




Genetic algorithm for black and white coloring problem on graphs

Algoritmo genético para el problema de coloración blanco y negro en gráficas

Luis Eduardo Urbán Rivero¹ , Javier Ramírez Rodríguez² , Rafael López Bracho² 
lurbanrivero@gmail.com, jararo@azc.uam.mx, rlb@azc.uam.mx

¹ Posgrado en optimización UAM Azcapotzalco.

² Departamento de Sistemas, UAM Azcapotzalco.

Av. San Pablo 180 Col. Reynosa-Tamaulipas Delegación Azcapotzalco C.P. 02200, Ciudad de México.

KEYWORDS:

Anticoloring, Genetic algorithm, BWC.

ABSTRACT

A classical problem in graph theory and combinatorial optimization is the known as Graph Coloring Problem (GC). This problem consists in assigning colors to vertices of a graph such that two adjacent vertices must have different colors. The objective in this problem is to find the minimum number of colors needed to coloring the graph under the coloring condition. In the case of the Graph Anti-Coloring Problem (GAC) the color assignment is opposite. In this case two adjacent vertices must have the same color or one of them does not have any color. In this problem the objective is different in the sense of number of colors which is fixed. This problem can be turned in an optimization problem. The GAC is NP-Complete, even with two colors[2]. In this work we deal with two colors special case of GAC with genetic algorithms and compare with Tabú Search which is the state of the art solution for this problem[3].

PALABRAS CLAVE:

Anticoloración, Algoritmo genético, BWC.

RESUMEN

Un problema clásico en la teoría de gráficas es el conocido como problema de coloración (GC por sus siglas en inglés). Este problema consiste en asignar colores a los vértices de la gráfica tal que vértices adyacentes deben tener colores diferentes. El objetivo de este problema es encontrar el mínimo número de colores necesarios para colorear la gráfica bajo la mencionada condición de coloración. En el caso del problema de anticoloración (GAC por sus siglas en inglés), la forma de asignar color es opuesta. En este caso vértices adyacentes deben tener el mismo color o al menos uno de ellos no debe tener color. En este problema el objetivo es diferente en el sentido del número de colores que en este caso es fijo. Este problema puede ser convertido en un problema de optimización. El GAC es NP-Completo aun cuando se usen sólo dos colores [2]. En este trabajo trataremos con el caso especial de dos colores del GAC mediante el uso de algoritmos genéticos y se compara con la búsqueda Tabú que es la mejor técnica conocida para la solución de este problema [3].

Recibido: 11 de julio del 2017 • Aceptado: 20 de octubre del 2017 • Publicado en línea: 28 de febrero del 2018

1. INTRODUCTION

In the 70s Berge proposed the following problem. If we have a $n \times n$ chessboard, b black queens and w white queens with b, w positive integers. Is it possible to place b black and w white queens so that black and white queens do not attack each other?[4]. The complexity of this problem remains open. Later in 1979 Lipton and Tarjan tried to solve this question but they failed. They used the vertex separator theorem[6] to solve it but they could only solve the case of the tower piece. Berend et al., propose the concept of anti-coloring as a generalization of the vertex separator theorem[2], due the vertex separator theorem works only with two colors.

Definition 1 (Anti-coloring Condition). An anti-coloring of the vertices of a graph $G = (V, E)$ is a partial coloring of the vertices $a : V \rightarrow S \cup \text{uncolored}$, where S is the set of colors, such that, for each $\{v, w\} \in E$, $a(v) = a(w)$ or at least one of them is uncolored.

Problem 1.1 (GAC).

Input: Given a simple and undirected graph $G = (V, E)$ and positive integers b_1, b_2, \dots, b_k .

Output: 1 If G has an anticoloring of V with b_i vertices of color i , for all $i \in \{1 \dots k\}$, 0 otherwise.

Example 1. Given the graph of the FIGURE 1 and positive integers $b_1 = 2$ (reds), $b_2 = 3$ (greens) and $b_3 = 2$ (blues), a valid anti-coloring can be viewed in FIGURE 1

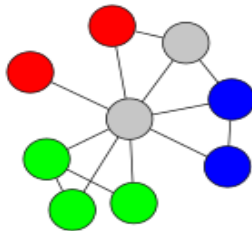


FIGURE 1: EXAMPLE OF THREE COLOR ANTI-COLORING (GRAY IS UNCOLORED)

If limited $k = 2$. We have the Black and White coloring problem (BWC). Berend et al., show that the BWC is NP-Complete [1]. In the rest of this work we deal with the optimization version of BWC (OBWC). In this case, we have the graph G with b vertices colored in black. The objective is to maximize the number of vertices that do

not have black vertices as neighbors.

Problem 1.2 (OBWC).

Input: A simple and undirected graph $G = (V, E)$ and positive integer b .

Output: Maximum number of vertices which do not have black vertices as neighbors.

Genetic Algorithms

The genetic algorithms is a set of general methods to solve optimization problems, proposed by Holland[5]. This method is based in the neodarwinist evolutionary theory, using concepts of natural selection and evolution for the best adaptation of the species. In the field of the optimization, genetic algorithms treat the solutions of certain optimization problem as individuals and the objective function as the environment.

Algorithm 1 Simple Genetic Algorithm

Require: $POP_{size}, P_{cross}, P_{mutation}$

Ensure: S_{best}

$POP = Initialization(POP_{size})$

$Evaluate(POP)$

$S_{best} = GetBestSolution(POP)$

While $\neg StopCondition$ **do**

$Parents = Selection(POP)$

$Children = Crossover(Parents, P_{cross})$

$Mutation(Children, P_{mutation})$

$Evaluate(Children)$

$S_{best} = GetBestSolution(Children)$

$POP = Replacement(POP, Children, P_{replacement})$

End while

Return S_{best}

In the Algorithm 1 we show the general form of the genetic algorithm. Note that it is necessary to chose many procedures in Algorithm 1 as the following:

Population size (POP_{size}).

Stop condition.

Initialization, selection and replacement method.

Crossover and mutation method.

Probability of crossing (P_{cross}), mutation ($P_{mutation}$) and proportion of replacement ($P_{replacement}$).

Many of these parameters are explicit in genetic

algorithm like POP_{size} or P_{cross} , but others do not. For example, the crossover method depends on individual representation and for a selected representation there are many crossing methods. To deal with this problem, some parameters were tuned with Automatic Tuning Tools like irace[7].

2. GENETIC ALGORITHM AND OBWC

For the application of genetic algorithm (GA) in OBWC we show a formulation as integer programming problem. Let $G = (V, E)$ a graph and a positive integer b . For all $v \in V$:

$$x_v = \begin{cases} 1 & \text{If vertex } v \text{ is black} \\ 0 & \text{Otherwise} \end{cases}$$

$$y_v = \begin{cases} 1 & \text{If vertex } v \text{ is white} \\ 0 & \text{Otherwise} \end{cases}$$

$$z = \text{Máx} \sum_{v \in V} y_v(1)$$

$$\sum_{v \in V} x_v = b \quad (2)$$

$$\sum_{s \in N[v]} y_s \leq (1 + \deg(v))(1 - x_v) \text{ for all } v \in V \quad (3)$$

The objective function is to maximize the number of white vertices on the graph. In constraint (2) we fix the value of b and the constraint (3) implies that if a selected vertex is colored with white (or black), its neighbors including it must have the same color or uncolored.

In our particular case we integrate whole model in the objective function of GA as follows:

$$f_{obj} = \sum_{v \in V} y_v - |V| \left(\sum_{v \in V} x_v - b \right)^2 \quad (4)$$

Is easy to see that the equation (4) does not include the anti-coloring behavior. To avoid this problem we implement a repair procedure in the representation of the solution. Since all variables in the model are binary we represent each solution as a binary vector x that allows all values of x_v for all $v \in V$. Due to y_v can be calculated from the values of x , genome does not include y but is calculated as auxiliary variable.

Algorithm 2 Objective function with repair procedure

Require: $G = (V, E), x$

Ensure: f_{obj}

$y = \neg x$

For each $v \in V$ **do**

If $y_v = 1$ **then**

$t = 0$

For $s \in N(v)$ **do**

$t = t \vee x_s$

End For

End If

$y_v = \neg t$

End For

Return $\sum_{v \in V} y_v - |V|(\sum_{v \in V} x_v - b)^2$

Now in our version of genetic algorithm for OBWC we deal with a lot of parameters which would be tuning. To avoid this problem we use irace tool [6]. This software implements F-Race procedures as a set of scripts to produce a set of parameters settings. The next list shows all parameters to be tuned its ranges and types, also including different operators for initialization, selection and crossover.

- Crossover probability: $P_{cross} \in [0, 1]$, float.
- Mutation probability: $P_{mutation} \in [0, 1]$, float.
- Replacement percentage:
 $P_{replacement} \in [0, 1]$, float.
- Population size:
 $POP_{size} \in [10, 150]$, integer.
- Number of generations:
 $N_{gen} \in [100, 50000]$, integer.
- Selection Operator:
 $sel_{op} \in \{rank, roulette, tournament, dss, srs, uniform\}$.

- Rank selection (rank): picks the best member of the population every time.

Roulette selection (roulette): this selection method picks an individual based on the magnitude of the fitness score relative to the rest of the population. The higher the score, the more likely an individual will be selected. Any individual has a probability p of being chosen where p is equal to the fitness of the individual

divided by the sum of the fitness of each individual in the population.

- Tournament selection (tournament): uses roulette selection to select two individuals then picks the one with the higher score.

- Deterministic sampling selector (dss) uses a two-staged selection procedure. In the first stage, each individual's expected representation is calculated. A temporary population is filled using the individuals with the highest expected numbers. Any remaining positions are filled by first sorting the original individuals according to the decimal part of their expected representation, then selecting those highest in the list. The second stage of selection is uniform random selection from the temporary population.

- Stochastic remainder sampling selector (srs) uses a two-staged selection procedure. In the first stage, each individual's expected representation is calculated. A temporary population is filled using the individuals with the highest expected numbers. Any fractional expected representations are used to give the individual more likelihood of filling a space. For example, an individual with a score of 1.4 will have 1 position then a 40% chance of a second position. The second stage of selection is uniform random selection from the temporary population.

- Uniform selection (uniform): picks randomly from the population. Any individual in the population has a probability p of being chosen where $p = \frac{1}{POP_{size}}$.

- Initialization Operator
 $init_{op} \in \{uniform, set, unset\}$

- uniform initialization: put 0 or 1 in each gene with probability 0.5.

- set initialization: put all genes in "1".

- unset initialization: put all genes in "0".

- Crossover operator
 $cross_{op} \in \{onepoint, twopoint, uniform, evenodd\}$

- One Point Crossover (onepoint): uses one random point between $[1, |V| - 1]$ in binary vector to create Childs.

- Two Point Crossover (twopoint): uses two different random points between $[1, |V| - 1]$ in binary vector to create Childs.

- Uniform crossover (uniform): pick with probability 0.5 which of two parents gives each gene.

- Even Odd crossover (evenodd): pick even genes from first parent and odd genes for second parent for first child. For second child, in opposite sense.

For mutation we only use simple bit flip. We also include another version of genetic algorithm with no fixed mutation rate (GA-NM). Thomas and Schütz proposed the next expression to control the mutation rate in each generation[1].

$$m(t) = \left(2 + \frac{n-2}{N_{gen}-1}t\right)^{-1}$$

Where n is the size of chromosome (solution representation), N_{gen} is the total number of generations and t is the current generation.

3. PARAMETER TUNED AND EXPERIMENTAL RESULTS

To perform parameter tuning, irace needs training instances. In our case we choose one of the hardest case of each instance proposed by Berend, et al., to test new heuristics for OBWC[3]. The training instances are:

- $C_{15} \boxtimes C_{11}$ and $b = 125$.
- $C_{21} \boxtimes C_{20}$ and $b = 100$.
- Planar graph with 200 vertices (PI_{200}) and $b = 150$.
- Planar graph with 240 vertices (PI_{240}) and $b = 150$.
- Binomial random graph with 500 vertices, edge probability $p=0.1$, ($rnd_{500,0.1}$) and $b = 150$.
- Binomial random graph with 500 vertices, edge probability $p=0.2$, ($rnd_{500,0.2}$) and $b = 150$.

After irace runs with 8000 executions with GA and GA-NM respectively it, give us the parameters of the TABLE 1 and TABLE 2

TABLE 1: BEST CONFIGURATIONS FOR GA AFTER IRACE EXECUTION

Configuration	P_{cross}	$P_{mutation}$	POP_{size}	N_{gen}	sel_{op}	$init_{op}$	$cross_{op}$	$P_{replacement}$
c_1	0.4507	0.0079	90	9419	dss	uniform	evenodd	0.4937
c_2	0.4626	0.0078	89	9502	dss	uniform	evenodd	0.4739
c_3	0.5014	0.0033	85	9181	dss	uniform	evenodd	0.4981
c_4	0.5055	0.0033	84	9241	dss	uniform	evenodd	0.4955
c_5	0.4119	0.0039	77	9249	dss	uniform	evenodd	0.4728

TABLE 2: BEST CONFIGURATIONS FOR GA-NM AFTER IRACE EXECUTION

Configuration	P_{cross}	POP_{size}	N_{gen}	sel_{op}	$init_{op}$	$cross_{op}$	$P_{replacement}$
c_1	0.6417	132	15674	uniform	uniform	twopoint	0.9325
c_2	0.9406	119	17611	roulette	uniform	twopoint	0.9770
c_3	0.6251	142	15810	uniform	uniform	twopoint	0.9113
c_4	0.8279	117	18970	roulette	uniform	twopoint	0.9836

All aspects of implementation were worked with GAlib which is a general purpose library to implement Genetic Algorithms.

In order to test GA and GA-NM performance with the automatic parameter tuning. We execute 100 times each instance with each configuration and we report the best result obtained (BEST) and plot quality of each algorithm as in the Figure 2 with the following formula:

$$Q_A = \frac{BEST_A}{BEST_{Known}}$$

Where $BEST_{Known}$ is the optimal or state of the art solution and $BEST_A$ The best solution for algorithm A.

To compare the performance of GA and GA-NM with the best known reported solution, we show $BEST_{TABU-R}$ and $BEST_{TABU-SG}$ as the solutions of Tabú search with random initial solution and Tabú search with semi greedy initial solution respectively which is the best performance heuristics for OBWC [2], as can be seen in TABLE 3. Also all results of TABLE 3 are summarized in the FIGURE 2.

TABLE 3: EXPERIMENTAL RESULTS

Alg/Instance	$C_{15} \times C_{11}$										$C_{21} \times C_{20}$	
	b=20	b=21	b=25	b=28	b=70	b=99	b=100	b=120	b=125	b=141	b=80	b=100
OPT	123	121	116	113	71	44	41	22	21	8	300	276
IP(Gurobi)	123	121	116	113	71	44	41	22	18	8	300	276
$BEST_{GA}$	123	121	116	113	71	44	41	22	18	8	298	276
$BEST_{GA-NM}$	123	121	116	113	71	44	41	22	18	8	300	276
$BEST_{TABU-R}$	123	121	116	113	71	44	41	21	18	8	300	276
$BEST_{TABU-SG}$	123	120	116	113	71	44	41	22	16	6	300	276

Alg/Instance	P_{200}^I							P_{240}^I						
	b=10	b=25	b=50	b=75	b=100	b=125	b=150	b=10	b=25	b=50	b=75	b=100	b=125	b=150
OPT	≥ 39	≥ 24	NA	NA	NA	NA	NA	≥ 63	≥ 48	≥ 25	NA	NA	NA	NA
IP(Gurobi)	187	170	146	119	93	70	45	186	170	143	119	91	69	45
$BEST_{GA}$	187	170	146	119	93	70	44	186	170	143	117	90	69	44
$BEST_{GA-NM}$	187	170	146	119	93	70	44	186	170	143	117	90	69	44
$BEST_{TABU-R}$	187	168	144	117	90	68	43	185	170	143	117	91	69	45
$BEST_{TABU-SG}$	187	169	143	115	93	65	45	186	169	143	119	89	69	41

Alg/Instance	$rnd_{200,0.1}$		$rnd_{500,0.1}$			$rnd_{500,0.2}$				$rnd_{600,0.05}$
	b=10	b=40	b=20	b=40	b=150	b=20	b=40	b=100	b=150	b=20
OPT	≤ 123	≤ 44	≤ 163	≤ 80	≤ 24	≤ 307	≤ 201	≤ 88	≤ 59	≤ 60
IP(Gurobi)	121	39	124	62	24	271	157	68	45	43
$BEST_{GA}$	121	39	147	62	16	289	176	62	40	50
$BEST_{GA-NM}$	121	39	150	68	16	290	177	68	40	54
$BEST_{TABU-R}$	121	39	150	71	17	290	180	58	43	53
$BEST_{TABU-SG}$	121	40	151	72	15	290	181	67	39	55

Gurobi is a state of the art solver for linear and integer programming. As you can see in many cases, Gurobi solutions are better than any heuristic, but in

the case of Gurobi, all instances were run in a specialized hardware during one hour each instance. In the case of genetic algorithm the biggest instance spends around 10 minutes in a single core computer.

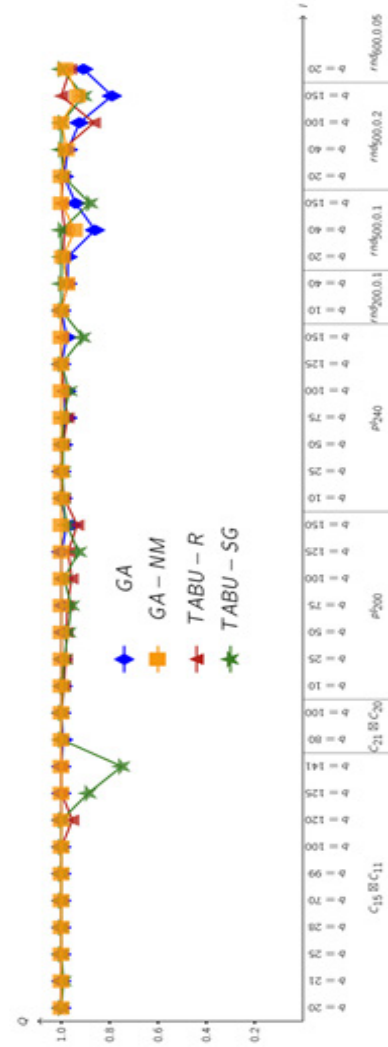


FIGURE 1: EXPERIMENTAL RESULTS PLOT

4. CONCLUSIONS

We have presented two versions of genetic algorithm through which we found the best known solutions in most of the instances. Since the paper were proposed, instances did not show execution time or number of objective function calls, we only report the quality of our solutions and the overall comparison in all possible instances of [2] and its better results. Genetic algorithms are an alternative for Tabú Search relatively easy to implement in this particular case, and with a better performance in most cases.

REFERENCES

- [1] Bäck, T., Schütz, M. Intelligent mutation rate control in canonical genetic algorithms. *Foundations of Intelligent Systems*. 1996, 158–167.
- [2] Berend, D., Korach, E., Zucker, S. A Reduction of the anticoloring problem to connected graphs. *Electronic Notes in Discrete Mathematics*. 2007, 28 445–451.
- [3] Berend, D., Korach, E., Zucker, S. Tabú search for the BWC problem. *Journal of Global Optimization*. 2012, 54 (4), 649–667.
- [4] Berge, C., Minieka, E. *Graphs and hypergraphs*. North-Holland publishing company Amsterdam.
- [5] Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press.
- [6] Lipton, R.J., Tarjan, R.E. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*. 1979, 36 (2), 177–189.
- [7] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Stützle, T., Birattari, M. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives*. 2016, 1 (3), 43–58.

ABOUT THE AUTHORS



Luis Eduardo Urbán Rivero es Ingeniero en computación por la Universidad Autónoma Metropolitana (UAM) Azcapotzalco, Maestro en optimización y Candidato a Doctor en Optimización por la misma Institución.



Rafael López Bracho es es Actuario por la Universidad Nacional Autónoma de México, tiene un Diplome d'études approfondies en investigación de operaciones por la Université Scientifique et Medicale de Grenoble, Francia y Doctor en Informática por la Université de Paris Sud, Francia en el año 1981. Actualmente se desempeña como profesor investigador en el Departamento de Sistemas de la UAM Azcapotzalco.



Javier Ramírez Rodríguez es Actuario por la Universidad Nacional Autónoma de México y Doctor en Ciencias Matemáticas por la Universidad Complutense de Madrid, España. Actualmente se desempeña como profesor investigador en el Departamento de Sistemas de la UAM Azcapotzalco.