MASTER DEGREE OF RESEARCH IN ARTIFICIAL INTELLIGENCE

COURSE 2019-2020

Assignment 3.6: Comparison of a population-based algorithm and a trajectory algorithm

Luis Gonzalez Naharro

NOVEMBER 18, 2019

Universidad Internacional Menéndez Pelayo

Asociación Española para la Inteligencia Artificial (**AEPIA**)

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

The objective of this work is to perform a comparison between a couple of metaheuristic algorithms: a population-based algorithm (namely, a genetic algorithm) and a trajectory-based algorithm (namely, a simulated annealing algorithm), in order to determine which one behaves better under a set of pre-established conditions.

As it is known, there is a class of computational problems known as NP (**Non-deterministic Polynomial**) problems, which cannot be solved on a feasible amount of time due to their computational complexity increasing exponentially as the size of the input increases. In order to tackle these problems in a more computationally cheap way, a special kind of algorithms named **metaheuristic algorithms** were developed, which can give approximate optimal solutions to these problems with a minimal amount of problem-specific-tailoring work. However, these algorithms do not guarantee to give the best solution (global optimum). Instead, they almost always give slightly worse solutions (local optima) which, in practice, tend to be very good. This fact, combined with their relatively cheap behavior, has led to their application in many fields, such as logistics, smart cities, etc.

The rest of this report is organized as follows: Section 2 details the implementation of the code for running the experiments, Section 3 details the experiments that have been carried out, Section 4 shows the results of the experiments and analyses them. Finally, Section 5 gives conclusions about the work done.

2 Technical implementation

The first task carried out in this work has been implementing the code needed for running the experiments. More specifically, code was needed for the following algorithms:

- **Genetic Algorithm**: this algorithm simulates the behavior of evolution in nature. To do so, it starts from an initial population that evolves by applying selection, crossover and mutation operands during a series of timesteps. This can be seen in more detail on Algorithm 1. Usually, the operators are defined as it follows:
  - *Selection*: the best solutions from the population are selected. There are various strategies for this process, such as Roulette Wheel Selection, Tournament Selection, etc.
  - *Crossover*: the previously chosen solutions are combined two-by-two to generate new solutions with characteristics from both their "parent" solutions. The crossover technique is heavily dependant on the problem and the solution codification.
  - *Mutation*: the resulting "children" are mutated by randomly altering elements from their solution. Once again, this process depends on the problem and the codification of the solution.
- **Simulated Annealing**: this algorithm simulates the annealing process in metallurgy, by allowing the current solution to change with a high probability to a worse solution in the beginning, and gradually decreasing this probability over time. This can be seen in more detail on Algorithm 2.

While the genetic algorithm was already implemented in the given code at [1], the implementation of the simulated annealing had to be done manually, by working over the already existing code. Another part that had to be implemented was the knapsack problem; more specifically, the **m-dimensional knapsack problem**, as stated in the given font at [2]. This problem is very similar to the original knapsack problem, with the difference that each item not only has weight but also a certain number of other unrelated constraints. The knapsack also has limit values for these constraints, and all of them must be met in order to have a valid solution.

Another issue that had to be fixed in the base program was the random seeds. The way it was before any modifications, the random values were purely random, without setting any seeds, thus leading to different behavior between executions and impeding the execution of certain statistical tests. The program has been

Algorithm 1 Genetic algorithm

```
procedure geneticAlgorithm(population)
    t := 0
    initialize(population)
    evaluate(population)
    while not stopCondition() do
        newPopulation := selection(population)
        newPopulation ← crossover(newPopulation)
        newPopulation ← mutation(newPopulation)
        evaluate(newPopulation)
        population ← replace(population, newPopulation)
        t ← t + 1
    return bestSolution(population)
```

Algorithm 2 Simulated annealing

```
procedure SimulatedAnnealing(temperature, coolingRate)
    solution := initialize()
    while not stopCondition() do
        newSolution := randomNeighbour(solution)
        if acceptProbability(solution, newSolution, temperature) > random(0, 1) then
            solution ← newSolution
        temperature ← decreaseTemperature(temperature, coolingRate)
    return solution
```

modified so as to be able to set a certain seed in the command line, and to guarantee result reproducibility. In addition, the program has also been modified to set the parameters of each metaheuristic algorithm also in the command line, in order to make an automated script to collect all the required metrics.

3 Experiments

As it is stated in the description of the assignment, the experiments must be carried out by tuning the hyperparameters of both the Genetic Algorithm and the Simulated Annealing. The parameters of the Genetic Algorithm already work well enough as they are on the code, so they have not been modified. However, the parameters for Simulated Annealing had to be manually tuned. While setting the cooling rate has been done on a trial-and-error basis, the starting temperature has been tuned with the method explained in [3].

Two kinds of experiments have been run, as it is asked on the description of the assignment:

- Leaving a very limited number of iterations (**500000** for these experiments), and returning the best fitness value obtained by the program.
- Leaving a high number of iterations (**5000000** for these experiments), and returning the number of evaluations of the objective function needed by each algorithm to find the optimum value or the maximum if the optimum wasn't found
- As an extra experiment, the same setup as before has been employed, but this time, the **execution time** of each instance has been measured. These experiments have all been run on the same machine, to guarantee equal execution conditions.

For these two kinds of experiments, the following configurations were applied to each algorithm:

- **Genetic algorithm**: *Population size:* 256*, Crossover probability:* 0.8*, Mutation probability:* 0.1
- **Simulated annealing**: *Initial temperature:* 448*, Cooling rate:* 0.0000015

Applying these configurations to the aforementioned kinds of experiments, we get 6 experiments to be executed: **Genetic algorithm testing best fitness**, **Genetic algorithm testing a number of evaluations**, **Genetic algorithm testing execution time**, **Simulated annealing testing best fitness**, **Simulated annealing testing a number of evaluations**, and **Simulated annealing testing execution time**. For each one of these experiments, 50 instances have been run, each one with random seeds varying between 0 and 49 inclusive. This setting for the random seeds guarantees that the values can be paired.

Finally, it is worth mentioning that the tool employed for generating the results graphics and the statistical tests is `exreport` [4].

4 Results

The following subsections show the results of the genetic algorithm versus simulated annealing in each of the three experiment scenarios specified (this is, *best fitness*, *number of evaluations* and *execution time*).

4.1 Best fitness

Figure 1 shows the results of the first type of experiment: comparing GA with SA by checking the best fitness value obtained by both after performing, at most, 500000 evaluations of the objective function.

By looking at the graphics, it is hard to notice any difference between them: the results are almost identical. However, there are minor differences, in the order of hundreds, between the two, as the Wilcoxon test points out: in fact, this difference is so significative that the test rejects the null hypothesis that both methods are equivalent, and says that the genetic method is better than the simulated annealing.

4.2 Number of evaluations

Figure 2 shows the results of the second type of experiment: comparing GA with SA by checking the number of evaluations of the objective function until finding the optimum value. Note that, in the case that such optimum is not found even with a high evaluation limit, the program will return this maximum value.

As it can be seen on the figures, there is a lot of cases in which one of the algorithms (even the two of them) cannot find the optimum value: in fact, the number of times they get stuck is relatively similar between the two of them, as the Wilcoxon test states: in this sense, both methods are similar.

4.3 Execution time

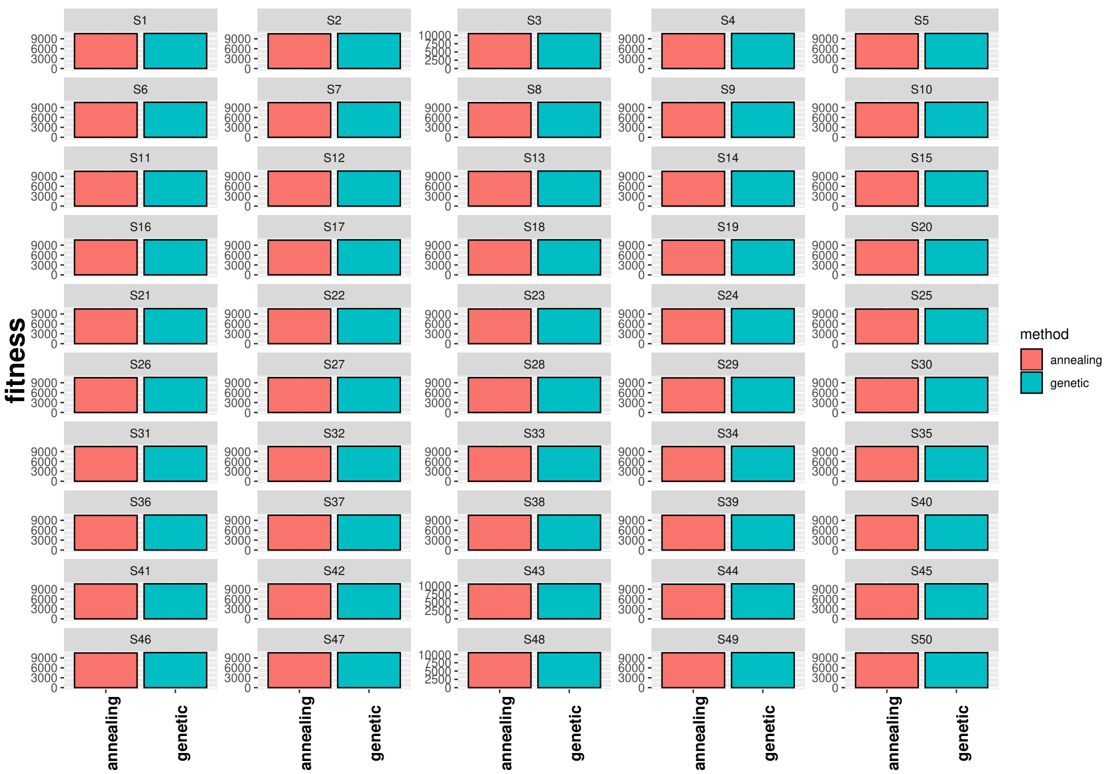
Finally, Figure 3 shows the results of the third type of experiment: comparing GA with SA in a way similar to the previous experiment (number of evaluations), but measuring the execution time instead of the number of evaluations.

In this case, as the graphics show, there is a clear dominance of simulated annealing over the genetic algorithm in execution time, except for the cases where SA doesn't find the optimum and GA does. This is further corroborated by the Wilcoxon test, which states that SA is better than GA.

4.4 Results analysis

By looking at the previous results, the first conclusion that can be drawn is that the genetic algorithm is much better than simulated annealing. This is something that the first statistical test corroborates, with the genetic algorithm outperforming the simulated annealing almost in every experiment instance.

However, there is a catch to this behavior, and it is that the genetic algorithm is computationally much more expensive than simulated annealing: as it can be seen in Table 1 (which represents the numerical values for the results shown in Figure 3), even the cases in which simulated annealing doesn't find the optimum value in the specified time limit, the time it takes outperforms some cases in which the genetic algorithm finds the optimum!



(a) GA vs SA comparison on best fitness obtained after 500000 evaluations cap, per-seed

Wilcoxon Signed Rank Test for output "fitness"

From experiment: Report

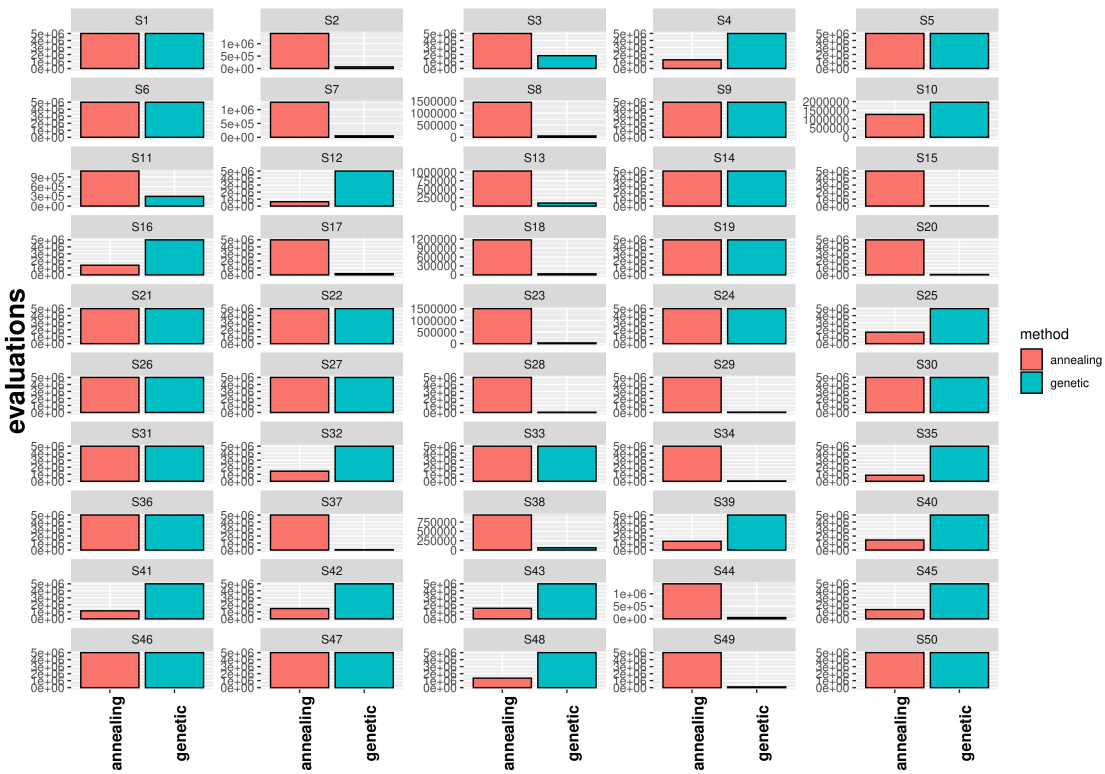
Wilcoxon Signed Rank test for output fitness

- H_0: genetic method is equivalent to annealing.
- H_1: genetic method is better than annealing.

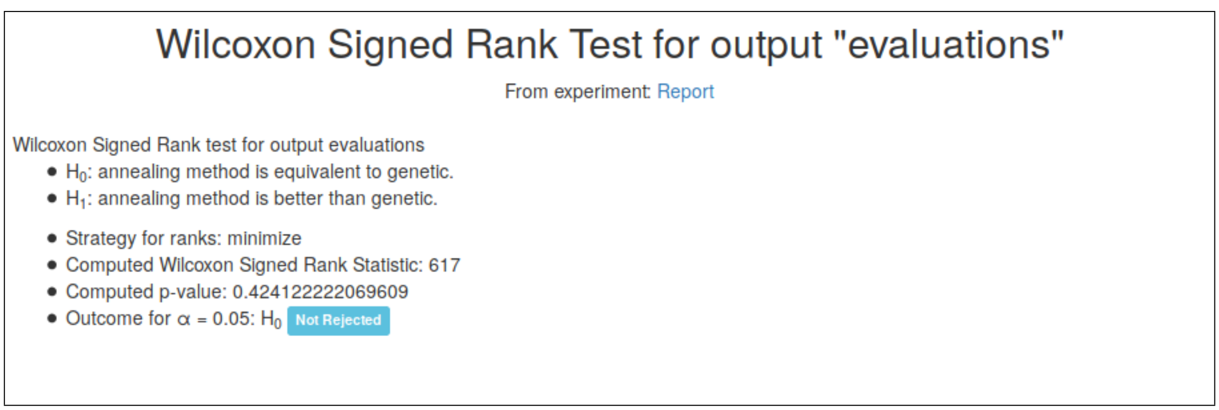
- Strategy for ranks: maximize
- Computed Wilcoxon Signed Rank Statistic: 0
- Computed p-value: 8.88178419700124e-16
- Outcome for α = 0.05: H_0 Rejected

(b) Wilcoxon paired test result for GA vs SA on best fitness after 500000 evaluations cap

Figure 1: GA vs SA: best fitness after 500000 evaluations cap



(a) GA vs SA comparison on number of evaluations until reaching optimum, per-seed



Wilcoxon Signed Rank Test for output "evaluations"

From experiment: Report

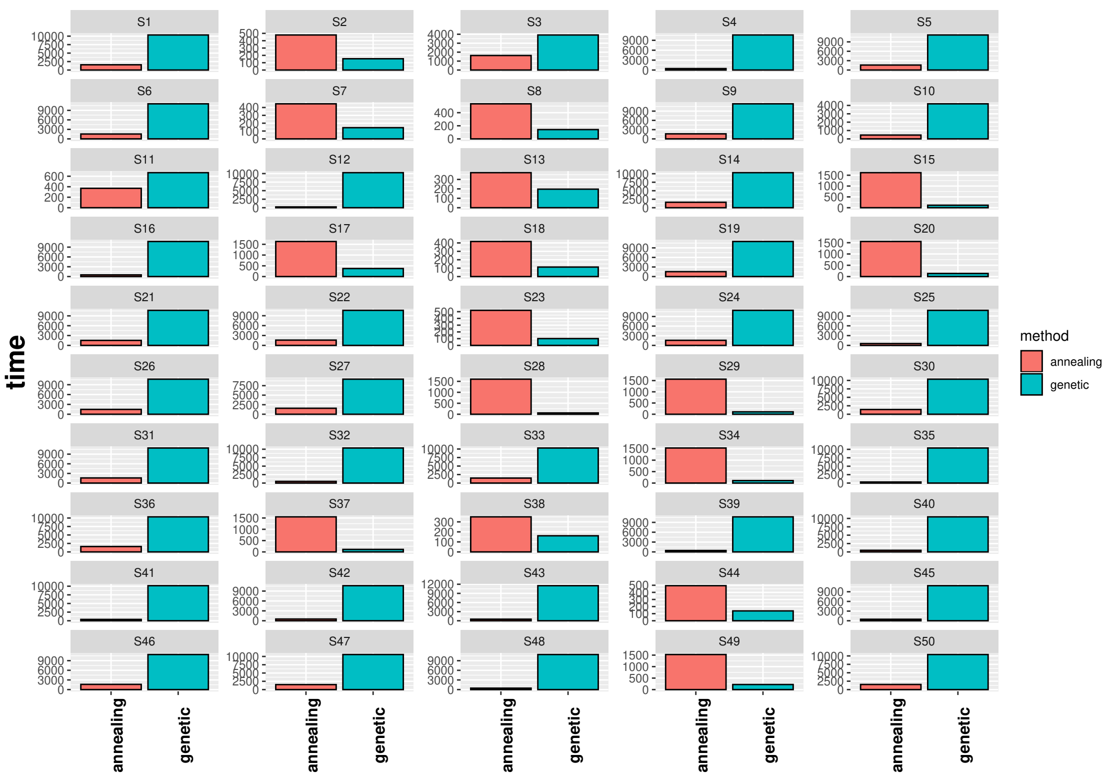
Wilcoxon Signed Rank test for output evaluations

- H_0: annealing method is equivalent to genetic.
- H_1: annealing method is better than genetic.

- Strategy for ranks: minimize
- Computed Wilcoxon Signed Rank Statistic: 617
- Computed p-value: 0.424122222069609
- Outcome for α = 0.05: H_0 Not Rejected

(b) Wilcoxon paired test result for GA vs SA on number of evaluations for optimum

Figure 2: GA vs SA: number of evaluations until reaching optimum



(a) GA vs SA comparison on execution time until reaching optimum, per-seed

Wilcoxon Signed Rank Test for output "time"

From experiment: Report

Wilcoxon Signed Rank test for output time

- H_0: annealing method is equivalent to genetic.
- H_1: annealing method is better than genetic.

- Strategy for ranks: minimize
- Computed Wilcoxon Signed Rank Statistic: 150
- Computed p-value: 1.99601182693243e-07
- Outcome for $\alpha = 0.05$: H_0 Rejected

(b) Wilcoxon paired test result for GA vs SA on execution time for optimum

Figure 3: GA vs SA: execution time until reaching optimum

With this information, it can be concluded that, even if the genetic algorithm is usually better than simulated annealing for finding the optimum, it may not be the best choice in less powerful computers, not only because of its slower speed, but also because of the higher memory consumption: obviously, it is more expensive to keep 256 elements in memory than just 1 or 2. So, there is really no better algorithm: each one will be better suited for some cases. It is also important that, since the statement of the assignment specified so, these experiments have been run on only one instance of the problem, so these results may not generalize to other cases.

5 Conclusions

In this assignment, a comparison between a genetic algorithm and simulated annealing has been performed over a specific instance of the knapsack problem. In order to do so, a code has been downloaded and modified in order to include both the knapsack problem and the SA algorithm, in addition to some more modifications to ease the process of collecting results. Then, experiments have been run and processed through a tool to extract statistical information. From these results, it has been concluded that, while GA usually outperforms SA in finding the optimum, SA uses much less computational resources, so each one is well-suited for different scenarios.

References

[1] Enrique Alba, *ssGA: Steady State GA.* `http://neo.lcc.uma.es/software/ssga/index.php`

[2] J. E. Beasley, *Multidimensional knapsack problem.* `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`

[3] Juan Frausto-Solis, E.F. Roman, David Romero, Xavier Soberon, and Ernesto Lian-Garcia, *Analytically Tuned Simulated Annealing Applied to the Protein Folding Problem.* In Computational Science-ICCS 2007 (pp. 370-377)

[4] J. Arias, J. Cozar, *exreport: An R package for easy reproducible research.* `http://exreport.jarias.es/`

(a) Best fitness

	genetic	annealing
S1	10604.0	10547.0
S2	10618.0	10507.0
S3	10570.0	10521.0
S4	10604.0	10540.0
S5	10604.0	10508.0
S6	10604.0	10535.0
S7	10618.0	10505.0
S8	10618.0	10481.0
S9	10604.0	10519.0
S10	10605.0	10470.0
S11	10618.0	10527.0
S12	10604.0	10557.0
S13	10618.0	10509.0
S14	10604.0	10499.0
S15	10618.0	10562.0
S16	10604.0	10555.0
S17	10618.0	10530.0
S18	10618.0	10560.0
S19	10604.0	10493.0
S20	10618.0	10556.0
S21	10604.0	10508.0
S22	10604.0	10515.0
S23	10618.0	10513.0
S24	10604.0	10510.0
S25	10604.0	10485.0
S26	10604.0	10510.0
S27	10604.0	10531.0
S28	10618.0	10549.0
S29	10618.0	10482.0
S30	10604.0	10448.0
S31	10604.0	10513.0
S32	10604.0	10504.0
S33	10604.0	10537.0
S34	10618.0	10517.0
S35	10604.0	10527.0
S36	10604.0	10489.0
S37	10618.0	10559.0
S38	10618.0	10547.0
S39	10604.0	10514.0
S40	10604.0	10473.0
S41	10604.0	10565.0
S42	10604.0	10563.0
S43	10588.0	10506.0
S44	10618.0	10500.0
S45	10604.0	10528.0
S46	10604.0	10489.0
S47	10604.0	10531.0
S48	10556.0	10528.0
S49	10618.0	10549.0
S50	10604.0	10491.0

(b) Evaluations

	genetic	annealing
S1	5000256	5000001
S2	65934	1411988
S3	1833894	5000001
S4	5000256	1238622
S5	5000256	5000001
S6	5000256	5000001
S7	51919	1265395
S8	55446	1450383
S9	5000256	5000001
S10	1962343	1281357
S11	300473	1084357
S12	5000256	621064
S13	90466	1040709
S14	5000256	5000001
S15	38949	5000001
S16	5000256	1381314
S17	175649	5000001
S18	36047	1175571
S19	5000256	5000001
S20	54638	5000001
S21	5000256	5000001
S22	5000256	5000001
S23	32888	1509057
S24	5000256	5000001
S25	5000256	1631830
S26	5000256	5000001
S27	5000256	5000001
S28	11626	5000001
S29	43104	5000001
S30	5000256	5000001
S31	5000256	5000001
S32	5000256	1442029
S33	5000256	5000001
S34	45974	5000001
S35	5000256	861258
S36	5000256	5000001
S37	44920	5000001
S38	61710	940714
S39	5000256	1250205
S40	5000256	1438560
S41	5000256	1146254
S42	5000256	1472901
S43	5000256	1510448
S44	52011	1399984
S45	5000256	1322201
S46	5000256	5000001
S47	5000256	5000001
S48	5000256	1358309
S49	111050	5000001
S50	5000256	5000001

(c) Time

	genetic	annealing
S1	10341	1556
S2	154	478
S3	3918	1625
S4	10608	446
S5	11202	1595
S6	11070	1539
S7	143	447
S8	142	535
S9	11369	1637
S10	4193	458
S11	668	370
S12	10270	229
S13	197	372
S14	10315	1607
S15	115	1618
S16	10720	511
S17	373	1631
S18	113	417
S19	10993	1540
S20	136	1553
S21	10677	1523
S22	10629	1600
S23	101	520
S24	10920	1584
S25	10733	554
S26	10685	1475
S27	9198	1568
S28	59	1599
S29	107	1563
S30	10370	1396
S31	10913	1600
S32	10357	498
S33	10296	1469
S34	108	1531
S35	10397	309
S36	10341	1589
S37	111	1550
S38	161	350
S39	10682	429
S40	10472	477
S41	10150	391
S42	10628	520
S43	11528	519
S44	139	494
S45	10882	463
S46	10902	1622
S47	10585	1507
S48	10924	451
S49	221	1526
S50	10417	1522

Table 1: Detailed results for the experiments performed. Note that S1 refers to seed value 0, S2 refers to seed value 1, and so on. Also, the results on b) and c) have been drawn from the same execution.