



Máster universitario en investigación en inteligencia artificial

Resolución de problemas con metaheurísticos

Comparison between population based and trajectory algorithms

Miguel Ángel Contreras Córdoba

Córdoba - 2 de febrero de 2026

Índice

1	Introduction	3
2	Evolutionary Algorithm	5
3	Simulated Annealing	6
4	Results	8

1. Introduction

In this work we are going to be doing a comparison between two algorithms: Evolutionary Algorithm (EA) and Simulated Annealing (SA) to solve the Quadratic Unconstrained Binary Optimization (QUBO) problem [1]. In this introduction we will define the QUBO problem and the methodology used.

The QUBO problem is defined as follows: given a value n and a symmetric matrix W of size $n \times n$, find an n -bit vector X that maximizes the Energy $E(X)$. The energy function used to evaluate the solutions is the Upper Triangular Version proposed by the Hiroshima University [2] (Formula 1.1):

$$E(X) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} W_{i,j} x_i x_j \quad (1.1)$$

To show an example: given a matrix of size 4×4 :

$$\begin{pmatrix} 1 & -2 & 0 & 0 \\ -2 & 3 & -1 & 0 \\ 0 & -1 & 2 & -3 \\ 0 & 0 & -3 & 1 \end{pmatrix}$$

There are 2 solutions that maximize the energy, therefore the optimum solutions:

- $\{0 \ 1 \ 0 \ 1\}$
- $\{0 \ 1 \ 1 \ 0\}$

To obtain the instance that would be used as an input for our algorithms, we tried downloading the 1024 bits Random Instance from [3] as suggested. Unfortunately, the URL to download the instances was down and it could not be downloaded. We decided to generate our own random instance, since the desired instance was also a randomly generated instance. We generated a CSV file where each row contains the values of i and j followed by the value of the matrix in that position. We also did not include in this file those pair which value was zero and the values were generated with a density of 5 %, meaning that 95 % of the values were zero. This was done to save memory and computational time. Lastly, the random values generated for each entry of the matrix are in the range $[-10, 10]$

As stated before, for this work we are comparing two algorithms:

- The EA [4] belongs to the category of trajectory algorithms, which establishes some strategies to go over the search space iteratively from an initial solution. All this strategies are explained in section 2.
- The SA [5] belongs to the category of population based algorithms, which established strategies are explained in section 3.

In the results section (Section 4.2), we have made a comparison between this two algorithm. This comparison has been made using different maximum number of evaluation as the stopping criteria. Due to the size of the problem, the calculation of the optimum value is not feasible.

2. Evolutionary Algorithm

This section will be focused in showing and explaining the pseudocode of the algorithm EA. The following pseudocode (Pseudocode 1) is based on the code provided:

Algorithm 1 Evolutionary Algorithm

```
1: start
2:  $eval \leftarrow 0$ ;  $P(eval) \leftarrow generateInitialPopulation()$ 
3:  $evaluatePopulation(P(eval))$  // Evaluate every individual of the population
4: repeat
5:    $p_1 \leftarrow selectParent(P(eval))$  // Select parent from the population
6:    $p_2 \leftarrow selectParent(P(eval))$  // Select parent from the population
7:    $child \leftarrow recombination(p_1, p_2)$  // Crossover
8:    $child \leftarrow mutation(child)$  // Mutation with a probability
9:    $evaluate(child)$  // Evaluate the child
10:   $eval \leftarrow eval + 1$ 
11:   $P(eval) \leftarrow replacement(P(eval - 1), child)$  // Build the new poblation
12: until ( $StopCriteria$ )
13: output Best solution in  $P(t)$ 
```

Now lets explain some of the key functions that appear in this pseudocode:

- **generateInitialPopulation()**: In this step an initial population is generated from which the algorithm will try to find the optimum. This populations was created by randomly generating individuals until reaching the population size.
- **selectParent()**: This function takes the current population as input and performs a Binary Tournament. This strategy consist on taking two random candidates from the population and select the best candidate.
- **recombination()**: For this function, the Single Point Crossover was the strategy used. This strategy generates a child by taking the parents, selecting a common random point inside them and combining the first part of one of the parents with the second part of the other.
- **mutation()**: The mutation strategy use in our case is Bit Flip Mutation. In this strategy a we take the child and go through each of the bits and change the bit value with a specified probability.
- **replacement()**: Lastly, the new population is generated by adding the child the current population and eliminating the worst individual. This strategy is called Elitist Replacement

3. Simulated Annealing

This section will be focused in showing and explaining the pseudocode of the algorithm SA. The following pseudocode (Pseudocode 2) is based on the code provided:

Algorithm 2 Simulated Annealing

```
1: start
2:  $S \leftarrow \text{generateRandomIndividual}()$ 
3:  $S_{\text{energy}} \leftarrow \text{evaluate}(\text{solution})$  // Evaluate the current solution
4: repeat
5:    $N \leftarrow \text{generateNeighbour}(\text{solution})$  // Generate a neighbour
6:    $N_{\text{energy}} \leftarrow \text{evaluate}(\text{neighbour})$  // Evaluate the neighbour
7:    $\text{delta} \leftarrow \text{neighbourenergy} - \text{solutionenergy}$  // Compare solutions
8:   if  $\text{accept}(\text{delta})$  then
9:      $\text{solution} \leftarrow \text{neighbour}$  // Accept the neighbour
10:  end if
11:   $\text{temperature} \leftarrow \text{temperature} \cdot \alpha$  // Modify the temperature
12: until ( $\text{StopCriteria}$ )
13: output  $\text{solution}$ 
```

Now let's explain some of the key functions that appear in this pseudocode:

- **generateNeighbour()**: This algorithm generates a neighbour from the current solution with the objective of improving the current solution. To generate a neighbour of the current solution, we modify a random byte from the solution.
- **accept()**: To accept a neighbour as the new solution we calculate the difference between the current solution energy and the neighbour energy (Formula 3.1):

$$\Delta E = N_{\text{energy}} - S_{\text{energy}}. \quad (3.1)$$

After calculating this, if $\Delta E \geq 0$, we accept the solution, since we always accept solutions that improve the current solution. If, on the contrary, $\Delta E < 0$, we accept the solution with a probability calculated with the following formula (Formula 3.2):

$$P_{\text{acceptance}} = e^{\frac{\Delta E}{T}} \quad (3.2)$$

where T is the temperature. At the beginning of the algorithm, when the temperature is high, the probability of accepting a neighbour that does not improve the solution is high, which coincides with our intention of exploring the neighbourhood to find distant solutions and avoid the local optimums. Later, when the temperature is low, the probability of acceptance will also be low, helping us exploit the neighbourhood to find the best solution around the current one.

- **Cooling schedule**: As the pseudocode shows, the temperature decreases with every iteration of the algorithm. The initial temperature will be a high value that

will due to the cooling schedule following this formula (Formula 3.3):

$$T = T \cdot \alpha \tag{3.3}$$

where α is a float number ranging from 0.3 to 0.99 depending on the problem.

4. Results

For this work we have a QUBO problem of size $n = 1024$ with the format explained in Section 1. The encoding of this problem is a binary vector with a length n . Each vector's position can either be a 1 or a 0, which will influence if a value of the matrix is considered to calculate the final energy or not. The parameters used for the experimental study are the following (Table 4.1):

Cuadro 4.1: *Parameters set used*

EA	Population size	[20, 50, 100, 200, 500, 1000]
	BitFlip Probability	[0.0005, 0.001, 0.002, 0.005, 0.01]
SA	Initial Temperature	[10, 50, 100, 200, 500, 1000, 2000, 5000]
	Cooling Rate	[0.7, 0.8, 0.9, 0.99]
Both	Max Iterations	[50000, 75000, 100000]

The results shows here, since it is not practical obtain the global optimum due to the problem size, we are using maximum iterations as the algorithm stop criteria. In our case, we have tested 50000, 75000 and 100000 maximum iterations. Both algorithm tested has random processes, that's way we have launched each algorithm 10 times for each parameters combination, all 10 with different seeds from 0 to 9. All the results obtain on this tests are appended to this report. From all this tests we are going to show the best combination for each algorithm. The results shown in Table 4.2 show the mean and standard variation for the Energy and Time of the 10 seeds:

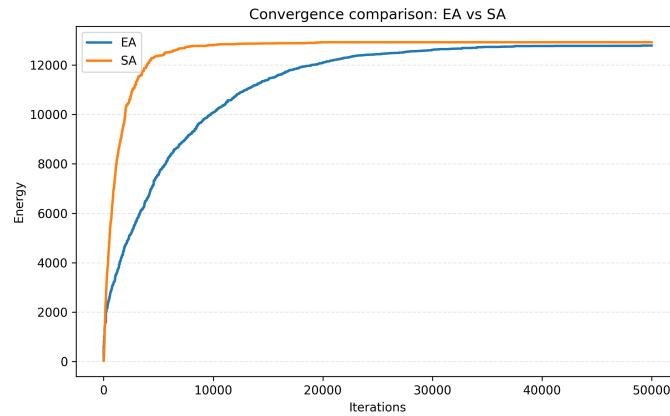
Cuadro 4.2: *Best results for 50000 iterations*

	EA	SA
	Population = 50	Initial Temperature = 1000
	Bit Flip Probability = 0.001	$\alpha = 0.9$
Energy	12848.50 \pm 57.62	12907.70 \pm 54.00
Time (s)	4.416 \pm 0.243	3.070 \pm 0.384

In this experiment we can see that the that SA gets both higher Energy and takes less Time than EA. We have choose 50000 iterations since we can see in the results that for all maximum iterations we get the same exact values. This tells us that the algorithms are converging before reaching the iterations limit. To test this, we are going to get a random seed and execute both algorithms with it using the parameters from Table 4.2. We can see the results in the following chart (Figure ??):

As we can see in the convergence chart, apart from getting to a slightly higher energy and do it so in a less time, we can also see that SA converges much faster than the EA.

Figura 4.1: *Convergence chart*



We can now conclude that SA performs way better since it reaches an optimal values both higher and faster that EA. This could be further tested by using bigger problem sizes but that falls outside of this works scope, though it can be considered for a future line of work.

Referencias

- [1] Quadratic unconstrained binary optimization, 2026. [Online; accessed 26-January-2026].
- [2] Qubo (quadratic unconstraint binary optimization), 2022. [Online; accessed 26-January-2026].
- [3] Qubo instance examples, 2022. [Online; accessed 26-January-2026].
- [4] R. M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13, 1958.
- [5] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5):975–986, Mar 1984.