Solving 1-0 Knapsack Problem using Ant Colony Optimization

Ashwin Padmanabhan

November 1, 2019

1 Ant Colony Optimization for Knapsack problem:

My Python implementation of the knapsack problem is attached with the folder. I have not used any external packages to implement this. I tested this for the instance name: **knapPI_13_50_1000.csv** and **knapPI_11_100_1000.csv**.

Quick code review: The implementation was inspired from the pseudo-code given in [1]. The **transition probability** is given by (probability that an ant selects a given item to place into the knapsack):

$$p_j = \frac{\tau_j^{\alpha} * \mu_j^{\beta}}{\sum_{j \in N_i} \tau_j^{\alpha} * \mu_j^{\beta}} \tag{1}$$

I initialised τ to 10 for each item initially. The μ was defined as:

$$\mu_j = \frac{z_j}{\frac{w_j}{C}} \tag{2}$$

where z_j is the value of objected indexed j, w_j is the weight of object j and C is the Knapsack capacity.
To solve the knapsack problem, I initially used $\alpha = 3$ and $\beta = 2$ (they are weights of importance we give to the pheromone trail and the μ), though I note that I am still arriving at the optimal for fairly a wide range of α and β, this is shown in the contour plot below and the optimal evolution for different values of α and β. N_i is the set of available (feasible) objects (we can place inside the knapsack) at a given stage during the construction of the partial solution. Also, I used 10 ants for each iteration and 20 iterations for each run of the ACO. 20 iterations seems enough to converge to the optimal.

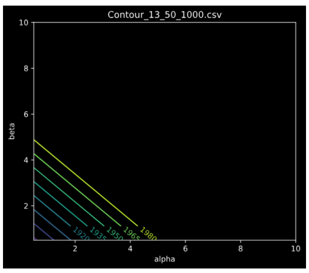
The **pheromone update rule** is:

$$\tau = \tau + \Delta\tau \tag{3}$$

$\Delta\tau$ is defined as:

$$\Delta\tau = \frac{1}{1 + \frac{z_{\text{best}} - z}{z_{\text{best}}}}$$

During **modelling of evaporation** a key parameter is ρ, the rate of evaporation, which is set at 0.2, meaning that the pheromone is reduced by a factor of 0.2 after each iteration with k ants. Also, the pheromone will never drop below 0.05 so that there is always non-zero probability of picking a certain item as long as the knapsack capacity is not violated (see below). To check for how often it converges, I ran the Ant Colony Optimization 30 times (for both instances) and it converges every



(a) Contour plot

```
alpha = 0.5 beta = 0.5 Global Best =   1872
alpha = 0.5 beta = 5.25 Global Best =   1989
alpha = 0.5 beta = 10.0 Global Best =   1989
alpha = 5.25 beta = 0.5 Global Best =   1989
alpha = 5.25 beta = 5.25 Global Best =   1989
alpha = 5.25 beta = 10.0 Global Best =   1989
alpha = 10.0 beta = 0.5 Global Best =   1989
alpha = 10.0 beta = 5.25 Global Best =   1989
alpha = 10.0 beta = 10.0 Global Best =   1989
```

(b) Average Fitness

Figure 1: Contour plots for instance **knapPI_13_50_1000**: (A) Function calls (B) Optimal fitness for different α and β

time. Statistics shown below.

	count	mean	std	min	25%	50%	75%	max
0	30.0	1989.0	0.0	1989.0	1989.0	1989.0	1989.0	1989.0

Figure 2: Summary for instances **knapPI_13_50_1000**

	count	mean	std	min	25%	50%	75%	max
0	30.0	1428.0	0.0	1428.0	1428.0	1428.0	1428.0	1428.0

Figure 3: Summary for instance **knapPI_11_100_1000.csv**

Comparison to Genetic Algorithm: Based on the analysis from two instances, the ACO seems much more robust than the GA. Over n iterations, GA converges lesser number of times than the ACO which converges every time. Also the run for the ACO takes lesser time than the GA for each iterations. But a comparison is difficult based on the fitness calls, since the ACO algorithm does not directly calculate the fitness while constructing the solution.

References

[1] Krzysztof Schiff. Ant colony optimization algorithm for the 0-1 knapsack problem. *Technical Transactions*, 2013.