SLS methods for the Multidimensional Knapsack Problem

Ubeda A. Keneth[1]

[1]Université libre de Bruxelles, Belgium

kubedaar@ulb.ac.be

1. Introduction

The main idea of this experiment is to compare the implementation of two different Stochastic Local Search (SLS) methods for the Multidimensional Knapsack Problem. There are many different algorithms proposed in the literature like: Ant colony optmization [Liangjun et al. 2007], Genetic algorithms [P.C. and J.E. 1998], Simulated Annealing [Qian and Ding 2007], etc. In this experiment the Genetic algorithm (GA) [P.C. and J.E. 1998] and the simulated Annealing (SA) [Qian and Ding 2007] have been implemented. The goal is to compare two different classes of SLS as are GA and SA which belongs to class population-based and single-state respectively. The algorithms differed from the ones proposed in the literature since some constructive heuristics (CH) and improvements have been added.

As a reminder the 0–1 Knapsack problem (KP) consists on placing a finite number of items in a knapsack, where each item has an associated value and volume; the solution should not exceed the capacity of the knapsack and most important obtain the highest possible profit. Therefore the MKP is based in the same idea as KP but allows to have more than one knapsack within the problem. This means that the MKP consists on n items, m knapsack. Due the multidimensional property the problem needs n * m constraints being careful of the knapsacks capacity. So the restriction for one item to be part of the solution is to not violate any constraint. An important point is the fact that the constraints are constructed based on the capacities and the weight or volume the item takes on the knapsack. One last important detail about the MKP is its objective function.

$$max \sum_{i=1}^{n} P_i \chi_i \tag{1}$$

This experiment is focused in measuring the relative solution quality with respect to the best known values and the solution quality distribution throughout three different computation run times. These metrics will be used to determine whether there is a significant difference between the SLS methods and also observe their performance individually. Each method is going to be well described in its own section.

The algorithms have been developed using Java 1.8 and Java Microbenchmark Harness (JMH) has been used for the performance tests, also more detail will be given in the Experiment section.

2. SLS Methods

Two methods from different classes have been implemented GA (population-based)[P.C. and J.E. 1998] and SA (single-state)[Qian and Ding 2007].

These methods are highly affected by a set of parameters that can make the perform really bad or really good depending on the set up values. This issue is going to be discussed within each method section.

2.1. Genetic algorithm

A genetic algorithm is often regarded as an intelligent probabilistic search algorithm based on the evolutionary process of biological organisms in nature. During the evolution process, natural populations evolve according to the principles of natural selection and survival. The individuals who are more successful in adapting to their environment have a better opportunity to survive and reproduce, on the other hand individuals who are less fit will be eliminated. The genes of the better individuals will be spread out to the successive generations, creating each time better and better individuals.

In order to simulate these processes GA needs an initial population of individuals and apply some genetic operators in each reproduction. It is important to mention that the fitness of an individual is evaluated with respect to a given objective function which in this is case is the MKP objective function. The genetic operators are:

- **Selection**
- **Crossover**
- **Mutation**

In this implementation a fourth operator called **repair** is used to repair solutions which after the other three operators violate any constraint. The details of the operators and the initial population will be explained in the following subsections.

One more important remark is that in this experiment the representation of each individual is a n-bit binary string, where n is the number of variables in the MKP and at the same time is the 0/1 array that represents the selected items in the solution.

2.1.1. Inital population

The initial population is initialized by a group of random solution individuals which have been improved with the First improvement (FI) algorithm. The random constructive heuristic has been selected due to the fact that GA is a stochastic method in which diversification is preferred. The random solutions have been improved in order to increase the quality of the individuals. Since the population size can be considerably big as a result of previous experiments the FI is the one which combined with random initial solutions yields the lower computation times and compared with the Best Improvement (BI) the solution quality is not significantly better.

It is important to mention that the population size might affect at the final result. Usually having enough diversification is good, however generate this initial population would take considerably time. So having a balance between individual solution quality and diversification is always better. This implementation generates 100 individuals in the initial population.

2.1.2. Parent Selection

The tournament selection method is used to select the parents which later will generate one or more children. The method consists in create two different pools of randomly selected individuals, each pool consisting of the half of the population. The two individuals with the best fitness, each taken from different pools, are chosen to be the parents. The number of individuals within each pool can increase if you want to increase the pressure on the more fit individuals. Because it can be implemented very efficiently the standard

binary tournament selection method has been selected for this implementation.

2.1.3. Crossover

To generate a new individual a bit from one or the other parent is copying. The parent of whom a bit will be taken is selected randomly according to a binary random generator [0,1].

2.1.4. Mutation

Once a new individual has been generated the mutation procedure is done shifting some randomly selected bits from the individual. An important remark at this point is that the mutation rate has to be small value in order to not lose too much from the parents and consequently lose the quality inherited from its parents. For this implementation 2 has been selected as mutation rate.

2.1.5. Repair

It is highly probable that the solution generated by the crossover and mutation may not be feasible which means that some constraints may be violated. In order to guarantee feasibility a random heuristic operator was used.
First the items in the solution are shuffled and one by one removed until the solution is feasible. After that the items not in the solution are also shuffled and one by one added while any constraint is violated.

2.1.6. Parameters and final algorithm

As it is mentioned before there are some parameters whose values affect the final result. Here is a list of the parameters taken into account in this implementations:

Population size: this is a trade off between diversification and quality.
Mutation rate: this has to be a small number in order to preserve quality of the genetic structure of the parents.
Finalization criterion: One can decide between apply all the process a defined number of time or maximum time.

The final GA is summarized in the following code.

```
1  initializePopulation();
2  Solution bestSolution = getBestSolFromPop();
3  long start = System.currentTimeMillis();
4  long end = start + tmax.longValue();
5  while (System.currentTimeMillis() < end) {
6      Solution c;
7      do {
8          LinkedList<Solution> selection = tournament();
9          c = crossover(selection);
10         c = mutate(c);
11         c = repair(c);
12     } while (isInPopulation(c));
13     updatePopulation(c);
14     if (c.getValue() > bestSolution.getValue()) {
15         bestSolution = c.copy();
16     }
17 }
18 return bestSolution;
```

The GA implemented has as a termination criterion a fixed amount of time. During each round the GA applies the different operators to generate new better quality solutions. If the GA generates an individual that already exist in the population the operators are applied until a different individual has been found. It is important to mention that each time a new individual is generated the population is updated substituting the individual with the lowest fitness by the new improved one.

2.2. Simulated Annealing

This method is well known as local search algorithm capable of escaping from local optima. Its name has been taken from its analogy to the process of physical annealing with solids, in which the solid is heated and then allowed to cool very slowly until achieve a stable state. This methods aims to simulate this type of thermodynamic behavior in searching for global optima. At each iteration of the SA a comparison between the current solution and the new generated after the annealing procedure is done, better solutions, greater objective function value for the MKP, are always accepted. However a fraction of lower quality solutions are accepted trying to escape a local optima in search of global optima. Since the formulation for accepting is the following:

$$exp[(f(\omega') - f(\omega))/t_k] \qquad (2)$$

The probability of accepting non-improving solutions depends on a temperature parameter. The key feature of SA is its mechanism to escape local optima by allowing worse moves. As soon as temperature is decreased to zero, worse moves occur less frequently, and the solution distribution that models the behavior of the algorithm converges to a distribution in which all the probability is concentrated on the set of globally optimal solutions, making the algorithm asymptotically convergent.

These algorithm is highly affected by its parameters configuration. The parameters taken into account for this implementation are the following ones:

Initial Temperature (t): Setting this parameter higher results in accepting worse solutions while setting it lower results in accepting only better solutions. The main goal is to find a trade off between escaping from local optima and be to weak and accept too many worse solutions which makes the algorithm performs slow.

Final temperature (e): This value is usually set to a number very near to 0.

Temperature control (α): This is a very sensible parameter which can determine if the temperature is reached very quickly or very slowly only changing decimals.

Iterations per temperature (m): This can be a key factor in the sense that if a global optima exists within a specific temperature it worth the effort of search deep enough but not too much because the risk of being trapped in a local optima also increases if this value increase since instead of looking around the algorithm look inside the current temperature.

Termination criterion ($maxtime$): Usually as a termination criterion the final temperature is used, but for this implementation an fixed amount of time has been required.

This implementation is based on the one proposed by [Qian and Ding 2007], which was designed to give good solutions as quick as possible. How ever since this experiment is designed to run the methods in a considerable fixed amount of time some modifications haven been done. The most important change is that if the final temperature (e) is reached before the $maxtime$ has been reached the algorithm starts over again using its initial configuration. The algorithm is described in the following code:

```
Solution IncSolution = w.copy();
long start = System.currentTimeMillis();
long end = start + tmax.longValue();
Float originalT = t;
while (System.currentTimeMillis() < end) {
    while (!(t < e)) {
        for (int mi = 0; mi < m; mi++) {
            Integer i = generator.nextInt(n);
            Solution w1 = w.copy();
            if (w1.getSolution()[i] == 0) {
                w1.addItem(i);
                w1.checkViolatedConstraints();
                while (!w1.isFeasibleSolution()) {
                    int itemToDrop;
                    do {
                        int rnd = generator.nextInt(w1.getNumSelected());
                        itemToDrop = w1.getSelected()[rnd];
                    } while (itemToDrop == i);
                    w1.removeItem(itemToDrop);
                    w1.checkViolatedConstraints();
                    Integer d = w1.getValue() - w.getValue()
                    if (d >= 0 || generator.nextFloat() < Math.exp((d / t))) {
                        if (w1.isFeasibleSolution()) {
                            w = w1.copy();
                        }
                    }
                }
            } else {
                w1.removeItem(i);
                PriorityQueue<Map.Entry<Integer, Object>> tmpPqueue
                        = new PriorityQueue<>(w1.sortNonInsertedList());
                while (!tmpPqueue.isEmpty()) {
                    int itemToAdd = tmpPqueue.poll().getKey();
                    if (i != itemToAdd) {
                        w1.checkBeforeAddItem(itemToAdd);
                    }
                }
                Integer d = w1.getValue() - w.getValue();
                if (d >= 0 || generator.nextFloat() < Math.exp((d / t))) {
                    w = w1.copy();
                }
            }
            if (IncSolution.getValue() < w.getValue()) {
                IncSolution = w;
            }
        }
        t = a * t;
    }
    w = IncSolution.copy();
    t = originalT;
}
return IncSolution;
```

As it is possible to observe each time the final temperature is reached the algorithm start again with its initial configuration. Also it is important to mention that when the item is already in the solution this is removed and items not in the solution are added randomly if no constraint is violated. On the other hand when the item is not in the solution the item is added and if any constraint is violated different items are randomly removed until no constraint is violated. It is also important to mention that since this algorithm starts with an initial solution a Random constructive heuristic has been selected in which is applied a Best improvement in order to start with a enough quality solution.

3. Experiment and results using JMH benchmark

JMH was used to measure the computation time taken by the different algo-

rithms. JMH is an OpenJDK project that aims to facilitate setting up a benchmark environment for Java performance tests. the tests have been automatized using JMH and it is important to mention that since GA and SA are Stochastic algorithm the results showed below are the average of 5 trials using different random seeds and in order to measure the quality solution throughout three different times each algorithm has been execute 20 times.

As a metric of solution quality, the Relative percentage deviation with respect to the best known solution values has been used.

$$\triangle_{ki} = 100 \cdot \frac{profit_{ki} - bestknown_i}{bestknown_i} \quad (3)$$

As a criterion of termination a maximum time in seconds was required which is defined as follows:

$$maxtime = n \cdot \frac{m}{10} \quad (4)$$

Table 1. Computer specifications

Type	Compute optimized AWS EC2 mc5.large
CPU	Intel Xeon Platinum 8124M 2 cores
VCPU	2
Clock speed	3 GHz
RAM	4GB
OS	TheAmazon Linux AMI OS
Java	Java 1.8

These are the random seeds used in the executions for the SLS methods, the first five seeds were used to the 5 trials. and the seed for the shuffle step in the initial solution for the SA.

```
private final static Long IMPROVEMENT_SEED =
    4619L;
private final static Long RANDOM_SEEDS[] = {
        7440L, 1437L, 7124L, 8923L, 5387L,
    1279L, 1263L, 1872L, 1252L, 1394L,
    1015L, 1009L, 3882L, 1130L, 4602L,
    1047L, 1353L, 2952L, 2177L, 1586L
};
```

For this experiment 60 different instances were provided, these instances are divided in two groups: 10 knapsack and 100 items (10x100) and 10 knapsacks and 250 items (10x250). Additionally in order to analyze the solution quality throughout various run times the first 5 instances from the 10x250 group were selected to form a new group (F5I10x250). Each group will be discussed independently

In order to compare the SLS methods a Wilcoxon test with Bonferroni correction has been done also there are summary tables showing the global results of the tests.

3.1. SLS configurations

The algorithms has been executed over the following configurations:

Table 2. GA configuration

GA	
Parameter	**Value**
Mutation rate	2
Population size	100
Termination criterion	$maxtime$

This configurations attempt to generate better solutions with a considerably initial population size, with 100 random individuals the algorithm reaches enough diversification and since these individuals are improved with FI it reaches enough good quality individuals for the next generations, which is at the same time the reason to change only 2 bits of its chromosome in the mutation procedure.

Table 3. SA configuration

SA	
Parameter	**Value**
Initial Temperature	100
Final Temperature	0.00001
Temperature control	0.845
Iterations per temperature	10
Termination criterion	$maxtime$

The goal of this configuration is to quickly converge since the temperature is not very high neither the temperature control. The Iterations per temperature parameter was thinking to not lose too much

time and be able to explore through different temperatures instead.

3.2. 10x100 Instances

This group of instances has been executed 5 times during 100 seconds.

Table 4. SLS methods 10x100 Instance group

	GA	SA
Instance	**Δavg**	**Δavg**
OR10x100-0.25_1	**0.03**	0.30
OR10x100-0.25_2	**0.22**	0.63
OR10x100-0.25_3	**0.18**	0.52
OR10x100-0.25_4	**0.08**	0.65
OR10x100-0.25_5	**0.25**	0.71
OR10x100-0.25_6	**0.27**	0.79
OR10x100-0.25_7	**0.27**	0.56
OR10x100-0.25_8	**0.23**	0.85
OR10x100-0.25_9	**0.24**	0.89
OR10x100-0.25_10	**0.00**	0.56
OR10x100-0.50_1	**0.04**	0.27
OR10x100-0.50_2	**0.03**	0.35
OR10x100-0.50_3	**0.19**	0.19
OR10x100-0.50_4	**0.15**	0.40
OR10x100-0.50_5	**0.11**	0.32
OR10x100-0.50_6	**0.08**	0.31
OR10x100-0.50_7	**0.05**	0.20
OR10x100-0.50_8	**0.10**	0.15
OR10x100-0.50_9	**0.00**	0.12
OR10x100-0.50_10	**0.20**	0.32
OR10x100-0.75_1	**0.00**	0.04
OR10x100-0.75_2	**0.09**	0.08
OR10x100-0.75_3	**0.01**	0.09
OR10x100-0.75_4	**0.02**	0.12
OR10x100-0.75_5	**0.00**	0.00
OR10x100-0.75_6	**0.16**	0.27
OR10x100-0.75_7	**0.03**	0.08
OR10x100-0.75_8	**0.09**	0.10
OR10x100-0.75_9	**0.02**	0.19
OR10x100-0.75_10	**0.00**	0.02
Δavg	**0.10**	0.34

It is easy to see in this table the behavior of the two SLS methods across all the instances since lowest deviation values are highlighted. One can observe that these values correspond to the GA individually and in average.

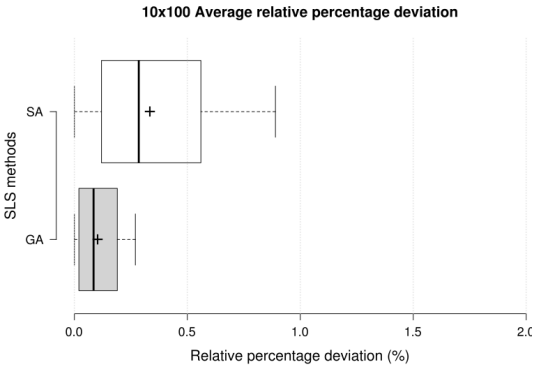

Figure 1. Percentage Deviation

A Wilcoxon Signed-rank test indicates that GA gets a lower relative deviation compared to SA as the Box plot shows where the distribution of GA is clearly below to the SA distribution.

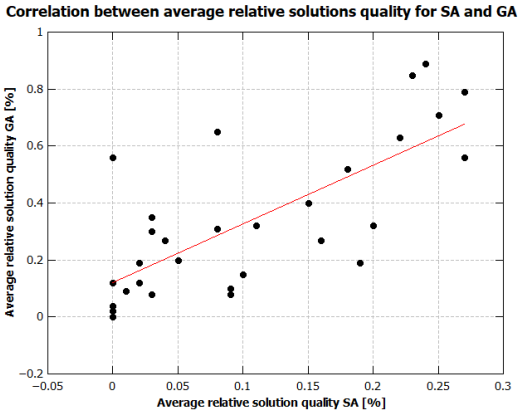

Figure 2. SA and GA correlation

As it is possible to see in the figure 4 the behavior of the average relative percentage deviation is not very similar between the two algorithms since the point are not quite close to the diagonal.

3.3. 10x250 Instances

This group of instances has been executed 5 times during 250 seconds.

Table 5. SLS methods 10x100 Instance group

	GA	SA
Instance	**Δavg**	**Δavg**
OR10x250-0.25_1	**0.33**	0.71
OR10x250-0.25_2	**0.31**	0.94
OR10x250-0.25_3	**0.30**	0.67
OR10x250-0.25_4	**0.52**	1.00
OR10x250-0.25_5	**0.56**	1.02
OR10x250-0.25_6	**0.45**	0.86
OR10x250-0.25_7	**0.33**	1.05
OR10x250-0.25_8	**0.52**	0.73
OR10x250-0.25_9	**0.26**	0.69
OR10x250-0.25_10	**0.38**	0.87
OR10x250-0.50_1	**0.18**	0.45
OR10x250-0.50_2	**0.21**	0.51
OR10x250-0.50_3	**0.17**	0.34
OR10x250-0.50_4	**0.16**	0.27
OR10x250-0.50_5	**0.17**	0.52
OR10x250-0.50_6	**0.18**	0.55
OR10x250-0.50_7	**0.15**	0.60
OR10x250-0.50_8	**0.18**	0.48
OR10x250-0.50_9	**0.15**	0.54
OR10x250-0.50_10	**0.16**	0.50
OR10x250-0.75_1	**0.06**	0.16
OR10x250-0.75_2	**0.06**	0.14
OR10x250-0.75_3	**0.09**	0.21
OR10x250-0.75_4	**0.07**	0.19
OR10x250-0.75_5	**0.07**	0.18
OR10x250-0.75_6	**0.09**	0.20
OR10x250-0.75_7	**0.09**	0.23
OR10x250-0.75_8	**0.07**	0.21
OR10x250-0.75_9	**0.07**	0.26
OR10x250-0.75_10	**0.07**	0.16
Δavg	**0.21**	0.51

It is easy to see in this table the behavior of the two SLS methods across all the instances since lowest deviation values are highlighted. One can observe that these values correspond to the GA individually and in average.

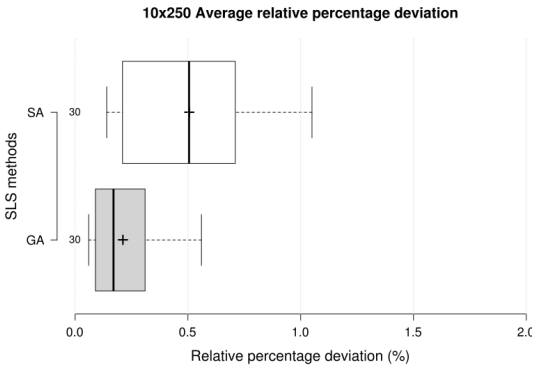

Figure 3. Percentage Deviation

A Wilcoxon Signed-rank test indicates that GA gets a lower relative deviation compared to SA as the Box plot shows where the distribution of GA is clearly below to the SA distribution.

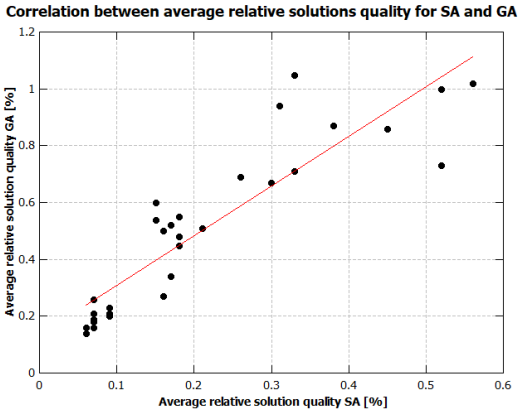

Figure 4. SA and GA correlation

As it is possible to see in the figure 4 the behavior of the average relative percentage deviation is not very similar between the two algorithms since the point are not quite close to the diagonal.

4. F5I10x250 Instances

This group of instances has been executed in various run times: 0.01 * maxtime (2.5 seconds), 0.1 * maxtime (25 seconds) and maxtime (250 seconds). The goal is to measure the solution quality distribution throughout different run times.

Table 6. Average relative percentage deviation

	0.001		0.1		1	
Instance	**GA**	**SA**	**GA**	**SA**	**GA**	**SA**
OR10x250-0.25_1_0.01	1.54	1.85	0.72	1.08	**0.34**	0.86
OR10x250-0.25_2_0.01	1.55	2.33	0.73	1.58	**0.35**	1.19
OR10x250-0.25_3_0.01	1.35	2.14	0.63	1.25	**0.31**	0.71
OR10x250-0.25_4_0.01	1.41	1.91	0.75	1.18	**0.43**	0.91
OR10x250-0.25_5_0.01	1.58	1.90	0.96	1.48	**0.59**	1.08
Δavg	1.48	2.03	0.75	1.31	**0.40**	0.95

It is easy to see in this table the behavior of the two SLS methods throughout three different run times across the first five 10x250 instances since lowest deviation values are highlighted. One can observe that these values correspond to the

GA per each run time and also it is possible to observe that that both GA and SA yield better quality solutions over time.

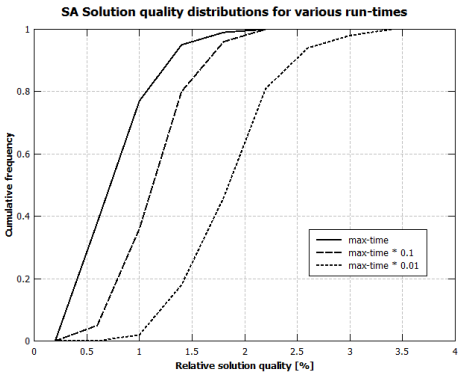

Figure 5. SA Solution quality distribution

If one observes the figure 5 it is possible to compare the quality distribution of SA through three different run times. One can also observe that over the time SA always results in better quality solutions.

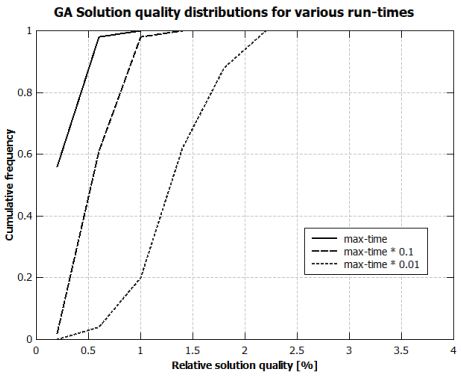

Figure 6. GA Solution quality distribution

If one observes the figure 7 it is possible to compare the quality distribution of GA through three different run times. One can also observe that over the time SA always results in better quality solutions.

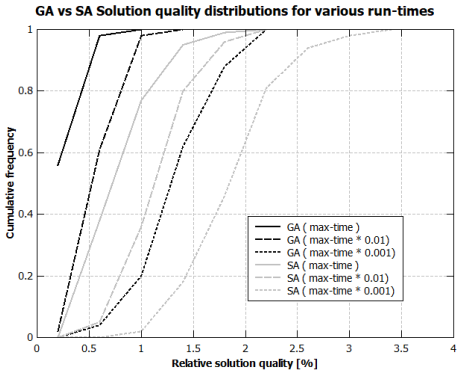

Figure 7. GA vs SA Solution quality distribution

In figure 7 one can compare the quality distributions of the two algorithms. One can observe that the first two best distribution are given by the GA in the $maxtime$ and $0.1 * maxtime$ run times respectively. Also one can observe that the solution quality distribution for the $maxtime$ and $0.1 * maxtime$ of the SA are better than the $0.01 * maxtime$ distribution for the GA.

5. Conclusion

Both algorithms generate very good quality solutions over the $maxtime$ used in this experiment. How ever since it is a significant statistical difference between the algorithms one can conclude that over the time the Genetic algorithm implemented generates, in average, better quality solutions for the Multidimensional Knapsack Problem. After observe the run time analysis, one can observe that for all the given run times Genetic algorithm generates better quality solutions than the Simulated Annealing, However its possible to see that both algorithms converge to the global optima since the much time they run the better quality solution they yield. One final observation is that the algorithms are subjected to its initial configuration therefore any change on these parameters can make a big difference in the algorithm itself and consequently a difference between different algorithms.

References

Liangjun, K., Zuren, F., Zhigang, R., and Xiaoliang, W. (2007). An ant colony optimization approach for the multidimensional knapsack problem. *Springer Science+Business Media, LLC 2008*.

P.C., C. and J.E., B. (1998). A genetic algorithm for the multidimensional knapsack problem. *The Management School, Imperial College, London SW7 2AZ, England*.

Qian, F. and Ding, R. (2007). Simulated annealing for the 0/1 multidimensional knapsack problem. *School of Management, University of Science and Technology of Suzhou, Suzhou 215008, China*.