**Benjamin Nguyen**

**CS 461 Project 2 Report – Simulated Annealing Algorithm**

**Description**

Setting up

In the beginning, the program initialized temperature **T** = 100, number of **change** = 0, number of **attempt** = 0, and created a list of 400 zeros (meaning nothing is chosen) then randomly selected 20 items of 400 items in the files by the function **createlist().** Then set the temperature equal to 100 (original temperature). Inside the loop, , the variable **attempt** was incremented by 1 and the program call the **modify()** function for the list.

Inside the **modify()** function, if the score (that was the total weight of the current list) exceeded 500, then it will be deducted by 20 for each score. Then it returned that score, the total weight, the total utility for the main program. After calculating the current score, the current weight, and the current utility, the program made a new list by changing the current list by calling the **change()** function.

**change\_c()**function would receive the current list as a parameter, and it randomly selected an index inside the list, if that index had the value 1, then the value would be changed to 0, conversely, if the value of that index was 0, it would be changed to 1. Then the function would return the new list for the main program.

The main program once again calls the function **modify()**to calculate the score, total utility, total weight for the new list.

Annealing begins

* Accepting Scenario 1: There was a variable called **delta\_u**determined by the difference between the new score and the old (current) score. If it was greater than or equal to 0 meaning that the new solution was better, the old (current) list was replaced by the new list. Variable **change**was incremented by 1.
* Accepting Scenario 2: If **delta\_u** is less than 0 indicating new solution was worse than the current one, but as long as its value can satisfy the condition edelta\_u/T > random(0,1), it is still be accepted and the new list replaces the old list. This step is very crucial for the algorithm, overcoming local maxima requires it to accept bad moves sometimes. Variable **change** was incremented by 1.
* The old list would be retained if those two requirement wasn’t met.

Changes and attempts checking

**(1)**In case there is no change accepted (change =0) and attempt >40000 means that a near-optimal solution is found, the program will print the output as well as write the output to the file **Final Result.txt.**That file contained the total number of items packaged, total weight, the total utility of the final chosen list, and it also wrote the progress that contains the state, temperature, attempt, the total weight, the total utility of each change as a CSV file named **Progress.csv.**Eventually, the loop was broken when the final solution was found.

**(2)** In case the temperature was larger than 0 (there are some changes accepted) but the attempt was larger than 40000, or the changes exceeded 4000 before the attempt reached 40000. The temperature would be decreased by multiplying itself for 0.95 (geometric cooling), **temperature** and **change** were reset = 0 and the loop would start over until it met the requirements in **(1).**

**Details**

In the program, I mostly used lists (linear data structure) because it was simple to work with. I found it very efficient, maybe the data could be handled better with a dictionary, I was not sure about that but I was familiar with lists so I chose it. The thing that made me feel proud of is the annealing part, I tried so much to find out how to make it run properly, in a few first attempts the results were very weird, they even had negative numbers, I put a breakpoint many times at different places to find out bugs. The loop was repeated thousands of times and was not easy to debug. Fortunately, everything worked out. I also frequently used deepcopy method to make sure every value I need was isolated, if I used the normal copy, python would generate shallow copy and it might affect the code.

**Overall**

This method is interesting. It simulates exactly the actual annealing in the real world. About the efficiency, the program could produce a very good solution, it nearly reached the peak, however, that was all the SA can do. Unfortunately, it never guaranteed the best solution, the final result varied each time I ran the program, it was different in some units, so there was probably a better result than what I reached. One more important feature was the efficiency and the time of simulated annealing depended heavily on the alpha number, which was 0.95 in my program. The closer to 1 of the alpha, the longer the time to run the program and the better the result. However, it was not good to set this value very close to 1, because it might take forever to run and the result was just improved a little amount. I tried assigning different values for it, and I realized that if I set 0.995 with the temperature 100, I need to wait no less than half an hour for the final result, 0.90 would be faster, the result was not as good as 0.995, but the improvement was not significant, it just could raise some units and was totally not worth to wait, so I decided to set 0.95, it took long enough to produce a good result, not as low as 0.9 and very close to 0.995, and it was definitely so much faster than 0.995. The iteration was also varied by changing the alpha number and the temperature, the higher the temperature and alpha number, the higher the number of iterations, there were about 200,000 to 330,000 states changed and about 3,000,000 iterations of the total.

There was one more thing that I inspected, for the high temperature, the utility is not actually raised, they just went back and forth in the range 300-400, no matter how high the temperature, just till the temperature got lower, specifically below 10, the changes in utility were gradually clear, and below 3, it increased from about 500 to more than 700 in just 2 last “degrees.” I could understand it because, at high temperatures, the program was “looser”, it easily accepted a bad result, but when the temperature was very low, it would be very strict in choosing these moves. That also explained why at high temperatures, the program ran so fast but the lower the temperature, the longer the program ran, actually, it was still running at the same speed but because of a few changes at low temperatures caused fewer things to show up on the console so I felt it was slower. In fact, the program, as well as the simulated annealing algorithm, were not slower, they were stricter.