Temporal Heterogeneity Improves Speed and Convergence in Genetic Algorithms

Yoshio Martinez, Katya Rodriguez, and Carlos Gershenson

Abstract—Genetic algorithms have been used in recent decades to solve a broad variety of search problems. These algorithms simulate natural selection to explore a parameter space in search of solutions for a broad variety of problems. In this paper, we explore the effects of introducing temporal heterogeneity in genetic algorithms. In particular, we set the crossover probability to be inversely proportional to the individual’s fitness, i.e., better solutions change slower than those with a lower fitness. As case studies, we apply heterogeneity to solve the N-Queens and Traveling Salesperson problems. We find that temporal heterogeneity consistently improves search without having prior knowledge of the parameter space.

I. INTRODUCTION

E NGINEERING is an area full of challenges which require efficient solutions. Often, due to the complexity of a problem, it is not feasible to find an optimal solution. Thus, it is desirable to find a good enough solution in the shortest possible time. Still, there is a plethora of different search algorithms that can explore a state space to find solutions. As the variety of state spaces to explore is infinite, an open problem lies in finding the parameters or methods that will allow an algorithm to find reliably and consistently a good solution for any given state space.

Genetic algorithms (GA) allow us to find solutions to various problems (combinatorial, regression, optimization) and are inspired by natural selection. These algorithms evolve a population in each generation and individuals are evaluated with a loss function to evaluate the progress of a population of solutions [11], [18]. GA have two main parameters. One is the crossover probability: how likely two solutions are to be combined (mate) to produce new solutions (offspring). The other is the mutation probability: how likely a solution will have random changes.

In this work, we explore the effect of adjusting crossover probabilities heterogeneously. That is, different individuals will have different probabilities (depending on their fitness). A similar exploration has been made related to mutation probabilities [4]. Two combinatorial optimization problems are used for testing the proposed genetic algorithm. The first one is the well-known N-queens problem which was proposed by Max Bezel in 1848 and belong to the NP-Complete problems [2], the number of solutions to evaluate with N-queens is N!, which shows that when N increases the problem becomes intractable when using greedy methods. This aims to place N queens on a N by N chess board, where no queen attacks any other, i.e., there is only one queen in each column, row, and diagonal. The second problem is the Traveling Salesman Problem (TSP), formally defined as follows [23], [22].

The TSP is a combinatorial problem with the objective of finding the path of the shortest length (or the minimum cost) on an undirected graph that represents cities or nodes to be visited. The traveling salesman begins at one node, visits all other nodes consecutively only once, and finally returns to the starting point. In other words, given n cities \{c_1, c_2, ..., c_n\}, and their permutations \{σ_1, σ_2, ..., σ_{n!}\}, the goal is to find σ_i such that the sum of all Euclidean distances between each node an its successor is minimized. The successor of the last node in the permutation is the first one, therefore with n nodes or cities, we’ll have \((n-1)!/n\) different paths. The Euclidean distance \(d\) between any two cities with coordinates \((x_1, y_1)\) and \((x_2, y_2)\) is calculated by:

\[
d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}
\] (1)

And the path length \(l\) to be minimized is calculated as follows:

\[
l = \left( \sum_{n=1}^{N-1} d(c_n, c_{n+1}) \right) + d(c_N, c_1) \] (2)

Thus, the aim of this paper is to evaluate the effect of temporal heterogeneity in genetic algorithms for solving these two combinatorial problems.

II. RELATED WORK

There are already several works related to genetic algorithms for solving the N-Queens problem. Bozikovic, et al. [5] solved the problem with the help of a parallel genetic algorithm. Farhan, et al. [9] found the 92 solutions for the 8-Queens. There is also related work to obtain the positions of the queens by chaining solutions for small values of \(N\) [1]. In the same way, there are several works that solve this problem with other methodologies, such as Hu, et al. [12] who used Particle Swarm Optimization (PSO) to treat this problem. The literature addressing the TSP is vast, e.g. [8], [10], [17], [14]. However, in this work, the aim is not to find all the possible solutions that a problem has (Table 1 lists the total number of solutions, possible combinations \(N!\)), and the probability that a random configuration is a solution for \(N <= 20\), but to accelerate the search process using a.

Authors are with Universidad Nacional Autónoma de México.
C. Gershenson is also with Lakeside Labs GmbH, Lakeside Park B04, 9020 Klagenfurt am Wörthersee, Austria. email: cgg@unam.mx
smaller amount of calculations both in the crossover and the evaluation stages of the GA.

### TABLE I

**Number of different solutions for the N-queens problem.**

| N  | solutions | combinations | solutions / combinations |
|----|-----------|--------------|-------------------------|
| 1  | 1         | 1            | 1                       |
| 2  | 0         | 2            | 0                       |
| 3  | 0         | 6            | 0                       |
| 4  | 2         | 24           | 0.0833                  |
| 5  | 40        | 720          | 5.5 \times 10^{-3}      |
| 6  | 724       | 362,880      | 1.9951 \times 10^{-4}   |
| 7  | 2,680     | \sim 399 \times 10^5 | 6.7139 \times 10^{-5} |
| 8  | 14,200    | \sim 4,790 \times 10^5 | 2.9644 \times 10^{-5} |
| 9  | 73,712    | \sim 62,270 \times 10^5 | 1.1837 \times 10^{-5} |
| 10 | 365,956   | \sim 871,782 \times 10^5 | 4.1936 \times 10^{-6} |
| 11 | 2,279,184 | \sim 1,307 \times 10^9   | 1.7429 \times 10^{-6} |
| 12 | 14,772,512| \sim 20,922 \times 10^9   | 7.0604 \times 10^{-7} |
| 13 | 95,815,104| \sim 355,687 \times 10^9 | 2.6938 \times 10^{-7} |
| 14 | 666,090,624| \sim 640,237 \times 10^{10} | 1.0403 \times 10^{-7} |
| 15 | 4,968,057,848| \sim 1,21645 \times 10^{26} | 4.084 \times 10^{-8} |
| 16 | 39,029,188,884| \sim 2.43290 \times 10^{27} | 1.6042 \times 10^{-8} |

Previously, a study entitled “Rank Diversity of Languages: Generic Behavior in Computational Linguistics” [6] observed that the most important elements in a system change more slowly than the less important ones. This research assigned a rank to a word depending on its frequency, where a rank “1” is the lowest, meaning that the word was the most frequently used in a given year, and words with a very low frequency were assigned a high rank. The aforementioned behavior was found: some words that were repeated a lot followed that trend for many years and changed their rank very slowly or not at all, while seldom used words changed their rank considerably every year. In a similar way, in another investigation entitled “Genetic Temporal Features of Performance Rankings in Sport and Games” [19], the same behavior was observed: successful athletes or teams who remained at a lower rank remained for longer time, while less successful athletes or teams (with higher ranks) changed their rank very quickly. The hypothesis is that such a temporal heterogeneity provides complex systems with a balance between stability and variability [16], [13], also known as criticality [5], [21]. Still, homogeneous models restrict this balance to a phase transition, making it difficult to find parameters and solutions without prior knowledge of a problem, or requiring other mechanisms to guide the evolution of parameters towards criticality [2], [24].

This encourages us to explore the effect of heterogeneity on genetic algorithms, given the evidence that criticality favors evolvability [20]. In the methods section we detailed how this approach is included in the genetic algorithms.

### III. METHODS

Our idea is to cross less fit individuals in the population more quickly than fitter individuals. Thus, those with a high fitness value will have a lower probability of mixing with other individuals. As it will be shown, this results a decrease in the number of calculations carried out, allowing the search process to be accelerated.

**A. Genetic Algorithms**

The proposed GA has a structure very similar to that used by previous work [3]. Once an initial population has been generated, a processes for each individual in the population is created. This allows an individual to be independently evaluated. The same applies for the crossover. This is not the case for mutation, since this genetic operator does not require as many instructions as the other two processes mentioned above.

**B. N-queens**

The goal is to place N queens on a N by N chess board. We will have a population V where each v_i represents an individual in the population. To place a queen on the board with the configuration of an individual, the columns of the board are taken as the position i in the array and the rows by the number contained in such position. Below an individual solution to the problem of the 8 queens is given and its placement on the board is shown in Figure 1(a):

\[ v_i = \{2, 4, 7, 3, 0, 6, 1, 5\} \]

The initial population of individuals consists of randomly generated vectors without repeated numbers per individual. Here the objective is to minimize the number of intersections between queens across the board (Algorithm 1): zero implies that there are no intersections and thus the problem is solved.

To calculate the probability of crossing over a couple of individuals v_a and v_b, the following steps are performed:

1) The entire population is evaluated and fitness is obtained for each individual \( \Phi(v_i) \).
2) The fitness of the worst individual is estimated in the population \( \tau \), and finally.
(a) 8 Queens  (b) 10 Queens  (c) 20 Queens

Fig. 1. Solutions to 8, 10 and 20 queens.

**Algorithm 1:** Objective function of $N$-queens.

Result: Return $x$

initialization $x$;

for $g = 1, 2,..., v$
do

for $i = 1, 2,..., g$
do

if $v_i = v_g$ or $\text{abs}(v_g - v_i) = \text{abs}(i - g)$

$x = x + 1$;

end

end

3) To decide if it is possible to crossover $\Psi(v_a, v_b)$, the following operation is evaluated:

$$p(v_a) = 1 - \{\Phi(v_a)/\tau\}.$$ (3)

This operation is performed for $v_a$ and $v_b$, thus obtaining $w_a$ and $w_b$ (Algorithm 2). This simple operation allows us to slowly crossover the best suited individuals and the worst are constantly crossing over.

**Algorithm 2:** Crossover for $N$-queens and TSP.

Result: Return $V$

if $(1 - \{\Phi(v_a)/\tau\}) <= \text{random}$ then

$w_a = \Psi(v_a, v_b)$

end

if $(1 - \{\Phi(v_b)/\tau\}) <= \text{random}$ then

$w_b = \Psi(v_b, v_a)$

end

C. Traveling Salesperson Problem

The proposed genetic algorithm will be applied to solve a case of the problem of the traveling salesperson, in the same way as for the problem of the $N$-queens. Here, vectors with integers values are generated, where each position of the vector corresponds to a point of Figure 2.

Then, all points have to be visited only once (without repetitions) to find the combination of nodes that give the shortest route. In the same way as with the $N$-queens problem, comparisons between each of the runs considering the distance obtained, considering the execution time and the number of instructions performed.

Fig. 2. Image with 7,000 points based on the logo of our university.

IV. EXPERIMENTS

A. $N$-queens

The problem of the $N$-queens is then solved for $N$ values of 20, 40, 60, 80, 100, 150, 200, 250, and 500. Ten or fifty runs were performed for each of the board configurations mentioned above, where the number of solutions per run, runtime, and instructions performed are compared. The configuration used for each board is shown in Table II.

| $N$          | Population | Crossover | Mutation | Generations | runs |
|--------------|------------|-----------|----------|-------------|------|
| 20, 40, 60, 80, 100, 150 | 300        | 0.9       | 0.1      | 500         | 10   |
| 200, 250     | 300        | 0.9       | 0.1      | 1000        | 10   |
| 500          | 500        | 0.9       | 0.1      | 3000        | 50   |

TABLE II
CONFIGURATIONS USED FOR $N$-QUEENS PROBLEM.
B. Traveling Salesperson Problem

As discussed earlier, the classical genetic algorithm will be tested and compared against a genetic algorithm using temporal heterogeneity. For the case of the TSP, only one run of the genetic algorithm is executed using the parameters shown in Table III.

| Points | Population | Crossover | Mutation | Generations |
|--------|------------|-----------|----------|-------------|
| 7000   | 300        | 0.9       | 0.1      | 80000       |

V. RESULTS

In this section, our results obtained are shown, starting with the N-queens problem, followed by the TSP. A personal computer with an Intel core i7 and 16 GB of RAM was used.

A. N-queens

Tables IV and V show the results for the heterogeneous and the classical (homogeneous) genetic algorithms, respectively. The first column indicates the number of queens to be placed on the board, the “Resolved” column indicates the number of times (of the 10 or 50 runs of the algorithm) that the algorithm reaches a valid dashboard configuration. On average, temporal heterogeneity leads to a higher percentage of runs that find a solution.

The number of average instructions shows those required for crossover, which is the part that we modified. These are drastically reduced with temporal heterogeneity.

Finally, the average time in minutes for runs with N is detailed. Consistently, temporal heterogeneity reduces total computation time.

| N     | Resolved | Crossover instructions | Time |
|-------|----------|------------------------|------|
| 20    | 10       | 203,106.7              | 0.54 M |
| 40    | 10       | 1,179,512.4            | 2.03 M |
| 50    | 10       | 2,319,122.2            | 2.45 M |
| 60    | 10       | 3,199,086.1            | 3.22 M |
| 80    | 9        | 8,698,568.0            | 6.34 M |
| 100   | 9        | 13,255,157.2           | 8.3 M  |
| 150   | 6        | 29,060,304.6           | 14.19 M|
| 200   | 6        | 55,644,604             | 18.33 M|
| 250   | 7        | 83,096,536.7           | 24.54 M|
| 500   | 26       | 380,562,758.3          | 58.31 M|

Having the fittest individuals without crossing over across generations and promoting crossover of the unfit seems to allow local optimal jumping and moving to better regions in the search space. Also, not crossing over the entire population allows to decrease the execution time of the algorithm and the instructions that are carried out, as the evaluation of the objective function is performed less times (Figure 4).

Figure 3 allows to visualize the effect of temporal heterogeneity on computation time as queens are increased. The graph seems to indicate that the advantage of temporal heterogeneity is maintained as the complexity of the problem increases. This suggests that the effect of temporal heterogeneity could still have good results if the number of queens is increased further.
These results encourage us to try more queens, thus solving the problem for 1000 queens. Running once for each algorithm for 6000 generations and a population of 300, the following results were obtained (Table VI):

| Heterogeneity | Resolved | Crossover instructions | Time (minutes) |
|----------------|----------|------------------------|-----------------|
| Yes            | 1        | 1,496,961,310          | 191.31          |
| No             | 0        | 2,170,172,477          | 256.34          |

It was interesting to notice that, the genetic algorithm that uses temporal heterogeneity, solves the problem of 1000 queens in just over 3 hours. On the other hand, the classical algorithm did not manage to solve the problem in 6000 generations.

In Figure 5(a) an individual of the population of 1000 queens in generation 0 is shown, where it is obviously not a correct solution. The queens that are well located are represented with a small asterisk; on the other hand, the queens that are under attack are represented with a blue circle, and green lines are drawn from it in all directions except vertical. In Figure 5(b) the solution to 1000 queens using the heterogeneous algorithm is presented. On the other hand, in Figure 5(c) the best individual that the classic algorithm generated is given, where it is clearly seen that it is not a solution to the problem due to having intersections (attacks).

**B. Traveling Salesperson Problem**

The results obtained using the genetic algorithm with and without heterogeneity applied to the TSP with 7,000 nodes are shown as follows.

| Heterogeneity | Distance  | Crossover instructions | Time (hours) |
|----------------|-----------|------------------------|--------------|
| Yes            | 24,032.29 | 211,337.7              | 87.24        |
| No             | 35,395.33 | 192,617.8              | 94.89        |

Table VII shows the distance, number of instructions and time. Figure 6 is the result obtained by running the genetic algorithm using temporal heterogeneity for 80,000 generations, that although the algorithm made more instructions in the crossover operation, it could obtain a considerably better result in slightly less time, suggesting that temporal heterogeneity could be valuable to solve search problems more complex than those studied here. Still, this goes beyond the scope of this paper will be the focus of future work.
VI. CONCLUSION

As it was shown in the problems studied here, temporal heterogeneity in genetic algorithms leads to better and faster results due to the fact that it performs less operations with the individuals, which allows to accelerate the search process. The rationale behind our method is similar to the simulated annealing algorithm [15]. Both algorithms advance with smaller steps when they are closer to the solution. Still, in simulated annealing, it is not trivial to find the optimal cooling constant, which varies with the particular problem. On the contrary, with the proposed method, we do not need any additional parameter to achieve good results in different instances of the combinatorial problems studied here.

Nevertheless, the benefits of temporal heterogeneity in genetic algorithms do not mean that it will be universally beneficial for all possible problems [25]. Thus, it is necessary to test temporal heterogeneity in different problems to truly measure its usefulness. In this paper we showed that temporal heterogeneity helps problems with integer coding, such as N-queens and TSP. As a future work, we will evaluate the potential benefits of temporal heterogeneity in real coding problems and stochastic environments. We have already trained a deep neural network to learn how to play Atari games by reinforcement with the help of temporal heterogeneity, obtaining encouraging preliminary results.

It would also be interesting to test whether temporal heterogeneity in mutations would provide additional benefits to that found in crossover. Also, the precise heterogeneity we used was arbitrary. It would be useful to explore systematically different degrees of heterogeneity, in the search of an “optimal” heterogeneity for a particular problem.

REFERENCES

[1] Bruce Abramson and Moti Yung. Divide and conquer under global constraints: A solution to the n-queens problem. Journal of Parallel and Distributed Computing, 6(3):649 – 662, 1989.
[2] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality: An explanation of the 1/f noise. Phys. Rev. Lett., 59(4):381–384, July 1987.
[3] M. Bozikovic, M. Golub, and Leo Budin. Solving n-queen problem using global parallel genetic algorithm. In The IEEE Region 8 EUROCON 2003, Computer as a Tool, pages 104 – 107 vol.2, 10 2003.
[4] Jorge Cervantes-Ojeda and Christopher Stephens. A rank-proportional generic genetic algorithm. In Theodore Simos, editor, Recent Progress in Computational Sciences and Engineering, pages 71–74. Taylor & Francis, 05 2006.
[5] Kim Christensen and Nicholas R Moloney. Complexity and criticality. World Scientific Publishing Company, 2005.
[6] Germinal Cocho, Jorge Flores, Carlos Gershenson, Carlos Pineda, and Sergio Sánchez. Rank diversity of languages: Generic behavior in computational linguistics. PLOS ONE, 10(4):1–12, 04 2015.
[7] Kelly Crawford. Solving the n-queens problem using genetic algorithms. 2:1039–1047, 01 2010.
[8] Marco Dorigo and Luca Maria Gambardella. Ant colonies for the travelling salesman problem. Biosystems, 43(2):73 – 81, 1997.
[9] Ahmed S. Farhan, Wadhah Z. Tareq, and Fouad H. Awad. Solving n queen problem using genetic algorithm. International Journal of Computer Applications, 122(12):11–14, July 2015.
[10] Chunhua Fu, Lijun Zhang, Xiaojing Wang, and Liying Qiao. Solving tsp problem with improved genetic algorithm. IAP Conference Proceedings, 1967(1):040057, 2018.
[11] David E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.
[12] Xiaohui Hu, Russell Eberhart, Yuhui Shi, and Yuhui Com. Swarm intelligence for permutation optimization: Case study of n-queens problem. Proceedings of the IEEE Swarm Intelligence Symposium, 2003, 07 2003.
[13] S. A. Kauffman. The Origins of Order. Oxford University Press, Oxford, UK, 1993.
[14] Fozia Khan, Nasiruddin Khan, Dr. Syed Inayatullah, and Shaikh Nizami. Solving tsp problem by using genetic algorithm. International Journal of Basic & Applied Sciences, 9, 08 2010.
[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.
[16] Christopher G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. Physica D, 42:12–37, 1990.
[17] Pedro Larrañaga, Cindy Kuijpers, R. Murga, I. Inza, and S. Díaz-Cerrada. Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review, 13:129–170, 01 1999.
[18] Melanie Mitchell. An Introduction to Genetic Algorithms. MIT Press, 1996.
[19] José A. Morales, Sergio Sánchez, Jorge Flores, Carlos Pineda, Carlos Gershenson, Germinal Cocho, Jerónimo Zizumbo, Rosalío F. Rodríguez, and Gerardo Ilúñez. Genetic algorithm for solving the 8-queens problem with improved genetic algorithm. AIP Conference Proceedings, 1967(1):040057, 2018.
[20] Norman H Packard. Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, Dynamic Patterns in Complex Systems, pages 293–301. World Scientific, Singapore, 1988.
[21] Andrea Roli, Marco Villani, Alessandro Filisetti, and Roberto Serra. Dynamical criticality: Overview and open questions. Journal of Systems Science and Complexity, 31(3):647–663, 2018.
[22] Sergio Sánchez, Germinal Cocho, Jorge Flores, Carlos Gershenson, Gerardo Ilúñez, and Carlos Pineda. Trajectory stability in the traveling salesman problem. Complexity, 2018:2826082, 2018.
[23] Susmita Sur Kolay, Satyajit Banerjee, and C. Murthy. Flavours of traveling salesman problem in vlsi design. In ICAI, pages 656–667, 01 2003.
[24] Christian Torres-Sosa, Sui Huang, and Maximino Aldana. Criticality is an emergent property of genetic networks that exhibit evolvability. PLoS Comput Biol, 8(9):e1002669, 09 2012.
[25] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1):67–82, April 1997.
Yoshio Ismael Martínez Arévalo Yoshio is a master’s candidate at Universidad Nacional Autónoma de México (UNAM). He has a bachelor’s degree in Computer Systems and an interest in Complex Systems, Deep Learning, and Evolutionary Computing. He has extensive experience as software developer in high and low level languages, he is also a fan of applied robotics, blockchain applications, and software acceleration by GPU’s.

Katya Rodríguez Vázquez K. Rodríguez-Vázquez received the BEng degree in Computing Engineering from the National Autonomos University of Mexico (UNAM), in 1994 and the PhD from the University of Sheffield, Sheffield, UK, in 1999. She is currently a full time Research at the Institute of Applied Mathematics and Systems (UNAM). She has published a number of papers in international journals and conferences. Her research interests are in evolutionary computation, multiobjective optimization and its applications. Dr. Rodríguez-Vázquez has been a Member of the Technical Program Committee for conferences related to evolutionary computation and reviewer of related journals.

Carlos Gershenson García Carlos Gershenson is a tenured, full time research professor at the computer science department of the Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas at the Universidad Nacional Autónoma de México (UNAM), where he leads the Self-organizing Systems Lab. He is also an affiliated researcher at the Center for Complexity Sciences at UNAM. He is a honorary member at Lakeside Labs, Austria. He was a Visiting Professor at MIT and at Northeastern University (2015-2016) and at ITMO University (2015-2019). He was a postdoctoral fellow at the New England Complex Systems Institute (2007-2008). He holds a PhD summa cum laude from the Vrije Universiteit Brussel, Belgium (2002-2007). His thesis was on “Design and Control of Self-organizing Systems”. He holds an MSc degree in Evolutionary and Adaptive Systems, from the University of Sussex (2001-2002), and a BEng degree in Computer Engineering from the Fundación Arturo Rosenblueth, México. (1996-2001).

He is Editor-in-Chief of Complexity Digest, Associate Editor for the journals Complexity and Frontiers in Robotics and AI, and member of the Board of Advisors for Scientific American. He has received numerous awards, including a Google Research Award in Latin America and the Audi Urban Future Award.