HOCHSCHULE KONSTANZ
TECHNIK, WIRTSCHAFT
UND GESTALTUNG

Coursework report

Simulated Annealing for the 0/1 Knapsack Problem

School of Computer Science

University of Applied Sciences in Constance

Felix Riehm

Student ID: 287144

E-Mail: F.Riehm@gmx.de

1. Introduction

For the guest lecture „Artificial Intelligence 2” the knapsack 0/1 problem should be solved by implementing a local search or a population algorithm. For solving this problem the dataset “mknapcb1” in the OR Library should be used. After the implementation we had to do performance tests with different parameters. Finally we had to review an existing paper which solves the knapsack problem.

2. The Algorithm

I decided to use the simulated annealing algorithm to solve the problem. Before I started I did a small research and used four different information sources [1] [2] [3] [4] as help. I really liked the solution from Fubin Qian and Rui Ding [1]. So I decided to build-on it. While the base algorithm is still the same I modified some parts of the algorithm. In the following chapters I will explain how I realized the representation of the solution, the evaluation function, the neighborhood function and the acceptance criterion.

2.1. Representation

The representation of the solution is simple. It is just a list of items. An item contains its weight value and its profit value. This representation makes it easy to iterate through the values in future functions.

In order to make this representation possible the values in the OR data file get parsed and converted into items in form of an object. A list of selectable items then get passed to the algorithm function.

2.2. Evaluation function

For the evaluation function I simply added the profit values of the items in the solution up. This may be not the smartest idea. For example you could add up all items by dividing the profit by the weight. In this way you would get a relationship between the profit and the weight. An item which is 15 worth and has 30 weight would be less attractive than an item which has a profit of 15 and a weight of 1.

2.3. Neighborhood function

In the neighborhood function the algorithm searches for the best item in the neighborhood. The neighborhood are the next five items in the list of all selectable items starting from the last added item in the current solution. In the beginning, the last added item is always the initial solution.

The size of the neighborhood can change. If the worst item, which gets dropped out of the solution if the solutions weight hits the constraint limit, is the same item which has been added from the neighborhood function, the neighborhood increases every time this happens by five. This is necessary in order to not get stuck in the local search by adding and deleting the same item over and over again.

If the algorithms tries to access items which are out of bounds it continues at the beginning of the list in which all the selectable items are in. It's like a window which slides over an infinite repetition of the same value sequence. This provides a constant neighborhood size.

2.4. Acceptance criterion

The acceptance criterion is the same as in the standard implantation of the simulated annealing algorithm. A better solution will always be picked and if it's not a better solution the function $P = e^{-c/t} > r$ decides.

In order to remember the best solution I assign the current solution to the best solution every time the current solution is better than the best solution.

2.5. Pseudo Code

To give a quick overview of the algorithm I wrote a pseudo code of the algorithm in the grey box below. The algorithms stops if the temperature reaches the final temperature. The mentioned “item list” in the pseudo code is the list of all selectable items from the test instance which get passed to the method.

```
Solution SimulatedAnnealing()
{
   Init variables;
   Create new solution s;
   Put the first item from the item list into the solution s;

   while(currentTemperature >= finalTemperature){
     while(m != repititionSchedule){
       bestNeighbour = find neighbour with the highest profit value;
       Create a new Solution s1 and add bestNeighbour to it;
       while(s1 is infeasible){
         drop the worst item w;
       }
       if(w == bestNeighbour){
         Increase the neighbourhood;
       }
       evaluationDelta = s1.evaluate() - s.evaluate();
       if(evaluationDelta > 0){
         s = s1;
       }else{
         if(Random(0,1) < e^(-evaluationDelta/currentTemp){
           s = s1;
         }
       }
       if(evaluate(bestSolution) < evaluate(s){
         bestSolution = s;
       }
       Increase the repitition counter m++;
     }
     Decrease the system temperature t = a*t;
   }
   Return bestSolution;
}
```

3. Performance tests

I used the third test instance (weight constraint = 11551) of the first test problem from the OR-Library file. I did three different performance tests. In the first one I measured at each iteration of the algorithm the result of the evaluation function of the current best solution and the current temporary solution. The parameters are set as follows: initial temperature = 500, temperature control parameter = 0.85, final temperature = $1.0 * 10^{-5}$, iteration at each temperature = 1 and neighborhood size = 5.

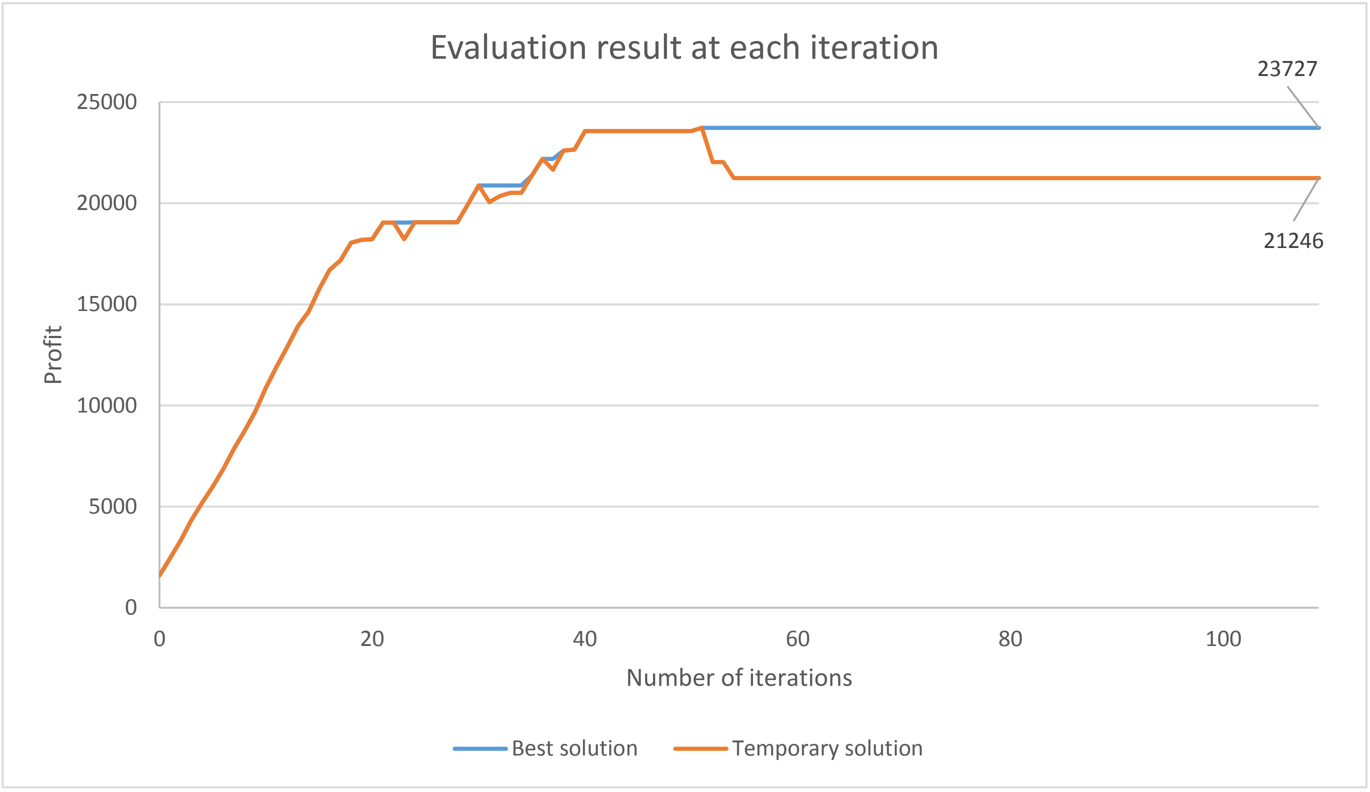

Figure 1: Evaluation result at each iteration

It is clear to see that taking a worse solution benefits the end result. A greedy search would have stopped at iteration 22 while simulated annealing tries to find a better solution through allowing worse states.

In the second performance test I changed the repetition value and measured the end results of the best solution and the CPU time it needed. All other parameters are the same as in the first test.

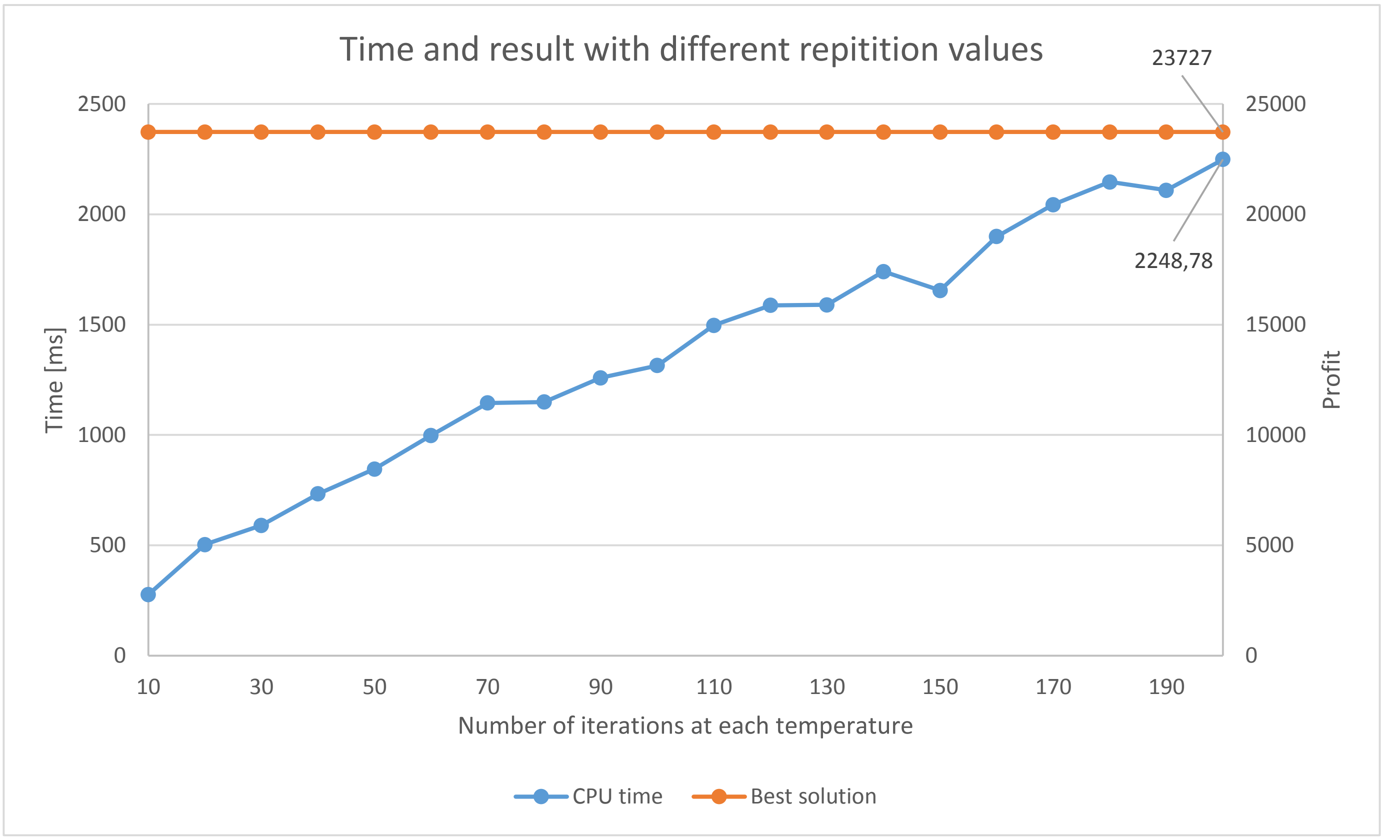

Figure 2: Time and result with different repetition values

The second test isn't thrilling. As I increased the number of iteration at each temperature the CPU time increases. The evaluation value of the solutions doesn't change. So in this case it would be better to stick to 10 as repetition value.

In the final test i used different temperature control parameters to see how the CPU time and result changes. All other parameters are the same as in the first test except the repition value. It is set to 100 which is the amount of selectable items.

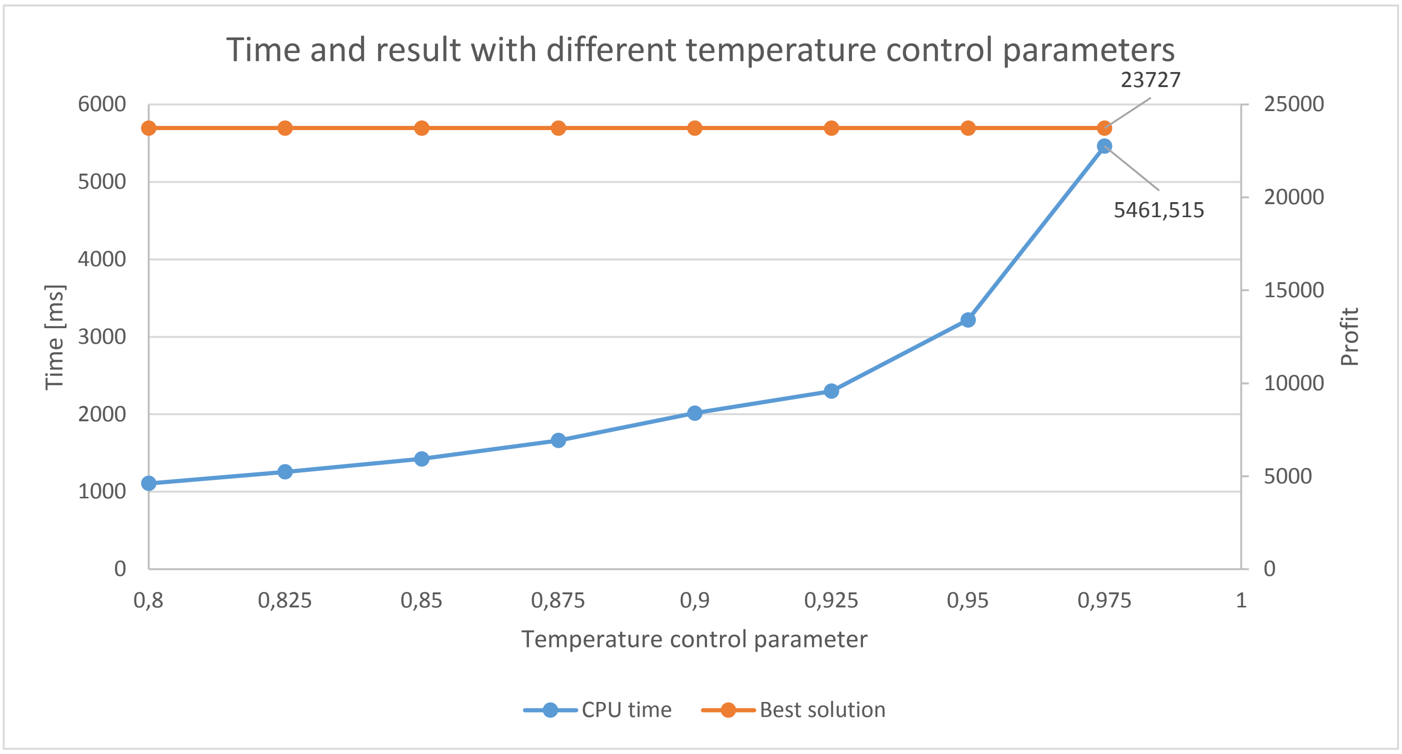

Figure 3: Time and result with different temperature control parameters

Even in the final test the result doesn't change. However the CPU time increases along with the temperature control parameter. That is because the temperature decreases slower with a higher α in the function $t = t * \alpha$. Therefore, the iterations increases and the CPU time increases.

4. Review of a paper

The paper I reviewed is the same I build-on my algorithm [1]. There are some things I like about the algorithm which they used in the paper. The first thing is the removal of an item if the solution is infeasible. It gives the algorithm the opportunity to look for new solutions even if the weight constraint has been met. I also like the random pick of an initial solution because it can give you a good starting state. On the other hand there is also a chance to get a bad starting state.

One of the things I dislike is that the algorithm can't adjust the size of the neighborhood. So the algorithm always uses the whole data set and picks an item randomly from it. This can lead to unnecessary iterations in case that an item is already picked. In general the algorithm just picks any item whatever if it’s a good item or not. The algorithm has the same behavior when it removes an item from the solution if it is infeasible. I just removes an item randomly instead of looking at its value. All this can slow down the time to find the best solution or even make the end result worse which can happen because the algorithm doesn’t run infinite.

5. References

[1] R. D. Fubin Qian, „Simulated Annealing for the 0/1Multidimensional Knapsack Problem,“ [Online]. Available: http://wenku.baidu.com/view/7a7b38d9a58da0116c174952.html.

[2] B. Leibig, „Simulated Annealing,“ [Online]. Available: http://www.cs.rit.edu/~ib/Classes/CS801_Spring11-12/Presentations/Brian_SimulatedAnnealing.pdf.

[3] C. Rickert, „Benchmarking a Simulated Annealing Approximation,“ [Online]. Available: http://www.colinrickert.com/sim-anneal-knapsack.pdf.

[4] P. N. &. S. Russell, Artificial Intelligence: A Modern Approach.