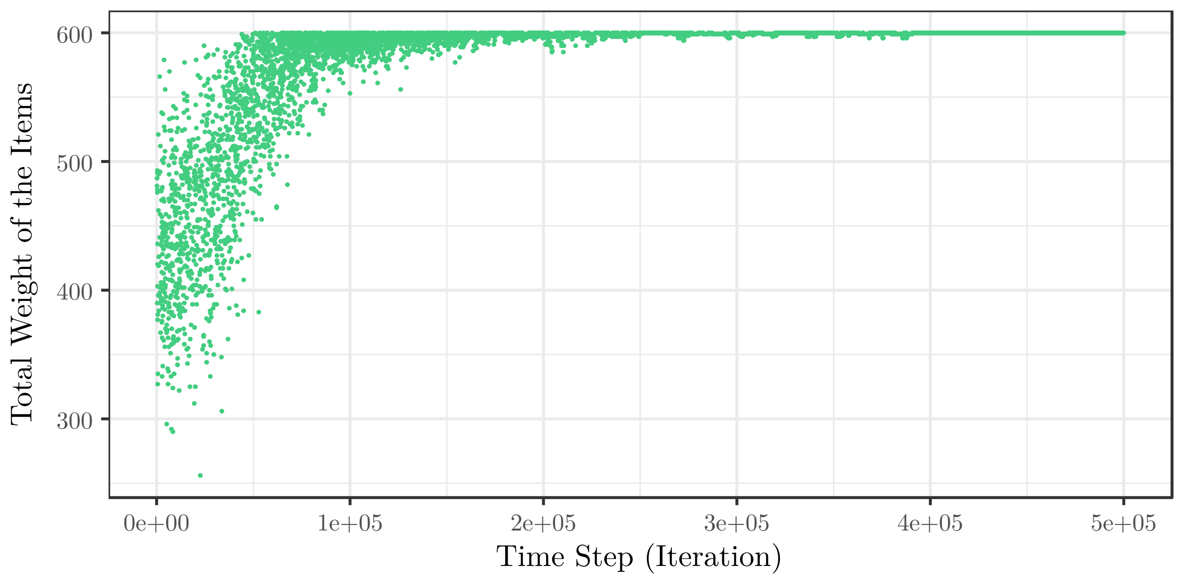

Figure 3: Evolution of the Total Weight of the Items in the Truck

Future Work

Some different experiments, for instance tuning the parameters have been left for the future work due to lack of time. There are some ideas that we would like to try during the implementation of simulated annealing algorithm to solve the knapsack problem. This analysis report mainly focused on maximizing the values with only considering only maximum one item of each good allowed in the truck. For this reason, we propose some ideas to be tested and implemented as part of future work and further improvements as follows:

(i) Solving knapsack problems with subjects to more constraints, for instance solving knapsack problem that allows more than one item to put inside the truck. This problem is so-called multidimensional knapsack problem. [2]

(ii) Tune the parameters of the program, such as the Cooling Schedule, and mainly the process of moving in the solution space.

References

[1] D Frenkel and B Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, 2nd edition, 2002. ISBN: 978-0122673511. DOI: 10.1063/1.881812.

[2] F Qian and R Ding. Simulated annealing for the 0/1 multidimensional knapsack problem. *Numerical Mathematics (English Series)*, 16(10201026):1–7, 2007.

[3] F Bergeret and P Besse. Simulated annealing, weighted simulated annealing and genetic algorithm at work. *Computational Statistics*, 12(4):447–465, 1997. ISSN: 0943-4062.