Multi objective GA for optimizing stock usage

ABSTRACT

This paper presents a method in which was used a Genetic Algorithm (GA) in order to optimize the search for viable solutions regarding a specific supply chain problem, stock usage. This paper starts by giving GA a quick review and presents some fields were the use of GA excel other solutions. The supply chain optimization problem is then presented, as well as some common solutions. After the short presentation of the problem, one specific study is presented, as well as their solution, that compares GA and Ant Colony Optimization (ACO) for a similar problem. Finally our own solution is presented, as well as the results achieved and possible future works.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**.

KEYWORDS

Genetic algorithms, Fitness evaluation, Multi-objective optimization, Artificial intelligence

ACM Reference Format:

. 2020. Multi objective GA for optimizing stock usage. In *Proceedings of the Genetic and Evolutionary Computation Conference 2020 (GECCO '20).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Genetic Algorithm (GA) is a powerful meta-heuristics for optimizing solutions, specially in cases were the search space can be extremely large, leading to the impossibility of solving it by hand, or even with rule based algorithms [1]. There are numerous problems on supply chain that can benefit from this kind of meta-heuristics for instance, scheduling optimization, planning, handling and controlling the storage of the goods [2]. The sheer amount of possible viable solutions for these problems make local search heuristics, analytical methods or even exhaustive search extremelly time consuming, and with non optimial solutions [3]. Due to this the usage of meta-heuristics, such as GA [1], Simulated Annealing [4] and ACO [5] are widely used and have been presenting very good results. Those types of meta-heuristics are extremely suitable to guide the search for global optimal solutions, and as such are being used extensively [2].

Due to easiness of implementing even a basic GA [1], this specific meta-heuristic is widely applied in the optimization field, usually being considered the first choice [6]. Even though there may be solutions of higher quality achieved by other methods, GA have proven to present novel and more diverse solutions [6]. Those novel solutions, in most cases, can achieve very high performance, usually with some small changes on the base algorithm [7].

One specific optimization problem on the supply chain, is how to attend the amount of Open Order Requests (OOR) that keep arriving, while having a finite number of resources. In the simpler case where we either attend a complete OOR or we simple don't attend, a simple GA or ACO can find very good optimal solutions, in a few interations [2]. However, when we add the complexity of attending part of the OOR instead of attending it completely, the simple GA fails to achieve optimal solution, and tend to stay on a local solution. This specific problem is usually solved via rule based programs, rules such as the time when the OOR was inserted on the system, the client priority (some clients can *"skip the queue"*), and product priority (some products have to be shipped sonner than others). When trying to optimize this kind of problems, we usually end up with a multi-objective GA, where the *fitness* function need to attend more than one goal, while still trying to find the global optimal solution.

This paper tries to improve a GA in order to reach the optimal solution. We started with the most basic GA [1, 6] and, inspired by the representation found in [2], we developed a GA that is capable of reaching the global optimal solution, while still mantaining the rule based approach that most companies tend to use.

On section 2 we present a basic review of the GA, as well as some utilization and results found via GA. The supply chain problem we want to attack is presented on section 3, as well as it's complexity that lead to the usage of a GA. On section 4 we present the algorithm used to find the solutions and on section 5 the results we found with this algorithm. Finally on section 6 we look into possible future work.

2 GA

Genetic Algorithms (GA) as presented by Holland [1, 8, 9] are a meta-heuristic based on the Darwinian principal of survival of the fittest. GA works by creating a random initial population P with i different individuals, evaluating them against the problem the population should answer via a fitness $f(i)$ function, selecting the best individuals of the population and creating new individuals over time. In the most basic form, a population is a simple binary representation of strings, where the individual i is a valid solution for the problem at hand. The selection of the best individuals (the most fitted i) is usually a simple Roulette based selection, where the most fitted have a higher probability of being selected. A simple *crossover* is responsible for creating a new population P, and *chromosome* of i has a very small probability of being mutated.

The fitness function $f(i)$ is a function that evaluates each individual i of the population P and tells how fitted (or adapted) the individual i is, regarding the problem to be solved. This function is usually what takes most of the problem related information. The creation of the new population is based on the selection, crossover

GECCO '20, July 8–12, 2020, Cancun, Mexico
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

$$i_1 = \mathbf{00110|01111}$$
$$i_2 = 10110|01001$$
$$i_{result} = \mathbf{00110}01001$$

Figure 1: Single point crossover

and mutation operators, a simple form of a GA can be seen on the Algorithm 1. The *SelectParent* function dictates the selection pressure the algorithm is executing on the fitted individuals. In its simplest form, the selection is based on a normalization of the fitness (as shown on function 1), however this tends to rapidly evolve to a local optimal solution, and as such this is not desired, one can use a linear scaling or an exponential mapping [10], even though, for small problems using a simple normalization of the fitness can lead to very good solutions, albeit with some extra generations [6].

$$p_i = \frac{f(i)}{\sum f} \tag{1}$$

Algorithm 1 Basic creation of new population

```
function NewPopulation($P$)
    $NewPop \leftarrow 0$
    for $i \in 1, .., |P|$ do
        $si_1 \leftarrow SelectParent(P)$
        $si_2 \leftarrow SelectParent(P)$
        $NewPop_i \leftarrow Crossover(p_1, p_2)$
        $NewPop_i \leftarrow Mutation(NewPop_i)$
```

The Crossover function is responsible for creating new individuals by mixing the phenotype of 2 or more parents. The simplest solution is called the single point crossover, where a random point is selected and the new individuals gets part of the first parent, and part of the second parent. This method can be seen on 1, and is **very** dependent on the selected representation.

The simplest form of Mutation selects random chromosomes of the phenotype and apply a very small probability p_m (usually as low as 0.05%), for the binary representation, if the chromosome is selected for mutation it is swapped. Just like the Crossover, this is **very** dependent on the selected representation.

Albeit very simple, this form of GA has achieved considerable results, as seen on [6], [7] and [2]. With some changes on it, GA also achieved high quality results, as can be seen on [10], [11], [12], [13], [14] and many others.

3 LOGISTIC SCHEDULING PROBLEM

There are numerous opportunities of utilizing GA meta-heuristics on the supply chain processes. Those are real world problems that range from the classic traveler salesman problem (TSP), scheduling problems and even packing problems [2], [6], [14]. What makes this particular field so attractive to those algorithms is the sheer amount of different possibilities. Take for instance the problem of defining what available Open Order Request (OOR, O) to attend, you have at least those parameters to take account of:

(1) Is there enough stock available?
(2) Can we partially attend the OOR?
(3) How old is the OOR, will we be able to ship in time?
(4) Can the vehicle mix different clients and different OORs?
(5) Can one type of product be mixed with other type of products?

In [2] it was presented a performance comparison between GA and ACO for that specific problem where:

(1) There wasn't enough stock available
(2) It was not possible to partially attend the OOR
(3) This was the complete $f(i)$ function
(4) This was not take in account
(5) This was not take in account

Even if you try to keep things as simple as possible, and only try to find what you can possible attend, giving you have a finite stock and you either attend or don't attend the orders, you still have $|O|!$ possibilities that you can look into. You can reduce this a little by using how late the order is (called *tardiness*, or T) but even so, you may end up having one order o utilizing all the available stock, and as such leaving other orders to be late. This usually is not a feasible solution from the point of view of the business itself, the best would be to spread the stock and attend as much as possible, in order to maximize the amount of orders O attended.

To solve this problem, [2] presented an GA that utilizes the fitness function 2, it is a minimization function that separates the orders in two groups, Deliverable (D) and Non Deliverable (ND) orders, the tardiness of all the orders (D and ND) is added to the fitness function, that way it is possible to prioritize older orders to be deliverable, while still allowing the GA to look for other solutions. The constant ϵ is just a very small number to prevent 0 division. We recommend looking into [2] for the complete work, it's a very fast GA that presents very high quality results.

$$f_i = \frac{\sum_{j \in O} T_j + |O^0_{ND}|}{|O^0_D| + \epsilon} \tag{2}$$

What makes this fitness function feasible, is the representation [2] used to define each individual of the population. Each chromosome of each individual of the population represents an order that was inputted on the system and is waiting to be consumed. Therefore it is possible to use a simpler GA (as presented on section 2) in a straightforward way. The result information is later used to continue the logistic process (ie: buying parts, storing, shipping, etc).

4 ALGORITHM USED

The supply chain problem we had at hand was very similar to the one presented in [2] however, it had three main differences:

(1) We were able to attend partial requests
(2) Each order o was related only to one client c, but each client should have a priority ϕ_c of being considered first
(3) Each product η also had a priority ϕ_η to be selected and shipped ahead of other products

When adding those extra parameters (attending partial orders, the ϕ_c of each client and the ϕ_η of each product as dictated by the business definitions) we end up with what is called a *multi-objective GA*, where the GA has more than one optimization target:

(1) Select the amount of products η to use on each order o

Table 1: Stock information

Product	Available
A	20
B	5
C	15
D	30

Table 2: Open Orders

Client	Order #	Product	Amount
c_0	O_0	η_A	10
		η_C	30
c_1	O_1	η_B	5
	O_2	η_C	10
c_2	O_3	η_A	2
		η_D	20
	O_4	η_A	18
		η_C	15

(2) Use the ϕ_c and ϕ_η as a guide on how to attend each order o
(3) Process each order o in time (minimize the T_o of each order)

If all those targets are achieved, we have a result that:

- Stock is optimally used (no order o is left unattended), leaving spare physical space to store more products
- Clients will receive their products in a timelly manner
- Products that need to be shipped first, will be shipped first
- Lateness fee are avoided

4.1 Representation

To completely represent the problem at hand, each individual has to have the complete information of what to ship in all orders. Take for instance the stock represented by table 1, in it we have 4 products (A, B, C and D), so each order **has** to have all four products, even if it's not sending.

Take for instance the orders presented on the table 2, we have 3 clients with a total of 5 orders, each of the orders have multiple items.

In order to ship everything, the representation used for the individual is the matrix 3, where each row i is one client, and each column η is one product from the stock. The value of each $O_{i\eta}$ is the amount of the product η used for the client i. The result is the array represented on figure 2.

$$O_{i\eta} = \begin{bmatrix} 10 & 0 & 30 & 0 \\ 0 & 5 & 10 & 0 \\ 20 & 0 & 15 & 20 \end{bmatrix} \tag{3}$$

Using a representation like this (2D matrix mapped to a 1D array) we can achieve some degree of simplicity, and as such the population I is a matrix of n individuals where each chromosome

$$O_i = 10, 0, 30, 0, 0, 5, 10, 0, 20, 0, 15, 20$$

Figure 2: Individual representation to attend the complete orders from table 2

is a random value in the interval $[0, max(S_\eta \vee O_\eta)]$. Formally, the representation of each individual can be stated as on equation 4, where $R(a, b)$ is a random uniform distribution between a and b (inclusive), S_η is the amount of stock available for product η and O_η is the amount of the product η as requested by the order O of the client, c is the clients that have orders requested. Please take note that we are adding all products η for each order o of each client c, which means that if, for instance, client 1 has requested 10 units of product A on order 1, and another 7 units of product A in order 2, we will consider that he wants 17 units of product A. For now this is by design and a field of future improvement.

$$c_{ij} = R(0, min(S_{\eta(j)}, \sum O_{\eta(j)}))\forall j \in \eta, \forall i \in c \tag{4}$$

By creating an individual in this way, we can guarantee that the individual i will have a random amount of products that were requested in the order, and that amount will never be greater than what's available in stock. The population then is a set of n individuals created via equation 4. For example, using the same information from tables 1 and 2, we could end up with a population I of $n = 3$ as in matrix 5

$$I_i = \begin{bmatrix} 10 & 0 & 5 & 0 & 0 & 2 & 7 & 0 & 15 & 0 & 10 & 17 \\ 5 & 0 & 2 & 0 & 0 & 5 & 10 & 0 & 5 & 0 & 5 & 20 \\ 7 & 0 & 15 & 0 & 0 & 0 & 7 & 0 & 7 & 0 & 10 & 18 \end{bmatrix} \tag{5}$$

However, we still need two extra pieces of information. The first is how to optimize the packing. Usually when creating a container, products are packed in pallets, and each client c can have a different configuration of pallets per product η. Using the same type of representation, we can have the matrix 6, where for each client i, each product η has a packing amount.

$$\Delta_{i\eta} = \begin{bmatrix} 2 & 5 & 3 & 6 \\ 2 & 5 & 3 & 6 \\ 4 & 1 & 3 & 6 \end{bmatrix} \tag{6}$$

When using the matrix 6 on the problem, we need to make each item of the matrix 3 a multiple of the matrix 6, there resulting in equation 7. We use the floor function, because usually we are not allowed to ship partial parts of the product, therefore the remainder of the $\frac{O_{ij}}{\Delta_{ij}}$ is kept as a leftover.

$$O_{ij} = \lfloor \frac{O_{ij}}{\Delta_{ij}} \rfloor \forall i \in c, \forall j \in \eta \tag{7}$$

The other information we need is the priority of the client, and the priority of the products. that we need to store in the representation, that is the priority of the client and the priority of the products. Each order $o \in O$ has a priority ϕ that is the product of the priority of each product and the priority of the client ϕ_c. As can be seen on equation 8.

$$\phi_o = \phi_c \times \sum \phi_\eta, \forall \eta \in c \tag{8}$$

4.2 Fitness

With the new representation for the problem (equation 4 and 8) we can extend the fitness presented on [2] to the one presented on equation 9.

$$f(i) = \frac{\sum_{i=0}^{o} T_i + (\sum_{i=0}^{o} (\eta - o_{ij}) * \sum_{i=0}^{o} \phi_i)}{\sum_{i=0}^{o} (o_{ij}) * \sum_{i=0}^{o} \phi_i + \epsilon} \tag{9}$$

For reading simplification, the fitness function can be defined as equation 10, where $T = \sum_{i=0}^{o} T_i$ is the tardiness of the orders, $ND = \sum_{i=0}^{o} (\eta - o_{ij})$ is the non deliverable amount of products η and $D = \sum_{i=0}^{o} o_{ij}$ is the deliverable amount of products η.

$$f(i) = \frac{T + \sum ND * \phi_o}{\sum D * \phi_o + \epsilon} \tag{10}$$

Basically, instead of creating two separate groups (Deliverable and Non Deliverable), we are working with only one group, and this group is being tested against the total amount of products for each orders. Each order is then assigned a priority ϕ that works as a weight across all the orders. If there's enough stock available to send, we will eventually get an individual where $S = D$ and thus $f = 1$. If however we don't get this, we promote higher individuals where S is more attuned to ϕ, thus guaranteeing the priority selection. The tardiness T works as a shift to balance the oldness of all orders and apply more selective pressure on them.

4.3 Selection

For the selection we used an exponential mapping presented on equation 11. With this equation we can adjust the selection pressure by using different values of β.

$$\theta_i = \exp^{-\beta \times \frac{f_i}{f_{max}}} \tag{11}$$

4.4 Crossover

Since we have improved the individual representation to add the amount of the product η selected to send, we are now unable to use single point crossover, otherwise we would end up with individuals that want to send products that may have not been requested on the order o. Due to this, the crossover is a permutation of each order o of the individual i. For each order o a random uniform value is choosen, if less than ϕ_{xover} that order is swapped with the same order of the other individual, otherwise it is kept the same. This can be seen on Algorithm 2.

Algorithm 2 Crossover of two parents

```
function CROSSOVER($i_1$,$i_2$, $\phi_{xover}$)
    $Individual \leftarrow 0$
    for $j \in 1, .., |i|$ do
        $R \leftarrow RandomUniform$
        if $R \leq \phi_{xover}$ then
            $Individual_j \leftarrow i_{1j}$
        else
            $Individual_j \leftarrow i_{2j}$
```

If $\phi_{xover} = 1$ the ending result will **always** be a complete mix between parents. Using these parameters quickly leads to a local optimal solution, but it also increases the complexity of searching new feasible and viable solutions. However if $\phi_{xover} = 0.5$ then each order o has 50% chance of being swapped, increasing the time it takes to find good solutions, but testing more feasible solutions in the meantime.

Table 3: Parameters used

Parameter	Value
$\|G\|$	500
$\|I\|$	4196
E	20%
ϕ_{xover}	50%
ϕ_m	0.5%
ρ	10%

4.5 Mutation

Each chromosome from our representation is an amount of product η to be shipped, Therefore we need to mutate this chromosome, if he is selected for mutation. There are many ways to mutate real numbers, for instance via transforming the real number in a binary reprensetation, and mutate some bits [8]. For sake of simplicity, if the product η is selected to mutate, we sample a new random uniform value from $[\eta - \rho, \eta + \rho]$, where ρ is a fixed % of the η value. One can then reach the equantion 12 for the mutation of each η product, where R is a random uniform number.

$$\eta = \begin{cases} \lfloor \frac{min(R(\eta - (\eta * \rho), \eta + (\eta * \rho), S_\eta)}{\Delta_{i\eta}} \rfloor & if R \leq \phi_m \\ \eta & otherwise \end{cases} \tag{12}$$

It's important to note that we are adding a new parameters for the mutation, the % to mutate (ρ). This leaves us with some extra control over the mutation, as we now have two different controls:

(1) The probability of each chromosome being mutated
(2) The amount of mutation we allow the mutated chromosome to have

This resulted in a remarkable way to escape local optimal solutions.

5 RESULTS

Using the algorithm presented on 4, we were able to reach very high performance solutions. To achieve those, we used the parameters presented on table 3. Since we used a very large population the selection pressure, together with the Elite E amount plays a very important role.

The results we are presenting are related to a particular execution were we stress tested the algorithm by not giving enough stock to attend all the orders, and allowing the packing and order information to be misplaced, therefore letting some spare products for the next execution. The expected overal % of η products being sent is around 82%, which was hitted early on the evolution, as can be seen on image 4. The images 3 and 5 presents the fitness and the amount of non deliverable η products over time. All these 3 images show a very early adoption of good local solutions, however from

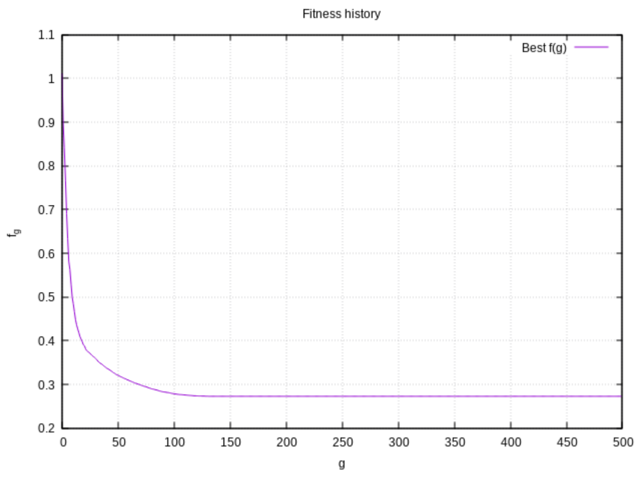

Figure 3: Fitness overt time for stress test execution

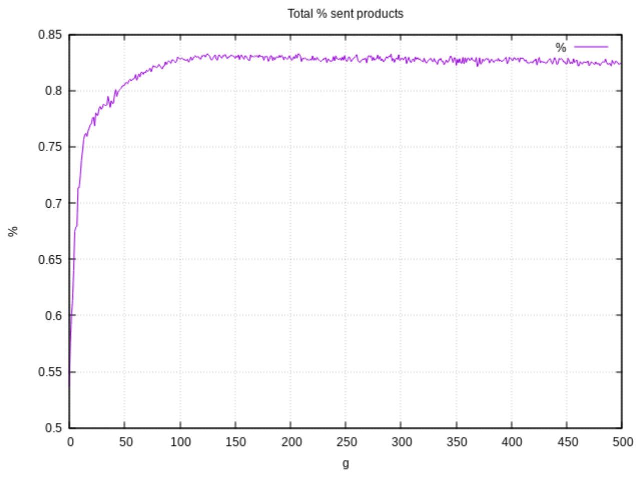

Figure 4: Deliverable % for stress test execution

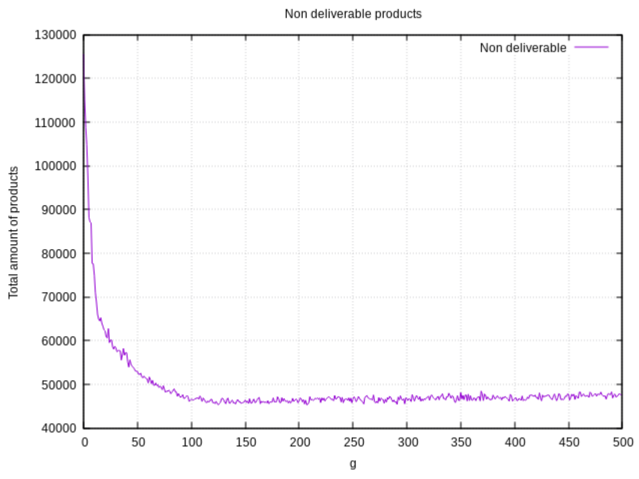

Figure 5: Non deliverable products over time

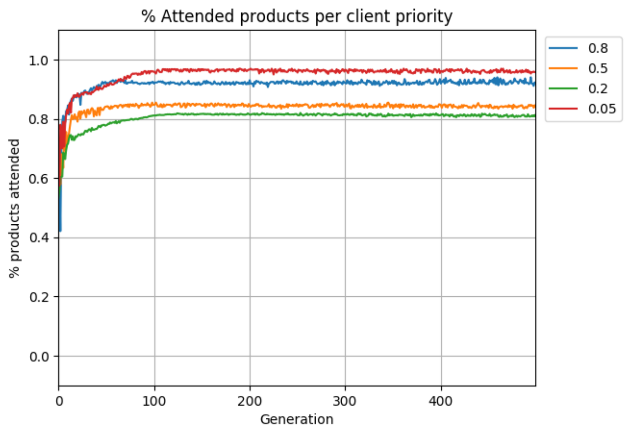

Figure 6: Clients evolution over time

image 4 it's possible to see that the GA is still searching for a global solution.

To better understand this we need to see the image 6, this plot shows the amount of % attended per ϕ_c (here we had 0.005, 0.2, 0.5 and 0.8 different ϕ_c). Since we could never reach 100%, the GA keeps trying to prioritize the $\phi_c = 0.8$, but the $\phi_c = 0.05$ is getting a higher priority, as it has enough η products on stock to sent to clients with this ϕ_c. This type of behaviour is expected and it's what makes this solution unique, if only the business rules was attended, we would have something like on generation 50, where there is more $\phi_c = 0.8$ being attended to, instead of $\phi_c = 0.05$. Yet we have a bigger fit if we prioritize the low ϕ_c in this specific case, therefore diminishing the cost of the f function, and achieving global optimal solutions.

What can be taken from those results is that, for a stress test scenario, the algorithm is still able to reach the global optimal solution, while still taking care of the multi objectives we had, that is:

- Select the amount of products η to use on each order o
- Use the ϕ_c and ϕ_η as a guide on how to attend each order o
- Attend each order o in time (minimize the T_o of each order)

6 FUTURE WORK

Even though the algorithm reached some pretty high end results, it's still far from over. Since this is a multi-objective GA, there may be different methods we could try to find the best fitness over time. Also, the GA is converging too quickly to the global optimal solution (less than 100 generations) even for a stress test scenario, this may lead to poor performance at the end.

Apart from all that, there is also the problem of the bin-packing that should take place **after** the amount of products to attend is found. This was not talked in this paper, but it is a very important factor to consider for real world scenarios.

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Cambridge, MA, USA: MIT Press, 1992.

[2] C. A. Silva, J. M. C. Sousa, and T. A. Runkler, "Rescheduling and optimization of logistic processes using GA and ACO," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 3, pp. 343 – 352, 2008.

[3] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.

[4] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[5] M. Dorigo and T. Stützle, *Ant Colony Optimization.* Scituate, MA, USA: Bradford Company, 2004.

[6] M. Mitchell, *Complexity: A Guided Tour.* New York, NY, USA: Oxford University Press, Inc., 2009.

[7] P. De Oliveira and M. Interciso, "Ternary representation improves the search for binary, one-dimensional density classifier cellular automata," in *2011 IEEE Congress of Evolutionary Computation, CEC 2011*, 2011.

[8] J. H. Holland, *Hidden Order: How Adaptation Builds Complexity.* Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995.

[9] J. H. Holland, *Emergence: From Chaos to Order.* Perseus Publishing, 1999.

[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1989.

[11] T. P. Runarsson and J. J. Merelo-Guervos, "Adapting Heuristic Mastermind Strategies to Evolutionary Algorithms," 2009.

[12] L. Berghman, D. Goossens, and R. Leus, "Efficient solutions for Mastermind using genetic algorithms," *Computers & Operations Research*, vol. 36, no. 6, pp. 1880–1885, 2009.

[13] J. Yang, C. Wu, H. Lee, and Y. Liang, "Solving traveling salesman problems using generalized chromosome genetic algorithm," *Progress in Natural Science*, vol. 18, pp. 887–892, jul 2008.

[14] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* Springer Publishing Company, Incorporated, 2nd ed., 2015.