UNIVERSIDAD INTERNACIONAL MENÉNDEZ PELAYO

MÁSTER EN INVESTIGACIÓN EN INTELIGENCIA ARTIFICIAL

RESOLUCIÓN DE PROBLEMAS CON METAHEURÍSTICAS

Work 3.6 Comparison of a population and a trajectory-based (knapsack) algorithm

Author: Christian Luna Escudero
DNI: 31875907T

May 5, 2024

Vídeo: https://drive.google.com/file/d/1ibyOjIXGHx55pQGYiWp2O4BBVwoBi7UZ

Código: https://github.com/ChrisLe7/Res-Prob-Metaheuriticas-UIMP

1 Introduction

In this work, a comparative experimental study will be conducted for the multidimensional knapsack problem. This is a classic combinatorial optimization problem that seeks the optimal distribution of limited resources to maximize total benefit while satisfying certain constraints. Unlike the classic knapsack problem (single capacity value), the multidimensional knapsack has various dimensions, each with its capacity, and each resource weights each dimension. To illustrate the problem, let us consider the following example: let's assume we want to manage an investment fund (the "knapsack"). We have a portfolio of potential projects to invest in (the "objects/resources"), and each one could generate a different benefit. However, we must adhere to an established budget. Each project involves a series of costs ("dimensions"): initial investment, operational cost, and risk cost, among others. For each of these costs, there is a budgetary limit ("capacities") that we cannot exceed. For example: 50 million for initial capital, 10 million for operational costs, 15 million for risk management, etc. Therefore, when selecting projects to invest in, the accumulated cost of each type should not exceed the budget assigned for that cost.

2 Problem Formalization

Let n be the number of objects, m be the number of dimensions, x be the solution vector to evaluate, p be the profit vector, w be the weight matrix, and c be the capacity vector. The formulation is as follows:

$$\text{Maximize } \sum_{j=1}^{n} p(j)x(j) \tag{1}$$

$$\text{Subject to } \sum_{j=1}^{n} w(i,j)x(j) \leq c(i), \quad \forall i \in 1,...,m \tag{2}$$

$$x(j) \in 0,1, \quad \forall j \in 1,...,n \tag{3}$$

3 Implementation of the Evaluation Function

When developing the evaluation function, the first step is to determine if there is any violation of the problem's constraints following equation 2. If there are no violations, the process continues to the sum of benefits of the objects included in the knapsack. Otherwise, the *fitness* of the solution will be the sum of the violations that occurred in each dimension multiplied by -1. In this way, invalid solutions will have a negative *fitness*, thus monitoring if these solutions progress towards better solutions even though violations of the constraints continue to occur. This would be particularly interesting in cases where all, or almost all, possible solutions violate the problem's constraints.

4 Implementation of the Genetic Algorithm

We based the implementation of our genetic algorithm on the strategy provided by [4], introducing some modifications to the crossover and mutation operators to improve performance.

4.1 Crossover

The base-line implementation only employed the single-point crossover operator (SPX). Now, a uniform crossover has been added. Both operators are designed to produce a single new individual, discarding the alternative.

- Single-point crossover (SPX): We select a random position on the chromosome to split it. The resulting offspring inherits the left side from the first parent and the right side from the second.
- Uniform crossover (UC): This method consists of choosing the genetic material for each position of the chromosome randomly from one of the two parents, which favors a greater diversity of genetic characteristics.

4.2 Mutation

The genetic algorithm can use the following mutation operations:

- Base-line: The system generates a random number between 0 and 1 for each allele of the individual. If this number falls below the mutation probability, the system alters the value of the allele.
- Swap: The system selects two alleles from the individual and swaps their values.

The swap mutation might not change the individual if the encoding is binary and swaps identical alleles (same value).

4.3 Selection of Individuals

The genetic algorithm selects individuals for crossing using a binary tournament of size 1. This process needs to be repeated twice to complete the crossover.

5 Population Replacement

The genetic algorithm uses a simple generational substitution for population replacement, where it replaces only the worst individual in the population with the new one.

6 Simulated Annealing Implementation

This implementation adopts the same structure as that used in the genetic algorithm. The following subsections will detail the most crucial parts.

6.1 Acceptance Function

Define Δ as the difference in fitness between the new solution and the existing solution, and denote T as the current temperature. If $\Delta > 0$, we always accept the new solution. If $\Delta \leq 0$, we accept the new solution with a $e^{\Delta/T}$ probability.

6.2 Initial Temperature Value

The initial temperature significantly influences the algorithm's behavior. As [1] described, when the temperature is very high initially, the algorithm tends to accept significantly worse solutions, leading to very random search behavior. Conversely, if the temperature is too low, the exploration will be similar to performing a local search. Therefore, it is crucial to find a balance in calculating this value, depending on the problem's nature and the objective function's values. To this end, we generate a series of random and neighboring solutions and calculate the fitness difference for worse solutions. Using the acceptance function, we set the acceptance probability value (with a high value to encourage random behavior initially) and solve for the temperature value. The resulting equation is shown below:

$$T_0 = \frac{\sum_{i=1}^{n}(f(j) - f(i))}{n \times log(P)} \tag{4}$$

where n is the number of solutions to generate, $f(i)$ is the fitness of the generated solution, $f(j)$ is the fitness of the neighboring solution that worsens, and P is the acceptance probability.

6.3 Neighborhood Environment

The neighborhood environment includes all solutions that differ by exactly one position from the current solution. To create this environment, we randomly select an index in the solution vector and invert its value (for example, changing a 0 to a 1, and vice versa).

6.4 Temperature Decrease

We exponentially decrease the temperature by applying a cooling factor between 0 and 1, following the method described by [1]. We also introduce a variable that triggers cooling at every x iterations, thus allowing for the acceptance of suboptimal solutions over an extended period.

6.5 Reheating

For an equitable comparison with the genetic algorithm, we have integrated a "reheating" mechanism. This process enables the algorithm to continue exploring the solution space and prevents it from halting prematurely due to a low temperature. We preserve the initially calculated temperature and, if the algorithm does not accept any new solution after 500 iterations, we reset the temperature to this stored value and resume the process.

7 Other improvements

We detail here a number of enhancements applied to both the genetic algorithm and simulated annealing (SA).

7.1 Command Line Arguments

We have integrated the *args4j* library to facilitate the input of command-line arguments, which configure the algorithms' parameters. This integration circumvents the necessity of recompiling the code for each configuration adjustment.

7.2 Reading Problem Data Files

We developed a function to dynamically read the data file, conforming to the format specified at [1]. Users must supply the filename via the command line.

7.3 Reproduction of Experiments

For easy experiment replication and to promote equitable algorithm comparison, we introduced a static object in the main class. This object generates pseudo-random numbers and offers the option to set a specific seed. All classes that require random components make use of this object.

8 Experimentation Results

This section details the results of algorithmic tests. We used 15 datasets from [3] and for each of them, we performed 30 trials (seeds 0-29) to calculate the mean and standard deviation of the study metrics, thus allowing the results obtained in the statistical tests to have higher precision [2].

8.1 Parameter Tuning

8.1.1 Genetic Algorithm (GA)

In assessing the genetic algorithm's performance post-enhancements, we explored the impact of varying crossover and mutation operators and their associated probabilities, with a constant population size of 15 individuals. We examined the following parameter values:

- **Crossover Type (TP)**: {SPX (0), UC (1)}
- **Mutation Type (TM)**: {Base (0), Swap (1)}
- **Crossover Probability (PC)**: {0.9, 0.75, 0.5}
- **Mutation Probability (PM)**: {0.01, 0.1, 0.3}

For a comprehensive analysis, we assessed all combinations of these parameters, consisting in total of 36 configurations. Space constraints preclude a full display here; thus, we present a table summarizing the two most and least effective configurations [2].

[1]http://people.brunel.ac.uk/ mastjjb/jeb/orlib/mknapinfo.html

[2]Please consult the webpage to see complete tables https://chrisle7.github.io/

Dataset	PC_0.9_PM_0.01_TC_1_TM_1	PC_0.9_PM_0.3_TC_1_TM_1	PC_0.9_PM_0.01_TC_1_TM_0	PC_0.75_PM_0.3_TC_0_TM_1
problem_mknap1	**3800.0000 ± 0.0000**	**3800.0000 ± 0.0000**	3706.6667 ± 355.1914	3780.0000 ± 109.5445
problem_mknap2	**8387.3633 ± 184.6547**	**8387.3633 ± 184.6547**	8386.1267 ± 127.6494	8318.0500 ± 332.4720
problem_mknap3	**4000.0000 ± 29.3316**	**4000.0000 ± 29.3316**	3950.5000 ± 134.0635	3974.5000 ± 59.3841
problem_mknap4	**5781.1667 ± 455.8900**	**5781.1667 ± 455.8900**	5617.1667 ± 485.8439	5733.6667 ± 368.7255
problem_mknap5	**11791.1667 ± 721.1912**	**11791.1667 ± 721.1912**	11727.6667 ± 653.8108	11454.1667 ± 794.0274
problem_mknap6	10347.2333 ± 248.0582	10347.2333 ± 248.0582	**10377.1000 ± 175.2599**	10288.8667 ± 237.1961
problem_mknap7	**16007.0333 ± 517.0312**	**16007.0333 ± 517.0312**	15885.1000 ± 698.2396	15838.1667 ± 616.5319
problem_mknap8	22921.4667 ± 338.8957	22921.4667 ± 338.8957	23005.5000 ± 451.7111	**23044.8000 ± 315.2338**
problem_mknap9	22891.4667 ± 435.1268	22891.4667 ± 435.1268	22926.7333 ± 346.0643	**23056.3333 ± 410.4359**
problem_mknap10	22413.9667 ± 388.4578	22413.9667 ± 388.4578	22321.3333 ± 372.7821	**22562.2667 ± 343.0913**
problem_mknap11	22441.8333 ± 276.2221	22441.8333 ± 276.2221	22311.3333 ± 342.0147	**22482.7333 ± 412.3892**
problem_mknap12	22773.3333 ± 434.5767	22773.3333 ± 434.5767	**22891.8000 ± 309.0910**	22608.7333 ± 336.5985
problem_mknap13	**23158.4667 ± 454.8408**	**23158.4667 ± 454.8408**	23005.8333 ± 496.0248	23124.1667 ± 435.6782
problem_mknap14	23927.4333 ± 412.3731	23927.4333 ± 412.3731	**24242.7000 ± 341.1307**	23859.8667 ± 461.8916
problem_mknap15	**22180.2000 ± 381.8350**	**22180.2000 ± 381.8350**	21935.1000 ± 396.9149	22099.3000 ± 369.9510

Table 1: Fitness of the Different Configurations of Genetic Algorithm ($\mu \pm \sigma$)

Table 1 shows that the best results are spread across various configurations, with no single configuration consistently outperforming the rest across all problems. We will conduct a statistical study to validate these observations and to determine if significant differences exist between the parameters.

Performing a Shapiro-Wilk test, as reported in Table 2, reveals that the p-values are all below 0.05, leading us to reject the null hypothesis of data normality[3]. Consequently, this precludes parametric tests for comparison.

Conf	Statistic	p-value	Results
PC_0.9_PM_0.01_TC_1_TM_1	0.8032849546444729	0.005857522677763511	Reject H0 with alpha 0.05
PC_0.9_PM_0.3_TC_1_TM_1	0.8032849546444729	0.005857522677763511	Reject H0 with alpha 0.05
PC_0.9_PM_0.01_TC_1_TM_0	0.8115650382419076	0.001717480879046161	Reject H0 with alpha 0.05
PC_0.75_PM_0.3_TC_0_TM_1	0.8011860506790375	0.006906974660481230	Reject H0 with alpha 0.05

Table 2: Shapiro-Wilk test (significance level of 0.05)

We applied the Friedman test to identify whether significant differences exist, and the results are presented in Table 3. No significant differences detected, so we selected the configuration with **Crossover Type**: UC, **Mutation Type**: Swap, **Crossover Probability**: 0.9, and **Mutation Probability**: 0.01 to compare with simulated annealing.

Statistic	p-value	Result
29.35945945945946	0.7367383661222924	No Reject H0 with alpha 0.05

Table 3: Results Friedman test (significance level of 0.05)

8.1.2 Simulated Annealing (SA)

As for simulated annealing, the parameter under variation is the temperature decay rate. Four potential values will be examined: 0.5, 0.75, 0.9, and 0.99. The number of iterations for cooling has been set to 5, and the number of iterations for generating the initial temperature is set to 10.

[3]Summarized table - Complete table available at https://chrisle7.github.io/

Dataset	0.5	0.75	0.9	0.99
problem_mknap1	3800.0000 ± 0.0000	3800.0000 ± 0.0000	3800.0000 ± 0.0000	**3800.0000 ± 0.0000**
problem_mknap2	8706.1000 ± 0.0000	8706.1000 ± 0.0000	8706.1000 ± 0.0000	**8706.1000 ± 0.0000**
problem_mknap3	3999.0000 ± 20.0603	4008.0000 ± 7.9438	4014.3333 ± 2.5371	**4015.0000 ± 0.0000**
problem_mknap4	5952.5000 ± 94.3923	6027.6667 ± 67.1942	6054.5000 ± 52.1330	**6099.6667 ± 19.9107**
problem_mknap5	11788.3333 ± 183.3422	11864.6667 ± 200.8513	12082.8333 ± 180.2713	**12276.5000 ± 114.5843**
problem_mknap6	10391.5667 ± 83.0146	10365.6333 ± 60.3915	10425.2333 ± 75.2553	**10488.6000 ± 68.0657**
problem_mknap7	15789.5333 ± 250.0175	15785.8000 ± 190.5094	15991.8000 ± 186.4498	**16172.8667 ± 176.4250**
problem_mknap8	20324.2333 ± 385.4660	20197.0667 ± 375.7584	20330.0000 ± 439.3204	**21228.7333 ± 367.9183**
problem_mknap9	20162.0333 ± 368.4218	20121.8667 ± 312.6166	20207.7000 ± 378.5655	**21011.6333 ± 403.8887**
problem_mknap10	19128.6333 ± 370.9124	19104.7000 ± 399.7310	19359.6667 ± 362.1584	**20132.4333 ± 290.8252**
problem_mknap11	19864.7333 ± 331.2741	20026.4667 ± 327.0370	20165.1000 ± 330.5027	**20796.5000 ± 213.1111**
problem_mknap12	19889.1333 ± 438.2063	19891.6000 ± 338.0935	20230.6000 ± 326.8718	**21031.6667 ± 454.5438**
problem_mknap13	20265.3667 ± 292.8914	20189.5333 ± 311.2098	20627.9000 ± 437.9512	**21326.1667 ± 265.1401**
problem_mknap14	20776.4333 ± 309.4471	20778.0000 ± 355.0920	21093.6667 ± 441.7886	**22183.8333 ± 374.7711**
problem_mknap15	19506.8667 ± 315.8788	19575.0333 ± 333.3518	19718.1000 ± 281.6777	**20367.3000 ± 290.3934**

Table 4: Fitness of the Different Configurations of Simulated Annealing ($\mu \pm \sigma$)

The Table 4 presents the average fitness results for each problem, considering 30 repetitions. At first glance, $DT = 0.99$ appears to provide the best performance, often also exhibiting lower standard deviation. To confirm the presence of significant differences between the parameters, a statistical study will be conducted.

When performing the Shapiro-Wilks test (see Table 5), it is evident in all cases that the p-value is less than 0.05, indicating that the data do not follow a normal distribution. This result precludes the use of parametric tests for comparison.

DT	Statistic	p-value	Results
0.5	0.8085108579494737	0.0032445710252630953	Reject H0 with alpha 0.05
0.75	0.8064721373423671	0.004263931328816414	Reject H0 with alpha 0.05
0.9	0.8094566687108026	0.002771665644598683	Reject H0 with alpha 0.05
0.99	0.816295525722131	0.0006477628610655484	Reject H0 with alpha 0.05

Table 5: Shapiro-Wilk test (significance level of 0.05)

Applying the Friedman test [5] to check for significant differences yields the results shown in Table 6. The identification of significant differences allows for the use of the Nemenyi test [2] to ascertain the relationships between the algorithms. The findings, displayed in Figure 1, lead to the conclusion that the best configuration is $DT = 0.99$. This setting not only shows a lower standard deviation but also significant differences compared to 0.75 and 0.5, albeit not with 0.9.

Statistic	p-value	Result
30.44	1.115152622910287e-06	Reject H0 with alpha 0.05

Table 6: Results Friedman test (significance level of 0.05)

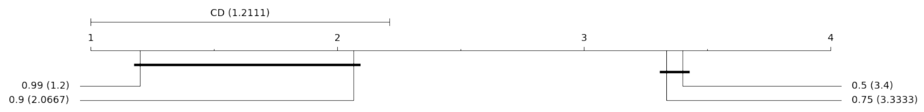


Figure 1: Results Nemenyi test (significance level of 0.05)

8.2 Algorithm comparison

Once we have adjusted the hyperparameters of the different algorithms, we can proceed to the comparative study. Table 7 presents the results for the optimal configurations of both the genetic algorithm and simulated annealing, detailing each problem's global optimum [4] and the corresponding execution time in milliseconds. For a balanced comparison, we capped the number of evaluations at 50.000. The results show that both algorithms perform similarly, although GA occasionally excels when the global optimum is unknown. Conducting a Wilcoxon test (see Table 8) reveals significant fitness disparities favoring GA. Yet, in terms of computation time, and at a significance level of 0.05, we detect no significant differences when evaluations are limited to 50.000.

Dataset	Optimum	GA	SA	Time GA	Time SA
problem_mknap1	3800	**3800.0000 ± 0.0000**	**3800.0000 ± 0.0000**	**40.2000 +- 6.3594**	44.1000 +- 7.5263
problem_mknap2	8706.1	8387.3633 +- 184.6547	**8706.1000 ± 0.0000**	218.4000 +- 59.6586	**72.7000 +- 17.8483**
problem_mknap3	4015	4000.0000 +- 29.3316	**4015.0000 ± 0.0000**	173.5000 +- 104.2579	**118.0000 +- 33.6216**
problem_mknap4	6120	5781.1667 +- 455.8900	**6099.6667 ± 19.9107**	262.3000 +- 41.6199	**228.5333 +- 38.0524**
problem_mknap5	12400	11791.1667 +- 721.1912	**12276.5000 ± 114.5843**	286.4333 +- 19.9390	**277.1000 +- 46.8338**
problem_mknap6	10618	10347.2333 +- 248.0582	**10488.6000 ± 68.0657**	282.2667 +- 13.8886	**278.0333 +- 17.6937**
problem_mknap7	16537	16007.0333 +- 517.0312	**16172.8667 ± 176.4250**	313.9333 +- 16.4420	**306.8667 +- 29.6575**
problem_mknap8	?	**22921.4667 +- 338.8957**	21228.7333 ± 367.9183	**387.5333 +- 22.7425**	397.4333 +- 28.7986
problem_mknap9	?	**22891.4667 +- 435.1268**	21011.6333 ± 403.8887	**383.7000 +- 22.9289**	386.8000 +- 29.8657
problem_mknap10	?	**22413.9667 +- 388.4578**	20132.4333 ± 290.8252	383.6000 +- 29.3453	**372.0333 +- 27.4144**
problem_mknap11	?	**22441.8333 +- 276.2221**	20796.5000 ± 213.1111	**390.1333 +- 30.9502**	400.0667 +- 21.5101
problem_mknap12	?	**22773.3333 +- 434.5767**	21031.6667 ± 454.5438	382.8667 +- 23.6450	**377.2333 +- 24.5127**
problem_mknap13	?	**23158.4667 +- 454.8408**	21326.1667 ± 265.1401	388.6000 +- 38.3024	**382.2000 +- 31.1453**
problem_mknap14	?	**23927.4333 +- 412.3731**	22183.8333 ± 374.7711	**333.8333 +- 21.9609**	376.6000 +- 29.7664
problem_mknap15	?	**22180.2000 +- 381.8350**	20367.3000 ± 290.3934	379.6000 +- 41.6782	**336.5667 +- 50.0301**

Table 7: Comparison of average fitness of algorithms vs. known global optimum

Score	Statistic	p-value	Result
Fitness	21	0.047989749694959505	Reject H0 with alpha 0.05
Time	32	0.1117687451396760525	No reject H0 with alpha 0.05

Table 8: Results Wilcoxon test (significance level of 0.05)

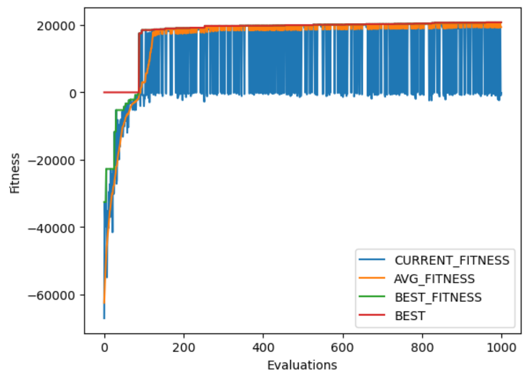
Next, a study will be carried out for one of the problems (problem_mknap12) studying how the algorithms behave over time. In this case, the number of evaluations will be modified, to determine the behavior of the algorithms in the case that the number of evaluations is less than the one used for the statistical tests.

Limiting the number of maximum evaluations to one thousand for seed 1 (see Figure 2), reveals distinct behaviors between the algorithms.

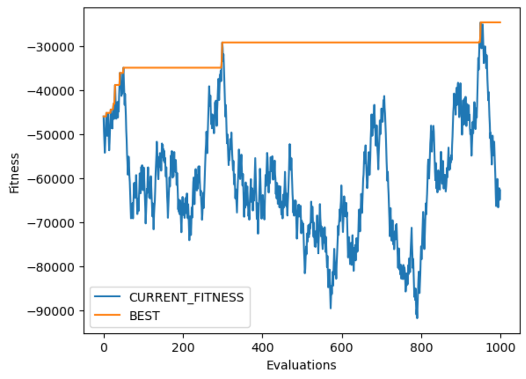
[4] "?" indicates an unknown global optimum

The genetic algorithm swiftly locates solutions that comply with the problem's constraints, often achieving strong fitness scores after just 150 evaluations. The population's average fitness gradually increases due to elitism, as each iteration culls the least fit solutions. Although average fitness may dip occasionally, it typically rebounds when the lower-quality solutions are removed. There is no consistent pattern in the quality of solutions produced; sometimes they improve, while at other times, they are suboptimal or even violate the constraints. The algorithm appears to be nearing a convergence phase, with the population's average fitness closely mirroring that of the best individual.

For simulated annealing, the random component hinders its ability to consistently yield solutions that fulfill the problem's constraints, sometimes resulting in negative fitness values. However, given the stochastic nature of both methods, such outcomes are not guaranteed.



(a) Algoritmo Genético

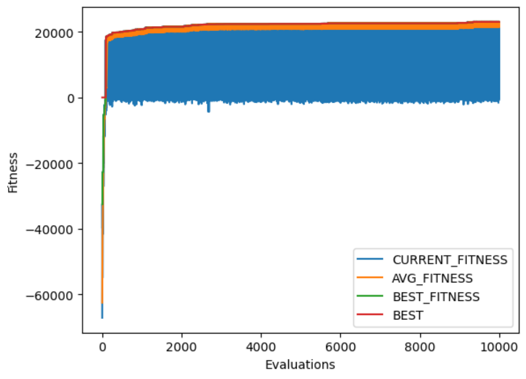


(b) Enfriamiento Simulado

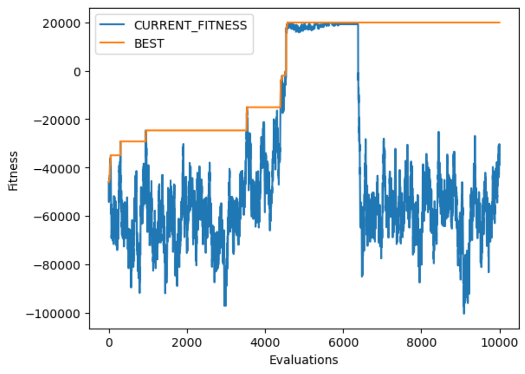
Figure 2: Fitness evaluation on problem problem 12 (seed 1) for the first 1.000 evaluations

Increasing the evaluation limit to ten thousand, Figure 3 demonstrates that the genetic algorithm maintains its performance pattern observed in previous tests.

In simulated annealing, from around iteration 4.000, we observe an ascent in solution quality as the temperature drops, preventing acceptance of inferior solutions. This shift marks the algorithm's entry into a specification phase, seeking the global optimum. By iteration 6.000, the temperature decreases to a point where the algorithm no longer accepts worse solutions and fails to find improvements, triggering a reheat after 500 iterations without solution acceptance, hence returning to a generalization phase.



(a) Algoritmo Genético

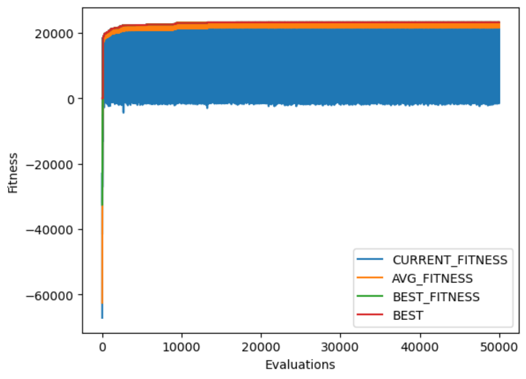


(b) Enfriamiento Simulado

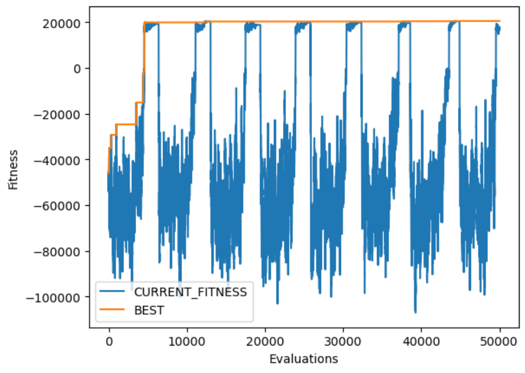
Figure 3: Fitness evaluation on problem problem 12 (seed 1) for the first 10.000 evaluations

Subsequent graphs will present the results with 50.000 maximum evaluations (see Figure 4), where both algorithms continue to exhibit trends consistent with the prior observations, indicating sustained performance.

Note that in simulated annealing without reheating, the likelihood of accepting worse solutions diminishes over time. This extended running can lead to deadlock. If the algorithm were to halt only upon discovering the global optimum, it might enter an endless loop.



(a) Algoritmo Genético



(b) Enfriamiento Simulado

Figure 4: Fitness evaluation on problem problem 12 (seed 1) for the first 50.000 evaluations

9 Conclusions

The present work has helped the student to better understand the operation of the meta-heuristic algorithms studied in the course. In addition, it has allowed a better understanding of the functioning of hypothesis tests.

Statistical analysis has shown that with a number of 50.000 evaluations, significant fitness differences emerge; time considerations, however, do not show discrepancies. With fewer evaluations (5.000), these fitness differences become even more pronounced, likely due to simulated annealing not entering the search process's intensification phase.

It is important to recognize that the genetic algorithm's use of simple generational replacement can cause premature convergence. Future research should investigate alternative replacement strategies, employ "restarts" to increase population diversity, and dynamically adjust crossover and mutation probabilities in response to the population's diversity and proximity to the optimal solution. This would enhance exploration within the search space. Additionally, parallelizing the evolutionary algorithm's evaluation phase may decrease the time required for population creation.

Bibliografía

[1] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing.* John Wiley & Sons, Inc., 1989.

[2] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *The Journal of Machine learning research* 7 (2006), pp. 1–30.

[3] *Multidimensional knapsack problem - OR-Library.* URL: `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html`. (accessed: 18-11-2023).

[4] *NEO - Our Software - ssGA: Steady State GA.* 2008. URL: `https://neo.lcc.uma.es/software/ssga/`. (accessed: 18-11-2023).

[5] *StaTDS - Library for statistical testing and comparison of algorithm results.* URL: `https://github.com/kdis-lab/StaTDS`. (accessed: 20-12-2023).