# Simulated Annealing and the Knapsack Problem

Benjamin Misch

December 19, 2012

## 1 The Knapsack Problem

The knapsack problem is a classic and widely studied computational problem in combinatorial optimization. We are given $n$ objects denoted by $x_i$ $(i = 1, 2, \ldots, n)$ each with corresponding weight $w_i$. We can imagine a person carrying a backpack who has access to each of these objects. Unfortunately, this person can only hold some maximum weight $W$ in his backpack. The problem follows: what is the optimal assortment of objects such that the person maximizes the value of objects in his pack with the constraint of weight W?

The applications of this simple problem can be found in many fields. 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. Because the knapsack problem is a very general problem in combinatorial optimization, it has applications in almost every field. One of the most interesting applications is in solving the so-called Oregon Trail problem.

## 2 Oregon Trail Problem

The Oregon Trail game, a staple of elementary school education in the 1990s, involves a player as a pioneer attempting to travel from Missouri to the West coast in the 1848 while keeping his family safe and alive. The game takes place in two stages. First, the player chooses his load out before heading out onto the Oregon Trail, and second, the player then goes on the trail. The knapsack problem applies to the first part. Given several thousand dollars, the player first chooses a size of wagon. Each wagon can carry a certain amount of weight. From here, in addition the budget constraint, we have a weight constraint. Next, we pick two traveling companions from a large assortment (including a banker, a chef, a fisherman, etc.); each choice is unique with a certain amount of weight, money, and utility value. Then we choose which supplies to take. This process includes food, weapons, medicine, comforts, and more. The problem is to choose from all of these combinations in order to maximize your chance of surviving to the West coast. This is a combinatorial optimization problem subject to multiple constraints and many specific utility functions with regard to each item. Clearly this problem is a very advanced and complicated version of the knapsack problem, but I think it paints a good picture as to how the simple knapsack problem can be extended to something of very high complexity.

# 3 The 0-1 Knapsack Problem

We will focus on the most basic and common version of the knapsack problem. This version limits each object to a number of copies of 0 or 1. We can mathematically formulate the problem as follows:

Let there be $n$ items, $x_1$ to $x_n$ where each $x_i$ has a value $v_i$ and weight $w_i$. The maximum weight that we can carry in the bag is $W$. It is common to assume that all values and weights are greater than zero. Maximize:

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

subject to the constraint

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

Maximize the sum of the values of the items in the knapsack so that the sum of the weights must be less than the knapsack's capacity. We can approach this problem in two ways: a simple deterministic model and a simulated annealing model.

# 4 Algorithm

The algorithm solving the Knapsack Problem is as follows. Imagine you are a thief looting a house. You see several items around the house that you would like to steal, but you can only carry a certain amount of weight or you will be caught running away. You start with zero weight in your bag.

0. Record your current assortment of objects. We will call this $\xi_1$. $\xi = \{x_1, x_2, \ldots, x_n\}, \quad x_i \in 0, 1$

1. You pick an item at random with equal probability.

2. Holding it in your hand, you assess whether to add it to your backpack.

   - If adding this new object into your bag pushes you over the weight limit, you select an item at random with equal probability from those in your bag and the one from your hand. You drop that item. You repeat this process until the objects in your bag are below the weight limit.
   - Otherwise, add the item to your bag.
   - Record the outcome as a trial assortment. We will call this $\xi_1'$.

3. Compare the value of the trial assortment with the original. These values are $V_1'$ and $V_1$, respectively.

$$V(\xi) = \sum_{i=1}^{n} v_i x_i, \qquad i = 1, 2, \ldots, n.$$

   - With some probability P, we accept the new trial assortment as the new assortment (We will analyze two methods for determining this probability). In other words, $\xi_2 = \xi_1'$.
   - Otherwise, we discard the trial assortment and set the original assortment as the new assortment. $\xi_2 = \xi_1$.

**Method 1: The Deterministic Model.** If the difference between the trial and the original is greater than zero, set the trial assortment as the new assortment. In other words, if $V_1' - V_1 > 0$, set P(Accept)=1, else set P(Accept)=0. This model does not converge to any single optimum; rather it alternates between several local optima.

**Method 2: Simulated Annealing Model.** For the theory behind Simulated Annealing, I refer the reader to Homework 10 and 11. In quick review, simulated annealing involves a cooling schedule determined by $\beta$ and a stationary probability $\pi_\beta(\xi) = \frac{1}{Z(\beta)} e^{-\beta V(\xi)}$ . If we cool slowly enough, we can avoid getting trapped in local optima. We determine the probability P by $\min\{1, e^{(\beta\Delta)}\}$ where $\Delta = V_1' - V_1$. $\beta$ is determined more or less by trial and error.

# 5 MATLAB Simulation

Now that we have the theory, we can test it on an example. Let's assume that you have the following objects. Our backpack can only carry a total weight of 20 $kg$.

$$\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{array} \begin{array}{c} \begin{array}{cc} v & w \end{array} \\ \begin{pmatrix} \$50 & 5\ kg \\ \$40 & 4\ kg \\ \$30 & 6\ kg \\ \$50 & 3\ kg \\ \$30 & 2\ kg \\ \$24 & 6\ kg \\ \$36 & 7\ kg \end{pmatrix} \end{array}$$

The code for simulated annealing is as follows. This code can be reappropriated easily for the deterministic model.

```
-----------------------------------------------------------------------------------------
value=[50 40 30 50 30 24 36];
weight=[5 4 6 3 2 6 7];
TotalWeight=20;
beta=0:.01:1;
n=1000;
Knapsack( value, weight, TotalWeight, beta, n)

function X = Knapsack( value, weight, TotalWeight, beta, n )
% Input:  value       =  array of values associated with object i.
%         weight      =  array of weights associated with object i.
%         TotalWeight =  the total weight one can carry in the knapsack.
%         beta        =  vector of beta values for simulated annealing.
%         n           =  number of simulations per beta value.
% Output: FinalValue  =  maximum value of objects in the knapsack.
%         FinalItems  =  list of objects carried in the knapsack.
%                        Entries in the vector correspond to object i
%                        being present in the knapsack.
```

```
    v=length(value);
    W=zeros(1,v);
    Value=0;
    VW=0;
    a=length(beta);
    nn=n*ones(1,a);
    for i=1:a
        b=beta(i);
        for j=2:nn(i)
            m=0;
            while m==0
                c=ceil(rand*v);
                if W(c)==0
                    m=1;
                end
            end
            TrialW=W;
            TrialW(c)=1;
            while sum(TrialW.*weight)>TotalWeight
                e=0;
                while e==0
                    d=ceil(rand*v);
                    if TrialW(d)==1
                        e=1;
                    end
                end
                TrialW(d)=0;
            end
            f=sum(TrialW.*value)-sum(W.*value);
            g=min([1 exp(b*f)]);
            accept=(rand<=g);
            %Deterministic Model
            %if f>=0
            if accept
                W=TrialW;
                VW(j)=sum(W.*value);
            else
                VW(j)=VW(j-1);
            end
        end
        Value=[Value VW(2:length(VW))];
    end
    FinalValue=Value(length(Value))
    x=0;
    for k=1:length(W)
        if W(k)==1
            x=[x k];
        end
    end
    FinalItems=x(2:length(x))
end
```

It is a good exercise to try and solve this problem by hand first. The final solution is $200 with objects 1, 2, 3, 4, 5. The simulated annealing code solved this correctly in every one of my trials, but the deterministic model would sometimes get stuck at $176 with objects 1, 2, 4, 7.

# 6 Comparison of Models

In theory the simulated annealing model should give us the correct optimum far more often than the deterministic model. To test this hypothesis, we can compare the two with a paired t-test. In this case, the null hypothesis is that the difference between the two models is zero. In order to gather the data for the test, we can run each simulation many times and compare the two. I modified my code slightly to run each simulation 100 times. I then catalogued the FinalValue for each model. I then used MATLAB to run the paired t-test.

```
[h,p,ci,stats]=ttest(FVSA,FVDet,left)
```

FVSA corresponds to the final values of the simulated annealing, and FVDet to those of the deterministic model. I received the following output:

```
h =
     1

p =
   1.4834e-09

ci =
    5.2265
    9.6535

stats =
    tstat: 6.6692
       df: 99
       sd: 11.1558
```

This shows that there is a significant difference between the two models. Because this is a one-tailed test, we see that the simulated annealing model is significantly greater than the deterministic model, meaning that the simulated annealing model provides us with a higher value on average.