A Genetic Algorithm for the Maximum Independent Set Problem

David Fandrei

Master 2 GENIOMHE,
Université d'Evry Val d'Essonne,
Evry-Courcouronne, France

Summary

Metaheuristics are an effective strategy to give approximate solutions for NP-hard problems. Genetic Algorithms, a subcategory of evolutionary heuristics, have been successfully applied to combinatorial optimisation problems. Here, a Genetic Algorithm is implemented in Python, such that it is easy to use whilst maintaining full control of its hyper-parameters. This algorithm was subsequently applied to the Maximum Independent Set problem (MISP) in undirected and unweighted graphs. Since the MISP is a constrained optimisation problem, a weighted penalty was introduced to the fitness function to address the issue of invalid solutions at each population that is tunable. The Genetic Algorithm is tested on a simple undirected graph. [1]

Keywords: Metaheuristics, Genetic Algorithm, Tabu Search, Combinatorial Optimisation, Maximum Independent Set

1. Introduction

A lot of naturally occuring optimisation problems of which the objective function is subject to constraints are **NP-hard**. That means the theorem $\boldsymbol{P \neq NP}$ applies to them. Once this is proven for a given problem, an algorithm that finds the optimal solution in polynomial runtime does not exist. Thus the search for an efficient heuristic of which the solution converges to the global optimum with available computational ressources is necessary.
A group of algorithms that finds a sufficiently good solution to problems of combinatorial optimisation are *Metaheuristics*. A common feature of different kinds of *Metaheuristics* is that they use a certain degree of stochasticity (Luke (2013)). In this report, a Genetic Algorithm, a population-based method that imitates the principles of naive Darwinian evolution, and Tabu search, a single-solution algorithm with problem-tailored constraints to guide the search space, are compared to solve the *Maximum Independent Set Problem* (MIS).

2. The Maximum Independent Set Problem

Graph Theory is a field of discrete mathematics which is of particular interest to computer science due to its capacity to model many problems in engineering, biological network science, cheminformatics or even transport geography.

In 1972, Karp introduced twenty-one NP-hard problems. One of those was the Maximum Independent Set Problem Karp (1972). In graph theory, an independent set S of a given graph $G = (V, E)$ is a number of vertices $v \in V$ such that no two vertices in S are adjacent, i.e. that no edges $e \in E$ are shared. *The Maximum Independent Set* (MIS) is the largest possible set of G. It can be defined as :

— $V' \subseteq V$ such that $\forall i, j \in V'$ the edge $\langle i, j \rangle \notin E$ and $|V'|$ is maximum (Back and Khuri (1994)).

The MIS is deemed NP-hard. Furthermore, the MIS is the complementary to another NP-hard graph problem : the *Minimum Vertex Cover*. It can be stated for any $G = (V, E)$ and $V' \subseteq V$:

— $\boldsymbol{V'}$ is the MIS of G,
— $\boldsymbol{V - V'}$ is the Minimum Vertex Cover of G.

3. Genetic Algorithms

Genetic Algorithms (GA) are evolutionary heuristics. The basic concept was designed to be similar to the mechanics of naive Darwinian evolution :

3.1. Basic concept

The first step is to initialise a random set of solutions of a predefined size, called population. Each solution is called a genome (sometimes also referred to as chromosome). On each iteration step, the "fitness", i.e. an evaluation score of the objective function on each genome, is compared. Therefore, it is of primordial importance to choose a problem-tailored representation of the solution. In the case of the

1. This work has been conducted as coursework for the *Algorithms and Combinatorial Optimisation* course as part of the Master 2 GENIOMHE of Université d'Evry Val d'Essonne. It was supervised by Prof. Franck Delaplace.

MIS, a 0-1 (1 meaning a vertex is included in the solution and 0 not included) seems obvious. For that reason, each genome consists of an n-length bit stream where n equals the number of total vertices of any given G.
In the next step, a new generation of offspring is created from the individuals of the parent population. During this step, a method similiar to natural selection is applied such that the individual genomes are replicated with a rate proportional to their fitness score. There are different computational methods to implement this idea, e.g. Rank selection, Biased Wheel, Tournament Selection, etc. As an alternative to the standard GA, a certain degree of elitism, i.e. the injection of the best individuals directly to the new generation, can be applied. In this implementation, two genomes of the parent population are selected for the following steps. One consideration that needs to be taken is whether to include only viable individuals (thus those for which the constraints of the initial problem apply) or also false results with high fitness. In order to avoid bias towards initial good individuals and the risk of convergence to a local maximum, I decided to include false results. However, the fitness function is adjusted such that the number of false individuals correlates to a penalty value subtracted from the fitness score (see *Fitness Function*).
Then, single-point crossover is performed. Crossover is the GA's distinguishing feature (Luke (2013)). Bit scores are exchanged at a random position of the genome. Furthermore, mutation through 0-1 bit-flip at a predefined rate is applied. This is repeated until the child population is completely filled.
The number of iterations can be customised. The individual with the highest fitness is stored on each iteration to visualise convergence. The final result of the algorithm corresponds to the highest ranked individual of the ultimate generation.

3.2. Python Implementation

The Genetic Algorithm was implemented in Python. The corresponding files can be found in the adjacent GitHub repository (see *Code Availability*).
The concept of the Genetic Algorithm was applied in a way it is both easy to use whilst providing the user the capacity to control the hyperparameters. It is possible to specify the genome size for each individual in a population as well as population size (number of individuals in a population). Furthermore, one can choose different types of selection methods : Rank selection, Tournament selection and Roulette-Wheel-Selection. So far, only Single-Point-Crossover is implemented. One can select mutation rate (probability) to ensure optimal convergence of the algorithm. At last, using the provided wrapper function, it is possible to apply elitism at each selection step, i.e. the two individuals with the highest fitness value are automatically transmitted to the next generation. The final solution of the algorithm is the individual of the last generation with the highest fitness score. The individual functions can be run one after the other or by using the wrapper function provided. Taken this into consideration, the algorithm can not only be applied to the MISP but to a wide array of NP-hard problems, making it a versatile tool to estimate such questions in polynomial runtime.

3.3. Algorithm Outline

Algorithm 1 Genetic Algorithm

```
 1: $populationsize \leftarrow$ set population size
 2: $P \leftarrow \{\}$
 3:
 4: for $populationsize$ times do
 5:     $P \leftarrow P \cup$ {new random individual}
 6: end for
 7:
 8: repeat $n - times$            ▷ define number of populations
 9:     for each individual $P_i \in P$ do
10:         FitnessFunc($P_i$)
11:     end for
12:     $Q \leftarrow \{\}$                    ▷ define child population
13:     for $populationsize/2$ times do
14:         $P_a \leftarrow$ SelectionFunc($P$)
15:         $P_b \leftarrow$ SelectionFunc($P$
16:         $C_a, C_b \leftarrow$ CrossoverFunc($P_a$, $P_b$)
17:         $Q \leftarrow Q \cup$ {MutateFunc($C_a$), MutateFunc($C_b$)}
18:         $P \leftarrow Q$
19:     end for
20: until $n$ is reached
21: best = □
22: for $n^{th}$ population do
23:     for each individual $P_i \in P$ do
24:         if FitnessFunc($P_i$) > best then
25:             best $\leftarrow P_i$
26:         end if
27:     end for
28: end for
29: return best
```

4. Fitness Function

4.1. Penalty Method

Genetic Algorithms are usually designed to handle unconstrained optimisation problems Kuri-Morales and Gutiérrez-García (2002). Since the MISP is a constrained problem, however, a strategy to treat invalid solutions (individuals) at each population must be adapted. Here, a penalty function was chosen. As the name suggests, the fitness values of invalid solutions are penalized instead of entirely discarded at each generation. I introduce the variable ϕ as weight to this penalty :

— For a Graph $G(V, E)$ and a solution $C \subseteq V$ produced by the GA :

$$Maximise \sum_{i \in C} x_i - \sum_{\forall i,j \in C; \langle i,j \rangle \in E} \phi x_{i,j} \tag{1}$$

5. Results

First, a small graph was created and the Genetic Algorithm as well as a Brute Force solution were run in order to compare the capacity to detect the MIS in simple applications (*see Figures 1, 2, 3*). As weight for the penalty of invalid results, I chose

$$(\sum v \in V)^{-1}$$

. The Genetic Algorithm was able to find the MIS in this instance, proving its overall utility for this problem.

6. Discussion

Here, I suggest a Genetic Algorithm implemented in Python and fully customizable for a wide array of NP-hard problems. I applied it to a simple graph in order to find an approximate solution to the Maximum Independent Set Problem.
I introduce a fitness function with a weighted penalty that can be tuned for more complicated problems. Like this, common Machine-Learning Hyperparameter optimization techniques like grid-search or random search can be used in order to find an optimal value and to allow the algorithm to converge fast.

Code Availability

The corresponding code including the script containing the Genetic Algorithm as well as a Notebook with solutions for a small graph and benchmark graphs can be found in the following GitHub repository :
https ://github.com/davidfdr99/GeneticAlgo_MISP

Appendix

Références

Back, T., Khuri, S., 1994. An evolutionary heuristic for the maximum independent set problem, in : Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, pp. 531–535 vol.2. doi :`10.1109/ICEC.1994.350004`.

Karp, R.M., 1972. Reducibility among combinatorial problems., in : Miller, R.E., Thatcher, J.W. (Eds.), Complexity of Computer Computations, Plenum Press, New York. pp. 85–103. URL : `http://dblp.uni-trier.de/db/conf/coco/cocc1972.html#Karp72`.

Kuri-Morales, A.F., Gutiérrez-García, J., 2002. Penalty function methods for constrained optimization with genetic algorithms : A statistical analysis, in : Coello Coello, C.A., de Albornoz, A., Sucar, L.E., Battistutti, O.C. (Eds.), MICAI 2002 : Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 108–117.

Luke, S., 2013. Essentials of Metaheuristics. second ed., Lulu. Available for free at http ://cs.gmu.edu/∼sean/book/metaheuristics/.

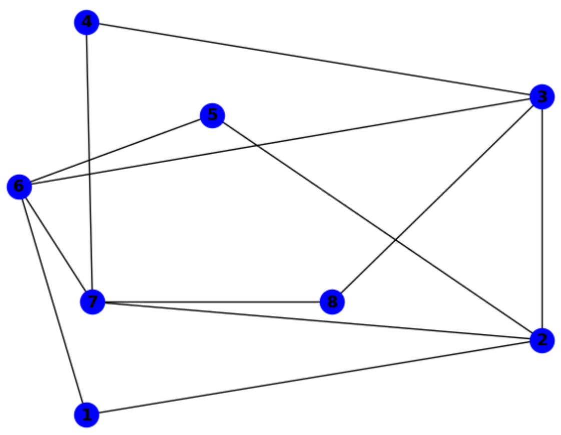

FIGURE 1 – A small graph with 8 edges :
(1, 2), (1, 6), (2, 3), (2, 5), (2, 7), (3, 6), (3, 8), (3, 4), (4, 7), (5, 6), (6, 7), (7, 8)

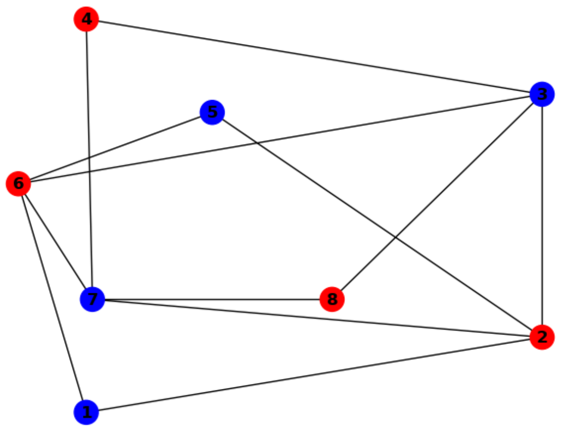

FIGURE 2 – Solution of the Brute Force Algorithm for the small graph.

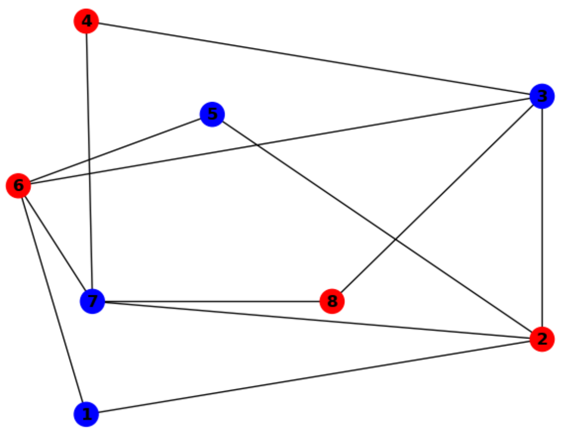

FIGURE 3 – Solution of the Genetic Algorithm for the small graph.