Relationship between Genetic Algorithms and Ant Colony Optimization Algorithms

Osvaldo Gómez and Benjamín Barán

Centro Nacional de Computación
Universidad Nacional de Asunción
Paraguay
{ogomez, bbaran}@cnc.una.py
http://www.cnc.una.py

Abstract. Genetic Algorithms (GAs) were introduced by Holland as a computational analogy of adaptive systems. GAs are search procedures based on the mechanics of natural selection and natural genetics. Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behavior of ant colonies. ACO was introduced by Dorigo and has evolved significantly in the last few years. Both algorithms have shown their effectiveness in the resolution of hard combinatorial optimization problems. This paper shows the relationship between these two evolutionary algorithms inspired by different nature phenomena.

1 Introduction

A Genetic Algorithm (GA) is a randomized search method modeled on evolution and introduced by Holland [1]. GAs are being applied to a variety of problems and becoming an important tool in combinatorial optimization problems [2]. Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behavior of ant colonies [3]. In the last few years, ACO has empirically shown its effectiveness in the resolution of several different NP-hard combinatorial optimization problems [4]. So, GAs and ACO are evolutionary algorithms inspired by different nature phenomena. One example of a well known NP-hard combinatorial optimization problem is the Traveling Salesman Problem (TSP). This paper uses the TSP as a case study problem as previous works did [3, 5, 6].

The present work is organized as follows. The TSP is summarized in Section 2. In Section 3, the Simple Genetic Algorithm (SGA) and the Omicron Genetic Algorithm (OGA) are described. New strategies for OGA are proposed in Section 4. The standard ACO approach, the Population-based ACO (P-ACO) and the Omicron ACO are presented in Section 5. The relationship between GAs and ACO algorithms is shown in Section 6. Finally, the conclusions are given in Section 7.

2 Study Problem

In this paper the symmetric Traveling Salesman Problem (TSP) is used as a case study problem for comparing the analyzed algorithms. The TSP can be

represented by a complete graph $G = (N, A)$ with N being the set of nodes, also called cities, and A being the set of arcs fully connecting the nodes. Each arc (i, j) is assigned a value $d(i, j)$ which represents the distance between cities i and j. The TSP then is the problem of finding a shortest closed tour visiting each of the $n = |N|$ nodes of G exactly once. For symmetric TSPs, the distances between the cities are independent of the direction of traversing the arcs, that is $d(i, j) = d(j, i)$ for every pair of nodes. Suppose that r_x is a TSP tour or solution, then $l(r_x)$ denotes the length of the tour r_x.

3 Genetic Algorithms

Genetic Algorithms (GAs) were introduced by Holland as a computational analogy of adaptive systems [1]. GAs are search procedures based on the mechanics of natural selection and natural genetics. Next, the Simple Genetic Algorithm (SGA) presented by Goldberg in [2] - which may be considered as a standard approach - is described.

3.1 Simple Genetic Algorithm

The Simple Genetic Algorithm (SGA) has a population P of p individuals P_x representing solutions of an optimization problem. The SGA uses a binary codification of these solutions or individuals represented by strings. In a maximization context, the value of the objective function of every individual of P is considered its fitness. An initial population is chosen randomly, then a set of simple operations that uses and generates successive Ps is executed iteratively. It is expected that P improves over time until an end condition is reached.

The operators of the SGA are reproduction, crossover and mutation. Reproduction is a process in which p individuals are selected with a probability proportional to their fitness to be parents. Crossover is a process in which 2 different parents are iteratively selected from the set of p parents to swap information between them to generate 2 new individuals (offspring). This is done choosing randomly a point of break for the parents and swapping parts between them. Then mutation is applied to every offspring. Mutation is the alteration of the bits of an individual with a small predefined probability, sometimes known as mutation coefficient (mc). These new altered individuals compose the new population P. Next, the main pseudocode of the SGA is presented, where comments start with a % symbol.

Pseudocode of the *SGA*

```
g = 0                                  % Initialization of the generation counter
P = Initialize population()            % Random generation of p individuals
REPEAT UNTIL end condition
  F = Reproduction(P)                  % Selection of p parents through a roulette
  S = Crossover(F)                     % Swap of information between different pairs of parents
  M = Mutation(S)                      % Alteration of individuals' bits with probability mc
  P = Update population(M)             % M is copied to P (P = M)
  g = g + 1                            % Increment of the generation counter
```

3.2 Omicron Genetic Algorithm

The literature in evolutionary computation has defined a great variety of GAs that maintain the same philosophy, varying operators and adding different principles like elitism [2, 7]. Using the Simple Genetic Algorithm as a reference, this Section presents a new version, the Omicron Genetic Algorithm (OGA), a Genetic Algorithm designed specifically for the TSP.

Codification. The OGA has a population P of p individuals or solutions, as the SGA does. Every individual P_x of P is a valid TSP tour and is determined by the arcs (i, j) that compose the tour. Unlike the SGA, that uses a binary codification, the OGA uses an n-ary codification. Considering a TSP with 5 cities $c1$, $c2$, $c3$, $c4$ and $c5$, the tour defined by the arcs $(c1, c4)$, $(c4, c3)$, $(c3, c2)$, $(c2, c5)$ and $(c5, c1)$ will be codified with a string containing the visited cities in order, i.e. $\{c1, c4, c3, c2, c5\}$.

Reproduction. The OGA selects randomly two parents (F_1 and F_2) from the population P, as does an SGA reproduction. The selection of a parent is done with a probability proportional to the fitness of each individual P_x, where $fitness(P_x) \propto 1/l(P_x)$. Unlike the SGA, where two parents generate two offspring, in the OGA, both parents generate only one offspring. In the SGA, p offspring are obtained first to completely replace the old generation. In the OGA, once an offspring is generated, it replaces the oldest element of P. Thus, the population will be a totally new one in p iterations and it would be possible to consider this population a new generation. In conclusion, the same population exchange as in the SGA is made in the OGA, but in a progressive way.

Crossover and Mutation. The objective of the crossover in the SGA is that the offspring share information of both parents. In the mutation, the goal is that new information is added to the offspring, and therefore is added to the population. In the SGA, the operators crossover and mutation are done separately. To facilitate the obtaining of offspring who represent valid tours in OGA, the crossover and the mutation are done in a single operation called Crossover-Mutation (CM). Even so, the objectives of both operators previously mentioned will stay intact.

To perform CM, the arcs of the problem are represented in a roulette, where every arc has a weight w or a probability to be chosen. CM gives a weight w of 1 to each arc (i, j) belonging to set A, i.e. $w_{ij} = 1\ \forall (i, j) \in A$. Then, a weight of $O/2$ is added to each arc (i, j) of F_1, i.e. $w_{ij} = w_{ij} + O/2\ \forall (i, j) \in F_1$, where Omicron ($O$) is an input parameter of the OGA. Analogously, a weight of $O/2$ is added to each arc (i, j) of F_2. Iteratively, arcs are randomly taken using the roulette to generate a new offspring. While visiting city i, consider $\mathcal{N}_i$ as the set of cities not yet visited and that allows the generation of a valid tour. Therefore, only the arcs $(i, j)\ \forall j \in \mathcal{N}_i$ participate in the roulette, with their respective weights w_{ij}. Even so the crossover is done breaking the parents

and interchanging parts in the SGA instead of taking arcs iteratively with high probability from one of the parents in the OGA, the philosophy of both crossover operators is the same.

To generate an offspring S_1, an arc of one of the parents will be selected with high probability (similar to crossover). But it is also possible to include new information since all the arcs that allow the creation of a valid tour participate in the roulette with probability greater than 0 (similar to mutation). The value $O/2$ is used because there are two parents, and then $w_{max} = O + 1$ can be interpreted as the maximum weight an arc can have in the roulette (when the arc belongs to both parents). When the arc does not belong to either parent, it obtains the minimum weight w_{min} in the roulette, that is $w_{min} = 1$. Then, O determines the relative weight between crossover and mutation.

Formally, while visiting city i, the probability of choosing an arc (i, j) to generate the offspring S_1 is defined by equation (1).

$$\mathcal{P}_{ij} = \begin{cases} \frac{w_{ij}}{\sum_{\forall h \in \mathcal{N}_i} w_{ih}} & \text{if } j \in \mathcal{N}_i. \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Next, the main pseudocode of the OGA is presented.

Pseudocode of the *OGA (version 1)*

```
g = 0                                   % Initialization of the generation counter
P = Initialize population()             % Random generation of p individuals
REPEAT UNTIL end condition
  REPEAT p TIMES
    {F1, F2} = Reproduction(P)          % Selection of 2 parents through a roulette
    S1 = CM({F1, F2}, 1)                % Generation of 1 offspring using {F1, F2}
    P = Update population(S1)           % S1 replaces the oldest individual of P
  g = g + 1                             % Increment of the generation counter
```

The above version was designed to emphasize the similarity with the SGA. However, the next version is simpler and completely equivalent.

Pseudocode of the *OGA (version 2)*

```
u = 0                                   % Initialization of the iteration counter
P = Initialize population()             % Random generation of p individuals
REPEAT UNTIL end condition
  {F1, F2} = Reproduction(P)            % Selection of 2 parents through a roulette
  S1 = CM({F1, F2}, 1)                  % Generation of 1 offspring using {F1, F2}
  P = Update population(S1)             % S1 replaces the oldest individual of P
  u = u + 1                             % Increment of the iteration counter
```

Example of an OGA's Iteration. To clarify the previous procedure, an example considering the TSP with 5 cities mentioned above is presented next. $O = 4$ and $p = 4$ are considered for this case.

Reproduction: The example assumes an initial population $P = \{P_x\}$ composed of 4 randomly selected individuals with their respective fitnesses f_x. This initial population is presented next.

First randomly chosen individual: $P_1 = \{c1, c4, c3, c2, c5\}$ with $f_1 = 10$

Second randomly chosen individual: $P_2 = \{c1, c3, c2, c5, c4\}$ with $f_2 = 8$

Third randomly chosen individual: $P_3 = \{c3, c5, c1, c2, c4\}$ with $f_3 = 1$

Fourth randomly chosen individual: $P_4 = \{c2, c5, c4, c1, c3\}$ with $f_4 = 5$

Two parents are randomly selected through a roulette, where the weights of the individuals in the roulette are their fitness. It is assumed that individuals P_1 and P_4 are selected to be parents.

$F_1 = \{c1, c4, c3, c2, c5\} = \{(c1, c4), (c4, c3), (c3, c2), (c2, c5), (c5, c1)\}$

$F_2 = \{c2, c5, c4, c1, c3\} = \{(c2, c5), (c5, c4), (c4, c1), (c1, c3), (c3, c2)\}$

CM. Iteration 1: First, an initial city is randomly chosen to perform CM. $c4$ is assumed as the initial city. Then, $\mathcal{N}_{c4}$ is composed by $\{c1, c2, c3, c5\}$, i.e. the set of not yet visited cities. The arc $(c4, c2)$ has a weight of 1 in the roulette because it does not belong to either parent. Arcs $\{(c4, c3), (c4, c5)\}$ have a weight of $1 + \frac{O}{2} = 3$ in the roulette because they belong to one parent. Finally, the arc $(c4, c1)$ has a weight of $1 + O = 5$ in the roulette because it belongs to both parents. It is assumed that the arc $(c4, c3)$ is randomly chosen through the roulette.

CM. Iteration 2: $\mathcal{N}_{c3}$ is composed by $\{c1, c2, c5\}$. The arc $(c3, c5)$ has a weight of 1 in the roulette because it does not belong to either parent. The arc $(c3, c1)$ has a weight of $1 + \frac{O}{2} = 3$ in the roulette because it belongs to one parent. Finally, the arc $(c3, c2)$ has a weight of $1 + O = 5$ in the roulette because it belongs to both parents. It is assumed that the arc $(c3, c2)$ is randomly chosen through the roulette.

CM. Iteration 3: $\mathcal{N}_{c2}$ is composed by $\{c1, c5\}$. The arc $(c2, c1)$ has a weight of 1 in the roulette because it does not belong to either parent. Finally, the arc $(c2, c5)$ has a weight of $1 + O = 5$ in the roulette because it belongs to both parents. It is assumed that the arc $(c2, c1)$ is randomly chosen through the roulette.

CM. Iteration 4: $\mathcal{N}_{c1}$ is composed by $\{c5\}$. The arc $(c1, c5)$ has a weight of $1 + \frac{O}{2} = 3$ in the roulette because it belongs to one parent. The arc $(c1, c5)$ is chosen because it is the unique arc represented in the roulette.

In conclusion, the new offspring is $S_1 = \{c4, c3, c2, c1, c5\} = \{(c4, c3), (c3, c2), (c2, c1), (c1, c5), (c5, c4)\}$. Notice that S_1 has 3 arcs of F_1 $\{(c4, c3), (c3, c2), (c1, c5)\}$ and 2 arcs of F_2 $\{(c3, c2), (c1, c5)\}$. Also, S_1 has an arc $\{(c2, c1)\}$ that does not belong to either parent. This shows that the objectives of the operators (crossover and mutation) have not been altered.

Population Update: The new individual S_1 replaces the oldest individual P_1. Next, the new population is shown.

$P_1 = \{c4, c3, c2, c1, c5\}$ with $f_1 = 7$

$P_2 = \{c1, c3, c2, c5, c4\}$ with $f_2 = 8$

$P_3 = \{c3, c5, c1, c2, c4\}$ with $f_3 = 1$

$P_4 = \{c2, c5, c4, c1, c3\}$ with $f_4 = 5$

The entire procedure above is done iteratively until an end condition is satisfied. Note that OGA is another version of GA like many other published versions.

4 New Strategies for OGA

New strategies referred to crossover, population update and heuristic information are proposed for OGA. Considering that the most relevant aspect mentioned above is the crossover of multiple parents, this new version is called Multi-Parent OGA (MOGA).

4.1 Crossover

This Section considers the generation of the offspring through the crossover of multiple parents. This idea is not new and it was proposed before [7]. More specifically, this strategy proposes that the p individuals of P are parents without any roulette intervention. Obviously, this crossover of p parents eliminates competition among individuals during reproduction. Nevertheless, the new population update strategy proposed in the next Section will solve this competition problem. Considering that there are p parents instead of 2, a weight of O/p is added to each arc (i,j) belonging to every F_x, i.e. $w_{ij} = w_{ij} + O/p \; \forall (i,j) \in F_x$. This way, when an arc belongs to the p parents, the weight of the arc will be $w_{max} = O + 1$. When an arc does not belong to either parent, the weight of the arc will be $w_{min} = 1$. This is done to maintain the weight limits (w_{max} and w_{min}).

4.2 Population Update

To reduce the possibilities that a bad individual enters the population, a competition strategy among offspring is considered. This new strategy replaces the most traditional parent competition strategy. This strategy consists on the generation of t offspring $\{S_1, ..., S_t\}$ in one iteration. Only the offspring with the best fitness ($S_{best} \in \{S_1, ..., S_t\}$) is chosen to enter P. As in the OGA population update strategy, S_{best} replaces the oldest individual of the population. Notice that the same effect is obtained (competition among individuals) with a different strategy.

4.3 Heuristic Information

Good TSP tours are composed with high probability by arcs with short length. Thus, it seems a good idea to give them better weights in the roulette. Then, considering the heuristic information $\eta_{ij} = 1/d(i,j)$, the probability of choosing an arc (i,j) to generate the offspring S_1 is now defined by equation (2).

$$\mathcal{P}_{ij} = \begin{cases} \frac{w_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{\forall h \in \mathcal{N}_i} w_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}} & \text{if } j \in \mathcal{N}_i. \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Where the input parameters α and β are defined as the relative influence between the weight of genetic information and the heuristic information η. Next, the main pseudocode of the MOGA is presented.

Pseudocode of the *MOGA*

```
u = 0                                   % Initialization of the iteration counter
P =Initialize population()              % Random generation of p individuals
REPEAT UNTIL end condition
  S =CM(P,t)                            % Generation of t offspring using P as parents
  S_best = best element of S
  P =Update population(S_best)          % S_best replaces the oldest individual of P
  u = u + 1                             % Increment of the iteration counter
```

So far several versions of GAs have been analyzed. Before the relationship of both algorithms is presented, ACO versions are considered in the next Section.

5 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic inspired by the behavior of ant colonies [3]. In the last few years, elitist ACO has received increased attention by the scientific community as can be seen by the growing number of publications and its different fields of application [4]. Even though there exist several ACO variants, the one that may be considered a standard approach is presented next [8].

5.1 Standard Approach

ACO uses a pheromone matrix $\tau = \{\tau_{ij}\}$ for the construction of potential good solutions. The initial values of τ are set $\tau_{ij} = \tau_{init}\ \forall(i,j)$, where $\tau_{init} > 0$. It also takes advantage of heuristic information using $\eta_{ij} = 1/d(i,j)$. Parameters α and β define the relative influence between the heuristic information and the pheromone levels. While visiting city i, $\mathcal{N}_i$ represents the set of cities not yet visited and the probability of choosing a city j at city i is defined as

$$\mathcal{P}_{ij} = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{\forall h \in \mathcal{N}_i} \tau_{ih}^{\alpha} \cdot \eta_{ih}^{\beta}} & \text{if } j \in \mathcal{N}_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

At every generation of the algorithm, each of the m ants constructs a complete tour using (3), starting at a randomly chosen city. Pheromone evaporation is applied for all (i,j) according to $\tau_{ij} = (1-\rho) \cdot \tau_{ij}$, where parameter $\rho \in (0,1]$ determines the evaporation rate. Considering an elitist strategy, the best solution found so far r_{best} updates τ according to $\tau_{ij} = \tau_{ij} + \Delta\tau$, where $\Delta\tau = 1/l(r_{best})$ if $(i,j) \in r_{best}$ and $\Delta\tau = 0$ if $(i,j) \notin r_{best}$. For one of the best performing ACO algorithms, the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MMAS}$) [6], minimum and maximum values are imposed to τ (τ_{min} and τ_{max}).

5.2 Population-based ACO

The Population-based ACOs (P-ACOs) were designed by Guntsch and Middendorf [8] for dynamic combinatorial optimization problems. As the standard approach, the initial values of τ are set $\tau_{ij} = \tau_{init}\ \forall(i,j)$, where $\tau_{init} > 0$. The P-ACO approach updates in a different way the pheromone information than the standard approach. The P-ACO derives the pheromone matrix through a population $Q = \{Q_x\}$ of q good solutions or individuals as follows. First, at every iteration each of the m ants constructs a solution using probabilities given in equation (3), the best solution enters the population Q. Whenever a solution Q_{in} enters the population, then τ_{ij} is updated according to $\tau_{ij} = \tau_{ij} + \Delta\tau$, where $\Delta\tau = \Delta$ if $(i,j) \in Q_{in}$ and $\Delta\tau = 0$ if $(i,j) \notin Q_{in}$. After the first q solutions enter Q, i.e. the initialization of the population is finished, one solution Q_{out} must leave the population at every iteration. The solution that must leave the population is decided by an update strategy. Whenever a solution Q_{out} leaves the population, then $\tau_{ij} = \tau_{ij} - \Delta\tau$, where $\Delta\tau = \Delta$ if $(i,j) \in Q_{out}$ and $\Delta\tau = 0$ if $(i,j) \notin Q_{out}$. P-ACO replaces the pheromone evaporation used by the standard approach in this way. The value Δ is a constant determined by the following input parameters, size of the population q, minimum or initial pheromone level τ_{init} and maximum pheromone level τ_{max}. Thus, $\Delta = (\tau_{max} - \tau_{init})/q$ [8].

FIFO-Queue Update Strategy. The FIFO-Queue update strategy was the first P-ACO strategy designed [8], trying to simulate the behavior of the standard approach of ACO. In the FIFO-Queue update strategy, Q_{out} is the oldest individual of Q. Next, the main pseudocode of the P-ACO FIFO-Queue is presented.

Pseudocode of the *P-ACO FIFO-Queue*

```
u = 0                                          % Initialization of the iteration counter
Q = Initialize population()                    % Generation of q individuals
REPEAT UNTIL end condition
  τ = Update pheromone matrix(Q)               % Update τ using Q
  T = Construct solutions(τ)                   % Generation of m solutions using τ
  T_best = best solution of T
  Q = Update population(T_best)                % T_best replaces the oldest individual of Q
  u = u + 1                                    % Increment of the iteration counter
```

Quality Update Strategy. A variety of strategies was studied in [9], one of them is the *Quality* update strategy. The worst solution (considering quality) of the set $\{Q, Q_{in}\}$ leaves the population in this strategy. This ensures that the best solutions found so far make up the population.

5.3 Omicron ACO

In the search for a new ACO analytical tool, Omicron ACO (OA) was developed [5]. OA was inspired by $\mathcal{MMAS}$, an elitist ACO currently considered among

the best performing algorithms for the TSP [6]. It is based on the hypothesis that it is convenient to search nearby good solutions [6].

The main difference between the $\mathcal{MMAS}$ and the OA is the way the algorithms update the pheromone matrix. In the OA, a constant pheromone matrix τ^0 with $\tau^0_{ij} = 1$, $\forall i, j$ is defined. OA maintains a population $Q = \{Q_x\}$ of q individuals or solutions, the best unique ones found so far. The best individual of Q at any moment is called Q^*, while the worst individual Q_{worst}.

In the OA the first population is chosen using τ^0. At every iteration a new individual Q_{new} is generated, replacing $Q_{worst} \in Q$ if Q_{new} is better than Q_{worst} and different from any other $Q_x \in Q$. In other words, if $Q_{new} \notin Q \rightarrow Q_{new}$ replaces the worst element of Q^+, where $Q^+ = \{Q, Q_{new}\}$. After K iterations, τ is recalculated using the input parameter Omicron (O). First, $\tau = \tau^0$; then, O/q is added to each element τ_{ij} for each time an arc (i, j) appears in any of the q individuals present in Q. The above process is repeated every K iterations until the end condition is reached. Note that $1 \leq \tau_{ij} \leq (1 + O)$, where $\tau_{ij} = 1$ if arc (i, j) is not present in any Q_x, while $\tau_{ij} = (1 + O)$ if arc (i, j) is in every Q_x.

Even considering their different origins, OA results similar to the P-ACO algorithms described above [8, 9]. The main difference between the OA and the *Quality* Strategy of P-ACO is that OA does not allow identical individuals in its population. Also, OA updates τ every K iterations, while P-ACO updates τ every iteration. Next, the main pseudocode of the OA is presented.

Pseudocode of the *Omicron ACO*

```
u = 0                                          % Initialization of the iteration counter
Q = Initialize population()                    % Generation of q individuals
REPEAT UNTIL end condition
  τ = Update pheromone matrix(Q)               % Update τ using Q
  REPEAT K TIMES
    Q_new = Construct a solution(τ)            % Generation of 1 solution using τ
    IF Q_new ∉ Q
      Q = Update population(Q_new)             % Q_new replaces the worst individual of Q+
    u = u + 1                                  % Increment of the iteration counter
```

6 Relationship between GAs and ACO

Considering a P-ACO FIFO-Queue and a MOGA with the same population size (i.e. $q = p$), the same number of ants or offspring (i.e. $m = t$) and the same τ_{ij}, w_{ij} limits (i.e. $\tau_{max} = O + 1$ and $\tau_{init} = 1$), the amount of pheromones an ant deposits and the genetic weight of an individual are the same also (i.e. $(\tau_{max} - \tau_{init})/q = O/p$). Paying attention to the pheromone matrix update and the roulette generation explained before, it is easy to see that both procedures are identical. Besides, the solution construction procedures and probabilities for both algorithms, shown by equations (2) and (3), are the same.

Although the initialization of the population of the P-ACO FIFO-Queue is not entirely at random as the MOGA, this aspect is irrelevant for these algo-

rithms main functionality. Consequently, both algorithms are the same as it is shown in the next pseudocodes, where the CM is divided in two stages.

Pseudocode of the *MOGA*

```
u = 0                                          % Initialization of the iteration counter
P = Initialize population()                    % Random generation of p individuals
REPEAT UNTIL end condition
  R = Generate roulette(P)                     % Generation of the roulette using P as parents
  S = CM(R, t)                                 % Generation of t offspring through the roulette
  S_best = best element of S
  P = Update population(S_best)                % S_best replaces the oldest individual of P
  u = u + 1                                    % Increment of the iteration counter
```

Pseudocode of the *P-ACO FIFO-Queue*

```
u = 0                                          % Initialization of the iteration counter
Q = Initialize population()                    % Generation of q individuals
REPEAT UNTIL end condition
  τ = Update pheromone matrix(Q)               % Update τ using Q
  T = Construct solutions(τ)                   % Generation of m solutions using τ
  T_best = best solution of T
  Q = Update population(T_best)                % T_best replaces the oldest individual of Q
  u = u + 1                                    % Increment of the iteration counter
```

6.1 New Strategy for MOGA

In this Section a new population update strategy for MOGA is proposed. Consider that S_{best} replaces the worst element of the set $\{P, S_{best}\}$ instead of the oldest individual, i.e. if $fitness(S_{best})$ is smaller than $fitness(P_{worst})$ then S_{best} replaces P_{worst} in P, where P_{worst} is the worst individual of P. This new strategy can be seen as a super-elitism due to the fact that P is composed by the best individuals found so far. Thus, this version is called Super-Elitist MOGA (EMOGA). At this point it is easy to conclude that EMOGA is equivalent to the *Quality* version of P-ACO.

6.2 New Strategy for EMOGA

An undesired characteristic of EMOGA is that one very good individual can dominate the population. In this way, the genetic diversity, a desirable property of GAs, is strongly reduced. A simple way to avoid this problem is to modify the population update strategy. So, in this new strategy S_{best} replaces P_{worst} only if it is different to the whole population. Due to the competition among individuals imposed by this population update strategy, the offspring competition is useless and will be eliminated. Besides, because of the little dynamism caused by this population update strategy, the roulette will be generated every H iterations without significant consequences.

This new version is called Diversified EMOGA (DEMOGA) because it forces genetic diversification. The main pseudocode of the DEMOGA and the OA are presented next to show that both algorithms are similar.

Pseudocode of the *DEMOGA*

$u = 0$ % Initialization of the iteration counter
$P =$*Initialize population()* % Random generation of p individuals
REPEAT UNTIL `end condition`
  $\mathcal{R} =$*Generate roulette(P)* % Generation of the roulette using P as parents
  REPEAT H TIMES
    $S_1 =$*CM($\mathcal{R}$, 1)* % Generation of 1 offspring using the roulette
    IF $S_1 \notin P$
      $P =$*Update population(S_1)* % S_1 replaces the worst individual of $\{P, S_1\}$
    $u = u + 1$ % Increment of the iteration counter

Pseudocode of the *Omicron ACO*

$u = 0$ % Initialization of the iteration counter
$Q =$*Initialize population()* % Generation of q individuals
REPEAT UNTIL `end condition`
  $\tau =$*Update pheromone matrix(Q)* % Update τ using Q
  REPEAT K TIMES
    $Q_{new} =$*Construct a solution(τ)* % Generation of 1 solution using τ
    IF $Q_{new} \notin Q$
      $Q =$*Update population(Q_{new})* % Q_{new} replaces the worst individual of Q^+
    $u = u + 1$ % Increment of the iteration counter

To clarify the origins and equivalences of so many versions, a summarized chart is presented in Figure 1.

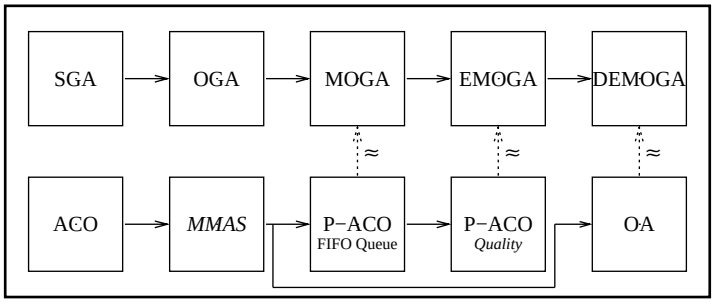

Fig. 1. Summarized chart for the versions of GAs and ACO

7 Conclusions

This work shows that a version of Genetic Algorithm - MOGA -, designed similarly to many others in the literature, is the same algorithm as a Population-based ACO algorithm, called P-ACO FIFO-Queue. Besides, this paper explains that the Omicron ACO is a Genetic Algorithm with multiple parents, super elitism and forced genetic diversity.

Finally, it can be concluded that GAs and ACO algorithms use the same principles to succeed in the combinatorial optimization problem TSP.

References

1. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975)
2. Goldberg, D. In: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
3. Dorigo, M., Di Caro, G.: The Ant Colony Optimization Meta-Heuristic. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London (1999) 11–32
4. Dorigo, M., Stützle, T.: The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. In Glover, F., Kochenberger, G., eds.: Handbook of Metaheuristics. Volume 57 of International Series in Operations Research and Management Science. Kluwer Academic Publishers (2003)
5. Gómez, O., Barán, B.: Arguments for ACO's Success. In Cantú-Paz, E., Foster, J.A., Deb, K., Davis, D., Roy, R., O'Reilly, U.M., Beyer, H.G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A.C., Dowsland, K., Jonoska, N., Miller, J., eds.: Genetic and Evolutionary Computation – GECCO-2004. LNCS, Seattle, Springer-Verlag (2004) to appear
6. Stützle, T., Hoos, H.H.: $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. Future Generation Computer Systems **16** (2000) 889–914
7. Mühlenbein, H., Voigt, H.M.: Gene Pool Recombination in Genetic Algorithms. In Osman, I.H., Kelly, J.P., eds.: Proceedings of the Meta-heuristics Conference, Norwell, USA, Kluwer Academic Publishers (1995) 53–62
8. Guntsch, M., Middendorf, M.: A Population Based Approach for ACO. In Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Raidl, G., eds.: Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim. Volume 2279., Kinsale, Ireland, Springer-Verlag (2002) 71–80
9. Guntsch, M., Middendorf, M.: Applying Population Based ACO to Dynamic Optimization Problems. In: Ant Algorithms, Proceedings of Third International Workshop ANTS 2002. Volume 2463 of LNCS. (2002) 111–122