Golden Ball: A Novel Meta-Heuristic to Solve Combinatorial Optimization Problems Based on Soccer Concepts

Article *in* Applied Intelligence · December 2013

DOI: 10.1007/s10489-013-0512-y

CITATIONS
60

READS
439

3 authors:

Eneko Osaba
Tecnalia Corporación Tecnológica
152 PUBLICATIONS **1,514** CITATIONS

SEE PROFILE

Fernando Díaz
Universidad Andrés Bello
27 PUBLICATIONS **298** CITATIONS

SEE PROFILE

Enrique Onieva
University of Deusto
150 PUBLICATIONS **2,424** CITATIONS

SEE PROFILE

Applied Intelligence manuscript No.
(will be inserted by the editor)

Golden Ball: A Novel Meta-Heuristic to Solve Combinatorial Optimization Problems Based on Soccer Concepts

E. Osaba · F. Diaz · E. Onieva

DOI: 10.1007/s10489-013-0512-y. The final publication is available at link.springer.com

Abstract In this paper, a new multiple population based meta-heuristic to solve combinatorial optimization problems is introduced. This meta-heuristic is called Golden Ball (GB), and it is based on soccer concepts. To prove the quality of our technique, we compare its results with the results obtained by two different Genetic Algorithms (GA), and two Distributed Genetic Algorithms (DGA) applied to two well-known routing problems, the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP). These outcomes demonstrate that our new meta-heuristic performs better than the other techniques in comparison. We explain the reasons of this improvement.

Keywords Meta-heuristics · Golden Ball · Distributed Genetic Algorithm · Routing Problems · Combinatorial Optimization · Intelligent Transportation Systems

1 Introduction

Nowadays, combinatorial optimization is a widely studied field in artificial intelligence, which is subject of a large number of works every year [1,2]. There exist lots of problems of this type, being the Traveling Salesman Problem (TSP) [3] and the job-shop scheduling problem [4] two of the most studied. The interest for the resolution of these problems lies on their complexity and applicability to real life. Being NP-Hard [5], a large number of studies focused on solving them are published annually, using a wide variety of techniques.

Normally, techniques used to solve these kinds of problems can be divided into two groups: heuristics and meta-heuristics. Both alternatives are designed

Deusto Institute of Technology (DeustoTech), University of Deusto
Av. Universidades 24, Bilbao 48007, Spain
Tel.: +34 944 139003 (ext. 2050)
E-mail: e.osaba@deusto.es

to optimize a problem iteratively, trying to improve one or more solutions. Those techniques explore the solution space with the aim of finding the best solution. Anyway, taking into account the complexity of the problems on which heuristics and meta-heuristics are applied, in many cases it is not possible to find the optimal solution. This way, the solution provided by these techniques has to get as close as possible to the optimal.

A heuristic is an optimization technique that solves a problem using specific information and knowledge of that problem. This way, heuristics explore the space of feasible solutions intensifying the search in the most promising areas, in order to achieve good optimization results quickly. On the other hand, a meta-heuristic is an optimization technique that solves a specific problem using only general information and knowledge common to all optimization problems. Meta-heuristics explore a larger area of the solution space in order to achieve good optimization results with independence of the problem.

Throughout history, there have been a lot of heuristics and meta-heuristics, each one with a different philosophy and characteristics, and applicable to many types of problems. They can be applied in a wide range of fields, like transport [6–10], software engineering [11–13] or industry [14–18]. As an example of these techniques the Simulated Annealing [19,20] and Tabu Search [21,22] algorithms can be found as local search algorithms. These meta-heuristics are based on a single solution, which is subjected to an optimization process that attempts to bring it closer to the global optimum as much as possible. Other widely used methods are the population based algorithms, which have a population of solutions that interact in a cooperative and competitive way. Examples of these methods are Genetic Algorithms (GA)[23–25] and Ant Colony Systems [26,27].

The main objective of this paper is to describe a new meta-heuristic to solve combinatorial optimization problems. This new meta-heuristic is a multiple population based algorithm, and we call it Golden Ball (GB). Like other population based techniques, GB starts the execution with the creation of a population of solutions. Then, it divides the different solutions of the problem in different teams, which work independently, and face each other in a competition. This competition is crucial to decide the transfer of solutions between teams and to decide the training model of each team.

In this paper, we demonstrate that our new meta-heuristic is a good alternative to solve various combinatorial optimization problem. To prove its quality, we have used our technique to solve two well-known routing problems and we have compared the obtained results with the ones obtained by two versions of the classical Genetic Algorithm and two Distributed Genetic Algorithms (DGA). The reasons for choosing GA and DGA to do the comparison, as well as the characteristics of the tests, are explained later.

The structure of this paper is as follows. A brief literature about multi-population techniques in Section 2. Then, in Section 3, we explain in detail our new meta-heuristic, and we mention the originality that our algorithm provides respect to existing ones. This is followed by the description of the different experiments (Section 4), and two test phases over the TSP and the

CVRP, that will be shown, respectively in Sections 5 and 6. We finish this paper with the conclusions of the study and further work (Section 7).

2 State of the art and related literature

As we have said in the introduction, *GB* is a multiple population technique, which means that the algorithm divides the entire initial population into different subpopulations. Throughout history some approaches have been developed following this philosophy. Below we describe some of most popular techniques of this type. We also discuss the literature of these techniques regarding routing problems, since in this work we use two problems of this type to perform the tests.

2.1 Parallel Artificial Bee Colony (PABC)

Single Artificial Bee Colony (ABC) algorithm was proposed by Karaboga in 2005 for multi-modal and multi-dimensional numeric problems [28]. ABC is a swarm based meta-heuristic that emulates foraging behavior of honey bees. The population of this technique consists in a colony in which three types of bees inhabit: employed, onlooker and scout ones; each of them with a different behavior. In ABC, a possible solution to an optimization problem is represented by a food source, and the fitness of the solution is depicted as the nectar amount of this source. Each employed bee has assigned a food source. Every iteration, this kind of bees finds a new food source, and depending on the nectar of the new source, the old one is replaced by the new one. Then, each onlooker bee selects a food source and it makes the same movement as the employed bees. A food source is abandoned if its quality has not been improved for a predetermined number of successive iterations. Then, the employed bee assigned to that source becomes a scout, and searches for a new food source randomly. This way, in ABC, exploitation is carried out by employed and onlooker bees, and the exploration by scouts.

A detailed survey on ABC can be found in [29]. That study notes that the works focused on an ABC applied to a routing problem are scarce (less than 10), since ABC initially was proposed for solving numerical problems and problems with continuous search spaces.

First versions of the Parallel ABC (PABC) appeared in 2009 [30], to perform a more thorough search than simple ABC. Anyway, to this day, there are only a few studies focused on that kind of version of the ABC. Regarding the way of working of PABCs, the main philosophy is the run of multiple instances of ABC, trying to increase the exploration capacity of the single ABC. Another crucial factor in PABC, which also helps the exploration and exploitation, is the bee migration or information exchange between different instances or subpopulations.

Some of the most interesting examples of PABC are the following. In [31] is presented an approach in which the problem variables are divided by different

instances of the same algorithm. All instances work independently, and the complete solution is obtained as the collection of the best solutions provided by all them. In [32], bees are divided into different subgroups, which explore the same solution space in the same way. Every iteration, two swarms are randomly selected to share their best solution. The best local solution from one swarm replaces the worst local solution on the other group. In [33], an improved version of the PABC is presented, where a local search enhances the search. In this work, three different parallel models are presented: master-slave, multi-hive with migrations, and hybrid hierarchical.

2.2 Parallel Particle Swarm Optimization (PPSO)

Particle Swarm Optimization (PSO) algorithm was first presented by Eberhart and Kennedy in 1995 [34]. PSO is one of the most used techniques in the swarm intelligence field, and although it was not initially designed to be applied to discrete problems, several modifications have made it possible. PSO was developed under the inspiration of behavior of bird flocks, fish schools and human communities. It works with a population (called swarm) of candidate solutions (called particles). The movements of every particle are guided by a parameter called velocity. This parameter is based on the actual position of the particle, and its best known position in the search space, as well as the best known position of the entire swarm.

To overcome the drawbacks of the conventional version of PSO (fast convergence to a local optimum [35,36], difficult balancing between exploration and exploitation [37], or the called "curse of dimensionality" [38]) first parallel and cooperative PSOs were introduced. As in PABC, in literature various methods have been proposed to implement the Parallel PSO.

In [38] a PSO based on cooperative swarms was introduced, which divides the search space into lower dimensional subspaces. The k-means and the regular split schemes are applied to split the solution space into swarms. Then, swarms optimize the different components of the solution space cooperatively. In [39] two different approaches (the competitive and the collaborative version) of the master-slave model are presented. In the competitive version, the master swarm enhances its particles taking the most fitted particle in all the slave swarm. In contrast, in the collaborative version, the master swarm updates its particles, depending on its ability to collaborate with slave swarms. In [40], the whole population is divided into different subgroups, which communicate with each other using three communication strategies. In the first communication strategy, multiple copies of the best particle of each swarm are mutated, and those mutated particles migrate and replace the worst particles in the other swarms. In the second communication strategy, best particle in each swarm is migrated to its neighbor swarms to replace some of the more poorly performing particles. The last communication strategy is a hybrid between the above two.

2.3 Parallel Genetic Algorithm (PGA)

Since its proposal in the 70's, GAs have become one of the most successful meta-heuristics for solving a wide range of problems. As in the other alternatives, with the aim of overcoming the drawbacks of GAs, such as premature convergence to a local optimum, and the imbalance between exploration and exploitation, Parallel GAs were proposed. PGAs are particularly easy to implement and promise substantial gains in performance. Reviewing the literature it can be seen that there are different ways to parallelize GA. The generally used classification divides parallel GAs in three categories: Fine Grain [41], Panmitic model or global single-population master-slave GAs [42] and Island model [43].

This last category is the most used, and it consists in a multiple populations that evolve separately most of the time and exchange individuals occasionally. These algorithms are known with different names, as *multi-deme*, *distributed* or *Coarse-Grained*. Many studies agree that this approach to develop parallel GAs is the most suitable, although the way and frequency of migration and the communication between the different populations are complex issues when implementing a GA of this type. There are a lot of papers in the literature describing innumerable aspects and details of the implementation of this kind of GAs [43–46]. In [47], a comprehensive survey about parallel GAs can be found. In this study, we can find several approaches for the migration and communication between different populations.

2.4 Imperialist Competitive Algorithm (ICA)

This meta-heuristic, proposed by Atashpaz-Gargari and Lucas [48], is based on the concept of imperialism, and divides the population into various empires, which evolve independently. Individuals are called countries, and they are divided into two types: imperialist states (best country of the empire) and colonies. In this technique, the colonies make their movement through the solution space basing on the imperialist state. Each movement is made in the direction in which the imperialist state is placed, trying to get close to it. Meanwhile, empires compete between them, trying to conquer the weakest colonies of each other. This way, powerless empires could collapse and disappear, dividing their colonies among other empires. Finally, all the empires collapse, except the most powerful one, and all the colonies will be under the control of this empire. In that moment, the convergence is met, and the execution finishes. To this day, this algorithm has only been applied to routing problems on a couple of occasions [49,50].

2.5 Seeker Optimization Algorithm (SOA)

Seeker optimization algorithm was proposed by Day et al. in 2006 [51]. This algorithm models the human searching behaviors, based on their memory,

experience, uncertainty reasoning and communication with each other. In this algorithm individuals are called seeker (or agent), and they are divided into K different subpopulations, in order to search over several different domains of the search space. All the seekers in the same subpopulation constitute a neighborhood. The seekers move along the solution space following a search direction, based on the current or historical positions of themselves or their neighbors (it depends on the behavior of each seeker). To avoid the rapid convergence of the subpopulations, the position of the worst $K-1$ seekers of each subpopulation are combined with the best one in each of other $K-1$ subpopulations, using a crossover operator. In the literature there are multiple studies focused on this technique [52–54], but until today, this algorithm has not yet been applied to any routing problem.

3 Golden Ball meta-heuristic

In this section, the new meta-heuristic Golden Ball is introduced. Firstly, in Section 3.1, a general description is made. Later, each phase of the algorithm is explained in detail. These phases are called initialization (Section 3.2), and competition (Section 3.3). Then, the termination criterion of the algorithm is defined (Section 3.4). Finally, this section ends explaining the original contribution of the GB technique (Section 3.5).

3.1 General description

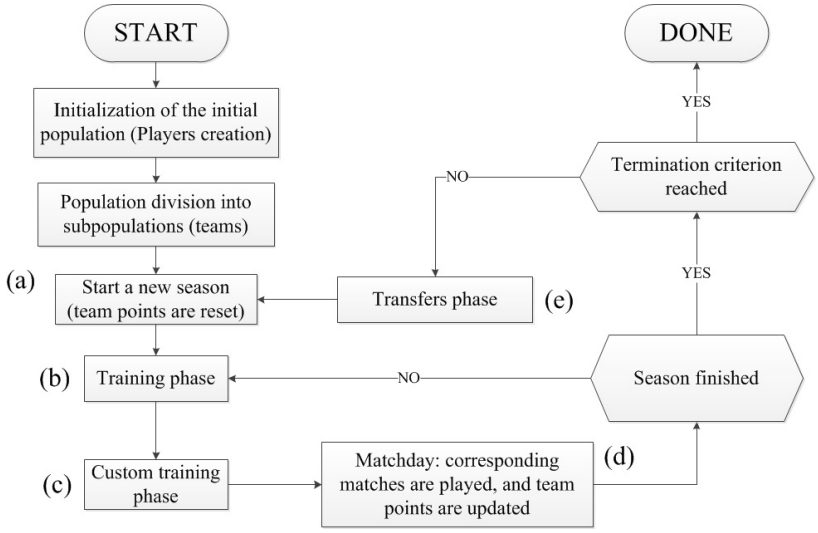

Fig. 1 Flowchart of the *GB* meta-heuristic

As we said in the introduction of this work, our new meta-heuristic is a multiple population-based algorithm which takes different concepts related to soccer for the search process. First, *GB* starts creating the initial population and dividing the solutions (called players) among the subpopulations (called teams) of the system. Once this initial phase has been completed, the first season begins. A season is divided into weeks, in which the teams train and face each other creating a league competition. When a season ends, transfer phase begins, in which the players and coaches can switch teams. This process is repeated until the termination criterion is met. Figure 1 shows the flowchart of our proposed algorithm, while in Algorithm 1 the execution is briefly schematized. Now, the different steps that form the proposed technique are explained in detail.

Algorithm 1: Pseudocode of the *GB*

```
Initialization of the players initial population (Section 3.2.1);
Division of the players into teams (Section 3.2.1);
repeat
    A season is executed (this step it detailed in Section 3.3);
until termination criterion not reached (Section 3.4);
The execution finishes returning the final solution (Section 3.4);
```

3.2 Initialization phase

In this first phase of the algorithm, the players and teams are created as can be seen in Section 3.2.1. Apart from this, in the initialization phase teams first evaluation is performed in the way shown in Section 3.2.2.

3.2.1 Players and teams creation

The first step is the creation of a set of solutions, called P, which will make up the initial population. All solutions are created randomly and they are called *player* (p_i).

$$P : \{p_1, p_2, p_3, p_4, p_5, \ldots, p_{TN*PT}\}$$

$$Where : TN = \ Total\ number\ of\ teams\ of\ the\ system$$

$$PT = \ Number\ of\ players\ per\ team$$

After generating P, players are randomly divided among the different teams t_i that form the league. This division is done iteratively, obtaining a player p_{ij} from P and inserting into t_i until reach TN. Once this division is done, players are represented by the variable p_{ij}, which means "player j of the team i". The set of teams is called T and it consists of TN teams, being $TN \geq 2$. With all this, as an example, teams would be formed as follows:

$$Team\ t_1 : \{p_{11}, p_{12}, p_{13}, \ldots, p_{1PT}\}$$

$$Team\ t_2 : \{p_{21}, p_{22}, p_{23}, \ldots, p_{2PT}\}$$

$$\ldots$$

$$Team\ t_{TN} : \{p_{TN1}, p_{TN2}, \ldots, p_{TNPT}\}$$

Thus, each player of the solution space is part of a different team, as we can see in Figure 2. The set of teams T is created as the one shown below:

$$T : \{t_1, t_2, t_3, t_4, \ldots, t_{TN}\}$$

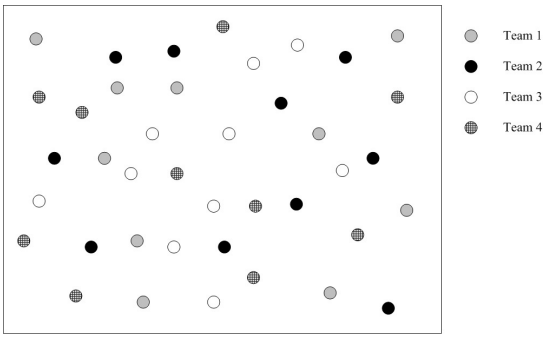

Fig. 2 Space of solutions, team division

3.2.2 Calculation of the teams power

It is logical to think that in the real world, the power or strength of each team depends directly on the quality of the players who make it up, the better the players are, the stronger a team is. This way, if one team is strong, it can win more games and be better positioned in the classification of the league.

The quality of each player p_{ij} is represented by a real number q_{ij}. This number is determined by a cost function $f(p_{ij})$, which depends on the problem. For example, in some routing problems, this function can be directly the total distance traveled. In other cases this function may be more complex and it can take into account details such as distance, the use of vehicles or any penalty for the failure of certain restrictions. Each team has a captain (p_{icap}), which is the player with the best q_{ij} of its team. Stated more formally:

$$p_{icap} = p_{ik} \in t_i \Leftrightarrow \forall j \in \{1, \ldots, PT\} : q_{ik} \geq q_{ij}$$

It should be borne in mind that, as in real life, several p_{ij} of the same team t_i might reach at a certain moment a similar or equal q_{ij}. These equalities are

temporary, because each p_{ij} evolves on its own way with training methods. Thereby, when several players have the same q_{ij} and that q_{ij} is the best of the team, p_{icap} is chosen randomly between these players.

To calculate the strength TQ_i of each team, the meta-heuristic takes into account the quality of all the p_{ij} that comprise that team. TQ_i could be expressed by the following formula.

$$TQ_i = \sum_{j=1}^{PT} q_{ij}/PT$$

3.3 Competition phase

In this central phase of the algorithm, teams train independently and cooperatively, and they improve their power by little steps. Meanwhile, teams face each other creating a league competition that helps to decide the transfer of players between different teams. This process is divided into seasons (S_i). Each season starts at the step (a) depicted in Figure 1, and has two periods of player transfers. A S_i also has as many matches as necessary to complete a conventional league, where all teams face each other twice. For this reason, every season is divided into two parts of equal duration. In these parts, all teams have to face each other team once. Finally, a season has as many training phases as matchdays. This process is shown schematically in Algorithm 2.

Algorithm 2: Pseudocode of a season process

```
Points of each team t_i are reset to 0;
for j = 1, 2 (each season is divided into two equal parts, as in real life) do
    for each matchday (Section 3.3.2) do
        for each t_i in the system do
            Training phase for t_i (Section 3.3.1);
            Custom training session for t_i (Section 3.3.1);
            Calculation of the quality TQ_i of t_i (Section 3.2.2);
        end
        Matchday in which matches are played (Section 3.3.2);
    end
    Period of player transfers (Section 3.3.3);
    Period of termination of coaches (Section 3.3.3);
end
```

3.3.1 Training methods

The training phase (step (b) depicted in Figure 1) is when all the players from each team receive a training session that makes them improve. In real life, each team has its own training method. Some of these training methods are more effective than others. This way, some teams improve more than others. This

fact is reflected in the ranking of the league competition, where the teams that use more effective trainings get a better position, since they have more power than other teams and they can win more matches.

To capture this situation on the proposed technique, each team has a different training method, namely, a successor function that works on a particular neighborhood structure in the solution space. For certain routing problems, a function of this type could be the well-known 2-opt or 3-opt [55]. The training method is assigned randomly at the initialization process. For each training, this function is applied a certain number of times (until its own termination criterion is reached) to improve a p_{ij}. The p'_{ij} generated is accepted only if $q'_{ij} > q_{ij}$. Thereby, each team examines in different manner the neighborhoods of the players it possesses, making the players evolve in a completely different way, depending on the team in which they are. This fact helps to the exploration and exploitation of the solution space, which also are enhanced by the fact that players can switch teams on multiple occasions. Algorithm 3 outlines schematically this process.

Algorithm 3: Pseudocode of a training proces

```
while counter < TerminationCriterion do
    Create a new player (p'_ij) from p_ij using the training function;
    if q'_ij > q_ij (quality of p_ij is improved) then
        p_ij = p'_ij (p_ij is replaced by p'_ij);
        Counter=0;
    else
        Counter++;
    end
end
```

It must be taken into account that the more times the function is applied, the more computational time is needed. In addition, the fact of applying this function more times does not mean a better performance, since the player can reach a local optimum. For this reason, as mentiones above, each training process has its own termination criterion. A training ends when there are a number of successors without improvement in the q_{ij} of the player trained. This number is related to the neighborhood of the team successor function, or training function in this case. For example, taking the 2-opt as training function, a training ends when there are $n + \sum_{k=1}^{n} k$ (the size of its neighborhood) successors without improvement, being n the size of the problem.

This process could make a change in the p_{icap} of a t_i. This fact occurs when a player p_{ij} improves the quality of the captain of its team. Figure 3 shows an example of this exchange.

Another kind of training is that we called *Custom Training*, the step (c) depicted in Figure 1. It may happen that a player p_{ij} is in a period when, despite receiving training, it does not experience any improvement in its q_{ij}.

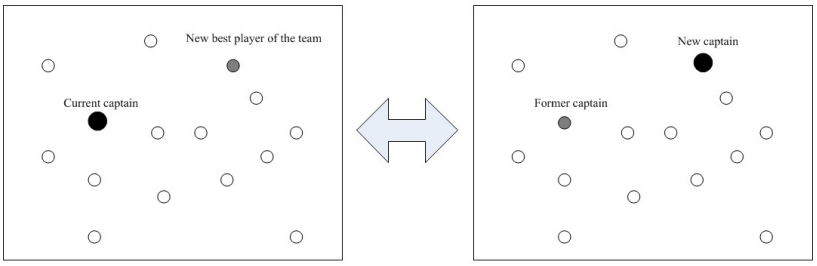

Fig. 3 Example of change of captain

From a sport point of view, this can happen because the player is focused too much on training qualities which cannot improve because he is in a moment of poor physical form. From the viewpoint of optimization, this happens when p_{ij} is in a local optimum. To make the p_{ij} run away of this rut or obstacle, we have created the custom training concept in our meta-heuristic. These trainings are performed by a p_{ij} with the help of the captain of its team. Through these trainings, p_{ij} can escape from a local optimum and it can move in the solution space to another region or point, which may be a promising point.

From a practical standpoint, it is a combination of the characteristics of these two teammates, resulting in a new player who has probably taken a leap into the solution space. This jump can be beneficial to the search process, as it help a thorough exploration of the solution space.

As an example of this kind of training, in which p_{ij} receives one custom training with the help of the captain of its team, we can suppose that the two players are as follows:

$$p_{ij} : [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9]$$

$$p_{icap} : [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9]$$

Where x_i and y_i are the components of the players, that is, the variable values of the solutions represented by p_ij. One possible combination of the features of both players, and therefore a player resulting from a custom training, could be the following:

$$p'_{ij} : [x_0, x_1, y_2, y_3, y_4, y_5, x_6, x_7, x_8, x_9]$$

The player p'_{ij} created by this kind of training replaces p_{ij}. If we take the TSP as example, x_i and y_i would be the different cities that comprise the environment, and a function that combines the characteristics of two players could be the well-known Order Crossover [56] or the Very Greedy Crossover [57].

In the real world, this is a widespread practice, in which players with more quality and experience teach their teammates to improve those qualities that they have yet to improve.

3.3.2 Matches

The matches, as in the real world, are between two teams. Matches are created as needed to complete a league, considering that all teams have to face each other twice in a season. This phase is the step (d) depicted in Figure 1.

Each match consists of PT chances. Each chance materializes in goal through a tournament between a p_{ij} for each team, which are faced by their team position. The player with higher q_{ij} win the chance and it suppose a goal for his team. The team that scores more goals is the winner of the match. As in real life, the team winner of the match obtains 3 points and the loser 0. In case of a tie each team gets 1 point. The points scored by each team are used to perform a classification, sorted by the number of points scored. Algorithm 4 shows the process of a match. The players of both teams are ordered in decreasing order of quality q_{ij}.

Algorithm 4: Process of a match

```
GoalsTeam1=0;
GoalsTeam2=0;
for each player in a team (PT) do
    if q_1i > q_2i (The player of the first team is better) then
        GoalsTeam1++;
        else if q_1i < q_2i (The player of the second team is better) then
            GoalsTeam2++;
        end
    end
end
```

3.3.3 Period of transfers

The period of transfers is a process in which teams exchange players between them. Thereby, all teams try to reinforce with the acquisition of new players. In soccer world this is a process that happens every year. Normally, top teams sign top players, while the other teams have to settle with lower quality players. In every season there are two periods of transfers, the first is in the half of the season, as a winter market, and the other at the end, as a summer market.

This fact has also been implemented in GB. It is placed in the step (e) represented in Figure 1. In this case, the score of each t_i and its position in the classification of the league are decisive factors to decide the type of transfer for each t_i. In the middle and the end of each season, teams that are in the top half of the classification are reinforced with the best p_{ij} of the teams of the bottom half. While the t_i in the lower half have to settle with the acquisition of the less good p_{ij} of top teams.

It has to keep in mind that the better position of the team, the better the player it receives. That is, the best team gets the best player of the worst

team. On the other hand, the second best team receives the second best player of the penultimate team. In addition, if TN is an odd number, the team at the central position of the classification does not exchange any player. Below is an example of this process for a league composed of 4 teams, taking into account that players are ranked in order of their q_{ij} and the t_i depending on their position in the league:

$$Team\ t_1 : \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$$

$$Team\ t_2 : \{b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8\}$$

$$Team\ t_3 : \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$$

$$Team\ t_4 : \{d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8\}$$

After the period of signings, these teams could be composed as follows:

$$Team\ t_1 : \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, \boldsymbol{d_0}\}$$

$$Team\ t_2 : \{b_0, b_1, b_2, b_3, b_4, b_5, b_6, \boldsymbol{c_1}, b_8\}$$

$$Team\ t_3 : \{c_0, \boldsymbol{b_7}, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$$

$$Team\ t_4 : \{\boldsymbol{a_8}, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8\}$$

These interchanges of p_{ij} help the search process, since they allow a different processing of solutions during the execution, avoiding falling easily into local optima and increasing the searching capability of the meta-heuristic. In other words, this process of neighborhood changing improves the exploration capacity of the technique, and especially, contributes to greater exploitation of promising regions of the solution space.

Other type of transfers that we take into account, and also help to the exploration and exploitation, are the called *special exchanges*. In the world of soccer, players change the team not just to go to a better team and win more titles. It is possible that a player has played a long time on a team, and his ambitions have fallen or he thinks that it cannot improve more if he stays in the same team for more time. Therefore, the player can decide to change of team, regardless of whether the target is a worse or better team.

In our meta-heuristic this fact is also raised, although it is applied differently to reality. If a p_{ij} takes a certain number of trainings without improvements in its q_{ij} even receiving custom trainings, it changes from its t_i to another random t_k, with the aim of obtaining new different trainings. In addition, it is no matter if $TQ_k < TQ_i$. To keep the PT per team, there is an exchange with a random p_{ij} of t_k. In addition, this exchange can happen at any time of the season, so, despite having a philosophy that adapts itself to real life, is not completely faithful to the rules imposed by the soccer federation. The number of trainings without improvements that has to happen before the team change is an input variable, which has to be set by the developer to the technique. In the experimentation shown in this paper, this number has

been set to 10. An example of this process might be as follows, assuming the following t_i and t_k:

$$Team\ t_i : \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$$

$$Team\ t_k : \{b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8\}$$

Supposing that a_3 is the player that has not experienced any improvement in its q_{ij}, an exchange is produced with a random player of another random team. In this case this team is t_k. Assuming that the selected player for the exchange is b_5, the teams would be as follows:

$$Team\ t_i : \{a_0, a_1, a_2, \boldsymbol{b_5}, a_4, a_5, a_6, a_7, a_8\}$$

$$Team\ t_k : \{b_0, b_1, b_2, b_3, b_4, \boldsymbol{a_3}, b_6, b_7, b_8\}$$

Moreover, in soccer, not only the players are transferred between teams, usually coaches are also replaced, when their coached teams are not getting the expected results, or when they are getting bad results continuously.

This fact, really common in real life, is also reflected in the proposed method, in a process called *Cessation of coaches*. In each period of transfers, all t_i from the bottom half of the table change their form of training, hoping to get another training method, i.e., another coach, which improves the performance and the TQ_i of the team, seeking for best results. The change of training is done randomly among all the types of training that exist in the system, allowing repetitions between different teams. This neighborhood random change, which affects all players of the team, improves the exploration capacity of the technique.

3.4 Termination criterion

The finalization of the algorithm depends on the fulfillment of three factors. This criterion has to allow the search performed by the algorithm to trace a wide part of the solution space. For this reason, the developed criterion is composed of three termination clauses:

$$\sum_{i=1}^{TN} TQ'_i \leq \sum_{i=1}^{TN} TQ_i \tag{1}$$

$$\sum_{i=1}^{TN} q'_{icap} \leq \sum_{i=1}^{TN} q_{icap} \tag{2}$$

$$BestSolution' \leq BestSolution \tag{3}$$

In other words, the execution of the GB finishes when the sum of the powers TQ'_i of all the teams does not improve comparing to the previous season (1), the sum of the quality q'_{icap} of all the captains has no improvement over the

previous season (2), and there is no improvement in the best solution found by the whole system ($BestSolution'$) in relation to the previous season (3).

When the finalization criterion is reached, the algorithm returns the p_{ij} with the best q_{ij} of the system, which is the solution that our technique gives to the problem.

3.5 Contribution of the *GB* technique

After describing in detail the *GB*, in this section the original contribution that our technique brings is introduced. We compare the *GB* with the meta-heuristics presented in Section 2, which are some of the most widely used today.

Regarding the PABC, the main differences compared to *GB* can be listed as follows:

1. In PABC there are three types of individuals in the population, each with different behavior. In *GB* all individuals behave the same way.
2. The *GB* presents a system of exchange of players between subpopulations based on the quality of each subpopulation. To the best of our knowledge, this system has never been given in the literature for any multi-population ABC.
3. In the PABC, bees share information between them. In *GB*, players not only share information between them, if necessary, they also share their own characteristics with the custom training.

In relation to the differences between *GB* and PPSO, the two following differences can be highlighted:

1. In PPSO, each particle performs its movement based on its current position, his best known position and the best position found in the entire swarm. In *GB*, all players make their moves autonomously, without the need of obtaining information from other individuals (Except for custom training, which is a combination between individuals and it is rarely performed).
2. In parallel versions of PSO, each subpopulation executes its movements in the same way, based on the parameter of velocity. In *GB*, each subpopulation evolves differently, each of them with a different neighborhood function. Moreover, players can change their way of evolving depending of the training function of the team.

Regarding PGAs, taking into account the philosophy of the three types of PGA, we can say that the Island Model or Distributed GAs (DGA) are the most similar to our *GB*. With all this, the main differences between DGA and *GB* are the following.

1. Both algorithms have two operators, the first one is applied to single individuals (mutation and training), and the second one is a cooperative

operator (crossover and custom training). As can be read in [47], the vast majority of the DGAs give more importance to the second operator, leaving the first in a secondary plane. In *GB*, by contrast, the local improvement of individuals receives more importance, using the cooperation between them on a few occasions, as a resource for exploration.
2. Regarding the cooperative operators and their way of working, in DGAs, the typical selection function of the uni-population GAs is used to choose the individuals that participate in a crossover. In *GB*, however, custom trainings are performed rarely, and between an individual probably trapped in a local optimum, and the better individual of its subpopulation.
3. As can be read in the literature [58, 47], in the DGAs, the migration system (or exchange of individuals between subpopulations) is a very complex issue, and there are a great variety of architectures and topologies. In *GB*, on the other hand, there is a well-defined exchange strategy with a dynamic topology, based on the quality of each one of the subpopulations.
4. In DGAs, subpopulations cooperate, while in *GB* they compete.

Talking about the ICA, the philosophy and way of working of ICA is very different compared to *GB*. Main differences may be listed as follows:

1. In ICA, all colonies make their movements based on the position of the imperialist state of their empire, while the latter remains motionless. In *GB*, on the other hand, each individual makes his moves autonomously.
2. In ICA, the number of subpopulations is reduced during the execution up to 1. In *GB*, subpopulations number is maintained throughout the execution.
3. Individuals transfer system is very different in both algorithms. While in *GB* it is an exchange, in ICA is a single transfer.

Finally, as happens with PPSO, the difference between SOA and *GB* are significant, being the most notable the fact that in SOA, each individual makes its movement based on its current or historical positions of themselves or their neighbors (depending on its behavior). In *GB*, on the other hand, each individual has the same behavior and makes its moves autonomously. Another important difference is the system of individual migration between subpopulations. In SOA, these migrations are performed making crossovers between individuals from different populations. In *GB*, complete individuals are exchanged, basing these migrations on the quality of each subpopulation.

3.6 Selecting the proper technique for comparison

As can be seen, our proposal offers several differences with each and every one of the techniques presented above. These techniques are some of the most used today, and we can say that our approach proposes certain originality respect to all of them.

Analyzing the philosophy and way of working of the algorithms described, it can be concluded that the DGAs are the techniques that shares most

similarities with our GB. In the evolution of their individuals, both meta-heuristics rely on two operators, a local and a cooperative one, which are used for the exploitation and exploration. In addition, both techniques are easy to apply to routing problems and easy to parameterize. For this reason, to check the quality of the proposed technique, we use the uni-population GA and DGA in the experimentation phase. All the details of this phase are explained in next section.

4 Experimentation setup

In this section, all the details about the experimentation that has been carried out in this work are introduced. First, in Section 4.1 the details of the 4 algorithms used in the comparison are described, and the two routing problems used are introduced. Then, in Section 4.2, a small study to determine how to adjust the parameters of the GB is conducted. Finally, another small study about the parametrization of one of the techniques used in the experimentation is performed in Section 4.3.

4.1 General description of techniques and problems used in the comparison

As mentioned before, results obtained by the GB are compared with the outcomes obtained by two versions of the basic uni-population GA [23, 24] and two different DGAs [47]. The comparison is performed with these algorithms, since, as we have seen in Section 3.5, are those with more similarities compared to GB. Next, the details of the four algorithms used in the experiment are introduced:

- Classic Genetic Algorithm with conventional parameters (GA_1): Classic GA, with classic structure, conventional operators, and conventional parameters, i.e., a high crossover probability and a low mutation probability. These concepts are the most used in the literature.
- Classic Genetic Algorithm with parameters adjusted to GB (GA_2): In this case, the structure of the algorithm is the classical, but the operators and the parameters are adjusted to be similar to those used in GB. With this, the number of individual movements (mutations and individual trainings) and cooperative movements (crossovers and custom trainings) made are the same. The functions used are similar for both algorithms, so that the function of mutation are one of those used by the GB as training method, while the crossover function is the same as the custom training function. To match this numbers, probabilities of crossover and mutation are adjusted. Our aim is to perform a 100% reliable comparison, wherein the only difference between the algorithms is the structure. Thus, it may be deduced which technique gets better results using the same operators the same number of times.

- Distributed Genetic Algorithm with conventional parameters (DGA_1): A multi-deme genetic algorithm, in which multiple subpopulations, or demes, evolve separately and exchange individuals occasionally. This is the most popular distributed GA. The topology used for this algorithm is the dynamic topology. In this topology, a deme is not restricted to communicate with some fixed set of subpopulations. This way, whenever a deme improves its best solution found, it shares this new best solution with all the other demes. This individual replaces the worst individual of each subpopulation. In every deme, a GA_1 runs in parallel, each of them with different crossover and mutation probabilities.
- Distributed Genetic Algorithm with parameters adjusted to GB (DGA_2): This last technique is a hybrid between DGA_1 and GA_2. The structure of the algorithm is exactly the same as DGA_1, but in each deme a GA_2 is executed. In this case, each deme has a different mutation function. Thus, the similarity with the GB, in terms of operators and parameters, is more faithful.

Tests were performed with two different combinatorial optimization problems oriented on the vehicle routing, which are the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP).

TSP [3], is one of the most famous and widely studied problems throughout history in operations research and computer science. It has a great scientific interest and today is used in a large number of studies [59–61]. This problem can be defined on a complete graph $G = (V, A)$ where $V = \{v_0, v_1, \ldots, v_n\}$ is the set of vertices which represents the clients of the system, and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the set of arcs which represents the interconnection between clients. Each arc has an associated distance cost d_{ij} which is in a known distance matrix C. The TSP objective is to find a route that visits each and every customer once and that minimizes the total distance traveled.

The second problem to be studied is the Capacitated Vehicle Routing Problem [62]. Like the TSP, this problem is also used in a large number of studies nowadays [6,63,64]. The problem can be defined on a complete graph $G = (V, A)$ where $V = \{v_0, v_1, \ldots, v_n\}$ is the set of vertices and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the set of arcs. The vertex v_0 represents the depot and the rest of vertices are the customers, each of them with a demand q_i. A fleet of vehicles is available with a limited capacity Q_k for each vehicle. The CVRP objective is to find a minimum number of routes of minimum cost such that *i)* each route starts and ends at the depot, *ii)* each client is visited exactly by one route and *iii)* the total demand of the customers visited by one route does not exceed the total capacity of the vehicle that performs it [65].

4.2 Setting Golden Ball parameters

Before starting the tests, a small study about the parametrization of the Golden Ball is shown in this section. It should be taken into account that performing a comprehensive study on the parameterization of the algorithm

would be very extensive. That study has been planned as future work, since the goal of this paper is the presentation of the technique, and the demonstration that it is a good alternative to solve routing problems. For this reason, in this paper we show a small portion of that study, in order to justify the parameter settings used, related to the number of teams and players per team. For that, we compare 4 different versions of GB, applied to the TSP. In this four versions, the initial population has 48 players, and the only difference between them is the distribution of players. This distribution is as follows:

- Version 1: 2 teams and 24 players per team.
- Version 2: 4 teams and 12 players per team.
- Version 3: 6 teams and 8 players per team.
- Version 4: 8 teams and 6 players per team.

The training functions used for the 4 versions are the same as has been used for the tests presented in the following section. These functions are described in Section 5.1. Results obtained by the four versions can be seen in Table 1. Those tests were performed on an Intel Core i5 2410 laptop, with 2.30 GHz and a RAM of 4 GB. Since this is a comparison between different GB versions, for each run we display only the average of the results and the average of the execution time (in seconds). The number of executions for each instance is 20. Instances were obtained from the TSP Benchmark TSPLIB [66]. The name of each instance has a number that displays the number of nodes it has.

Instance	Version 1		Version 2		Version 3		Version 4	
Name	Avg.	Time	Avg.	Time	Avg.	Time	Avg.	Time
Oliver30	420.30	0.12	**420.00**	0.18	**420.00**	0.32	**420.00**	0.47
Eilon50	431.30	0.46	**427.00**	0.85	427.40	1.23	427.40	1.64
Eil51	433.30	0.54	428.50	0.91	429.20	1.43	**427.90**	1.76
Berlin52	7599.80	0.75	**7542.00**	1.25	**7542.00**	2.18	**7542.00**	2.84
St70	685.85	1.55	679.45	2.09	678.20	4.08	**678.00**	5.75
Eilon75	553.55	1.63	544.35	3.37	**541.50**	5.23	541.70	6.53
Eil76	553.10	2.09	545.30	3.85	545.60	5.52	**544.45**	6.99
KroA100	21549.60	5.54	21386.70	8.12	**21318.70**	13.98	21325.70	17.76
KroB100	22729.20	4.58	22311.05	7.51	22337.50	11.95	**22284.7**0	21.55
KroC100	21055.30	5.76	20968.25	8.05	20846.60	16.84	**20840.25**	18.14
KroD100	21602.00	6.44	21485.80	7.75	**21481.60**	13.93	21510.60	17.48
KroE100	22385.00	6.78	22266.80	7.95	22211.90	16.79	**22157.80**	19.08
Eil101	648.70	6.08	643.70	7.97	641.20	12.76	**640.55**	19.68
Pr107	45049.00	6.46	44693.00	9.45	**44492.90**	16.18	44561.85	24.26
Pr124	59664.10	10.41	59348.20	14.67	59402.10	17.45	**59288.10**	32.25
Pr136	99215.45	18.33	98906.50	21.25	98356.85	29.53	**98155.80**	62.36
Pr144	59120.40	19.92	58712.00	25.76	58698.50	50.63	**58656.55**	69.54
Pr152	74952.20	21.56	74320.70	28.45	74275.35	55.30	**74190.00**	64.25

Table 1 Results of the four GB versions. For each instance, the results average and the execution time average are shown.

Some conclusions can be drawn if the data presented in Table 1 are analyzed. We can see a slight trend of improvement in the results when the number of teams increases. This improvement could be explained simply. The greater the number of teams, the greater the exploration and exploitation

capacity of the algorithm is. This is so, because the number of training sessions and the number of interactions between teams increases (exchanging more players, for instance).

Even so, this fact involves a significant increase of runtime, which is not directly proportional to the improvement in results. In this paper, to achieve the goals we have proposed, we select the option which best balances the runtime and results quality. This option is the version 2, with 4 teams and 12 players per team. This version has acceptable execution times, and is the alternative that more improvement offers regarding its previous version (version 1). Versions 3 and 4, on the other hand, need very high execution times in relation to the version 2, without offering significant improvements in the results quality.

4.3 Setting GA_1 parameters

In order to make a fair and rigorous comparison, another small study about the parametrization of the GA_1 is performed in this section. As in Section 4.2, four different GA_1 versions are compared. The initial population has also 48 individuals, and the only difference between each version is the crossover and mutation probabilities (p_c and p_m). The characteristics of each technique are as follows:

- GA_1^{v1}: $p_c = 95\%$ and $p_m = 5\%$.
- GA_1^{v2}: $p_c = 90\%$ and $p_m = 10\%$.
- GA_1^{v3}: $p_c = 80\%$ and $p_m = 20\%$.
- GA_1^{v4}: $p_c = 75\%$ and $p_m = 25\%$.

Crossover and mutation functions used in these tests are the same as in the following section, and they are described in Section 5.1. Results obtained by the different version of the GA_1 are shown in Table 2. These tests follow the same directions as those presented in the previous section.

If the results obtained in this small study are analized, it can be concluded that the differences between the four versions are not very large in many instances. Anyway, it can be seen how GA_1^{v3} gets better results in 55.56% of the instances (10 out of 18), while GA_1^{v1}, GA_1^{v2} and GA_1^{v4} obtain better outcomes only in 5.55% (1 out of 18), 11.11% (2 out of 18) and 27.78% (5 out of 18) of the cases, respectively. On the other hand, regarding execution times, despite an improving trend can be seen according the probability of crossover descends, the differences are not too wide. For these reasons, and taking into account the quality of the results, version 3 ($p_c = 80\%$, $p_m = 20\%$) has been chosen for the experimentation of this work.

5 Tests with Traveling Salesman Problem (TSP)

In this section, the details of the tests performed with the TSP are detailed. First, in Section 5.1, characteristics of the algorithms used are introduced.

Instance	GA_1^{v1}		GA_1^{v2}		GA_1^{v3}		GA_1^{v4}	
Name	Avg.	Time	Avg.	Time	Avg.	Time	Avg.	Time
Oliver30	427.65	0.35	426.05	0.40	**425.95**	0.29	427.20	0.25
Eilon50	458.50	1.86	453.05	1.75	**451.55**	1.31	452.25	1.27
Eil51	459.45	2.24	453.70	2.12	**453.65**	1.85	453.80	1.59
Berlin52	7969.20	2.34	7949.30	1.88	7945.65	1.42	**7896.45**	1.31
St70	743.35	5.54	720.55	5.75	**711.85**	5.21	712.65	4.85
Eilon75	593.95	6.12	601.60	6.61	582.05	5.86	**579.10**	5.23
Eil76	615.80	6.44	593.40	6.24	**582.85**	6.06	584.60	5.34
KroA100	22842.95	15.12	**22536.80**	14.52	22559.30	12.48	22591.30	11.54
KroB100	24096.45	16.47	23717.70	15.75	**23342.40**	12.45	23378.45	12.15
KroC100	22374.25	15.46	22239.65	14.28	**22010.30**	12.82	22148.15	12.13
KroD100	22781.35	13.93	22778.40	14.31	22642.25	12.21	**22469.80**	11.85
KroE100	23413.95	13.70	23433.15	13.81	**23228.35**	11.13	23332.65	10.84
Eil101	721.00	19.85	700.75	18.42	696.00	17.29	**685.05**	17.16
Pr107	47827.70	18.19	47118.65	18.01	47356.15	16.84	**46956.45**	15.72
Pr124	61271.70	28.75	**60832.30**	26.45	60871.80	24.52	61062.25	24.15
Pr136	**102637.85**	42.53	103810.80	41.78	102819.10	38.43	103040.65	36.95
Pr144	64064.20	58.75	62563.30	56.42	**60715.40**	53.61	61209.00	53.15
Pr152	78429.90	75.87	78071.20	72.48	**76819.05**	68.15	77581.60	67.58

Table 2 Results of the four GA_1 versions. For each instance, the results average and the execution time average are shown.

Then, in Section 5.2 results obtained by the techniques are shown. Finally, these result are analyzed in Section 5.3.

5.1 Characteristics

Now, parameters and characteristics of the algorithms used to address the TSP are detailed. First, these are the characteristics of the *GB*:

- *Number of teams (NT):* 4
- *Number of players per team (PT):* 12
- *Termination criteria for the training sessions:* A session finishes when there are a number of successors proportional to the neighborhood of the training function without improvements
- *Number of trainings without improvement for a player to make a change of team:* 10
- *Number of trainings without improvement for a custom training:* 5

The characteristics of GA_1 are as follows:

- *Size of the population:* 48
- *Number of generations:* The execution of the algorithm terminates when there are a number of generations proportional to the neighborhood of the mutation function without improvements.
- *Crossover probability:* 80%
- *Mutation probability:* 20%
- *Selection function and Survivor function:* 100 % Elitist

On the other hand, the characteristics of DGA_1 are as follow:

- *Number of subpopulations:* 4
- *Number of individuals per subpopulation:* 12
- *Number of generations:* The execution of the algorithm terminates when there are a number of generations proportional to the neighborhood of one randomly chosen mutation function without improvements in the best solution found by all the demes.
- *Crossover probability:* 95%, 90%, 80% and 75%, respectively
- *Mutation probability:* 5%, 10%, 20% and 25%, respectively
- *Selection function and Survivor function:* 100 % Elitist

The characteristics of GA_2 and DGA_2 are the same as GA_1 and DGA_1, respectively, changing only the probabilities of crossover and mutation to 0.003% and 100%, to fit with the parameters of GB.

The objective function for this problem is the total distance traveled, so that the objective is to minimize it. In both techniques, solutions are encoded using the Path Representation [67]. The successor functions used as training functions are as follows:

- *Vertex Insertion:* This function selects randomly a node of the route, removes it and reinserts in another random position. This operator is often used for any kind of problem, because it is easy to implement and it obtains very good results [68,69].
- *Swapping:* In this case two nodes are selected randomly to swap their positions. It is also a very used operator [70,71].
- *2opt:* It was defined by Lin in 1965 [55] and, as in other cases, this operator has been used a lot of times for any kind of problem [72–74]. This function eliminates at random two arcs within the existing path and creates two new arcs, avoiding the generation of sub tours.
- *3opt:* The operation way of 3opt, proposed also by Lin, is similar to 2opt, with the difference that in this case the arcs removed are 3. The complexity of using this operator is greater than the 2opt. Therefore, the 3opt has not enjoyed the popularity of the previous. Despite this, the operator has been used a large number of times throughout the history [75,76].

These functions are used as training functions in the GB algorithm and as mutation functions for DGA_2. For the GA_1, GA_2 and DGA_1 the *2opt* is used as mutation operator, since it is the one that gets best results. The custom training operator for the GB is as follow:

- *Golden Help Funtion:* In this operator take part two players, the player to coach and the captain of the team. The result of this operator is a new player that combines the characteristics of both players. Assuming a 10-node instance, an example of the 2 players could be next:

$$p_{icap} : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$$

$$p_{ij} : [4, 2, 6, 5, 0, 1, 9, 7, 8, 3]$$

First, the first half of the nodes (in this case, customers, or cities) composing the captain are selected and inserted in the new player.

$$p'_{ij} : [0, 1, 2, 3, 4]$$

Finally, the remaining nodes are inserted in the same order in the final solution.

$$p'_{ij} : [0, 1, 2, 3, 4, 6, 5, 9, 7, 8]$$

This function is also used as crossover operator in GA_2 and DGA_2. In GA_1 and DGA_1, the well-known Order Crossover (OX) is used [56].

Finally, Table 3 summarizes the characteristics of the 4 algorithms with which we compare our proposal.

Alg.	Population	p_c and p_m	Cross. operators	Mut. operators
GA_1	1 population, 48 individuals	80% & 20%	OX	2-opt
GA_2	1 population, 48 individuals	0.003% & 100%	Golden Help Function	2-opt
DGA_1	4 subpopulations, each with 12 individuals	Respectively: 95% & 5%, 90% & 10%, 75% & 25%, 80% & 20%	OX	2-opt (the same for all subpopulations)
DGA_2	4 subpopulations, each with 12 individuals	0.003% & 100%	Golden Help Function	2-opt, 3-opt, Swapping & Vertex Insertion (a different function for each population)

Table 3 summary of the characteristics of GA_1, GA_2, DGA_1 and DGA_2 for the TSP

5.2 Results

Table 4 shows the results obtained by the GB and GA. Those tests were performed on the same computer described in Section 4.2. In this case, for each instance we display the total average ($\bar{x}$) and, in brackets, the standard deviation. We also show the best result obtained ($\dot{x}$) and the average of the execution time (T), in seconds. The number of executions for each test is 20. We have used the same 20 seeds to generate random initial solutions, thus, the final result depends only on the evolution of the technique. The instances are the same used in Section 4.2, and the number inside brackets beside the name of each instance is its optimum value.

In order to determine if GB average is significantly different than the averages obtained by other techniques, we have performed Students t-test. The t statistic has the following form:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{(n_1-1)SD_1^2+(n_2-1)SD_2^2}{n_1+n_2-2}\frac{n_1+n_2}{n_1 n_2}}}$$

Instance	Golden Ball		GA_1		GA_2		DGA_1		DGA_2	
Oliver30 (420)	**420.0**	(±0.0)	425.9	(±7.1)	426.0	(±9.9)	428.0	(±4.8)	424.1	(±6.8)
$\dot{x}$ & T	420	0.2	420	0.3	420	0.2	420	0.3	420	0.2
Eilon50 (425)	**427.0**	(±1.5)	451.5	(±15.1)	442.5	(±6.5)	442.1	(±7.4)	435.8	(±5.8)
$\dot{x}$ & T	425	1.1	435	1.3	432	1.5	433	0.9	429	0.8
Eil51 (426)	**428.6**	(±1.3)	453.6	(±10.5)	444.8	(±9.8)	442.5	(±7.9)	438.4	(±4.8)
$\dot{x}$ & T	427	1.4	439	1.8	434	1.8	432	1.1	430	1.3
Berlin52 (7542)	**7542.0**	(±0.0)	7945.6	(±168.6)	7841.3	(±256.7)	7914.3	(±186.9)	7866.0	(±296.4)
$\dot{x}$ & T	7542	2.1	7542	1.4	7542	1.8	7542	1.2	7542	1.9
St70 (675)	**679.4**	(±3.5)	711.8	(±33.6)	716.4	(±44.3)	719.6	(±18.8)	699.0	(±11.9)
$\dot{x}$ & T	675	4.2	682	5.2	684	4.2	705	3.8	683	3.9
Eilon75 (535)	**544.3**	(±3.3)	582.0	(±14.3)	565.6	(±12.0)	570.4	(±10.6)	557.1	(±8.9)
$\dot{x}$ & T	536	5.4	570	5.8	550	5.5	556	5.1	544	4.5
Eil76 (538)	**545.3**	(±3.7)	582.8	(±15.0)	569.7	(±11.5)	574.8	(±15.1)	563.5	(±6.4)
$\dot{x}$ & T	539	5.5	560	6.0	545	5.7	556	5.1	552	5.1
Eil101 (629)	**643.7**	(±4.3)	696.0	(±16.8)	676.6	(±11.2)	678.7	(±13.7)	665.6	(±10.3)
$\dot{x}$ & T	636	8.9	676	17.2	657	10.7	657	12.9	643	8.4
KroA100 (21282)	**21386.7**	(±99.7)	22559.3	(±538.4)	21838.8	(±419.1)	22757.1	(±433.6)	21940.6	(±313.1)
$\dot{x}$ & T	21282	9.5	21679	12.4	21376	9.3	22206	13.7	21478	10.4
KroB100 (22140)	**22311.0**	(±139.6)	23342.4	(±468.6)	22896.3	(±424.8)	23323.1	(±375.8)	22815.3	(±329.7)
$\dot{x}$ & T	22140	9.7	22574	12.4	22178	10.5	22763	13.5	22264	10.9
KroC100 (20749)	**20968.2**	(±111.3)	22010.3	(±607.1)	21536.1	(±396.1)	22311.5	(±582.6)	21472.3	(±321.9)
$\dot{x}$ & T	20769	9.3	21348	12.8	20880	9.8	21454	13.6	21039	9.5
KroD100 (21294)	**21485.8**	(±188.2)	22642.2	(±543.2)	22205.6	(±401.6)	22719.0	(±616.2)	22065.0	(±406.2)
$\dot{x}$ & T	21294	9.7	21696	12.2	21495	9.9	22013	12.7	21459	10.8
KroE100 (22068)	**22266.8**	(±158.1)	23228.3	(±416.4)	22752.7	(±304.3)	23062.8	(±443.9)	22819.0	(±312.2)
$\dot{x}$ & T	22068	9.8	22418	11.1	22147	9.5	22299	13.5	22819	10.4
Pr107 (44303)	**44693.0**	(±210.7)	47356.1	(±1210.1)	45614.4	(±1389.4)	46533.1	(±1507.9)	45506.8	(±1323.4)
$\dot{x}$ & T	44391	10.1	45512	16.8	44387	10.6	44872	16.1	44438	12.4
Pr124 (59030)	**59348.2**	(±190.3)	60871.8	(±694.0)	59943.7	(±544.7)	61149.0	(±888.2)	60193.8	(±569.4)
$\dot{x}$ & T	59030	16.2	59953	24.5	59030	17.1	59490	30.2	59076	14.3
Pr136 (96772)	**98906.5**	(±1296.2)	102819.1	(±1929.7)	100610.5	(±1230.7)	102585.2	(±3241.8)	100949.0	(±1706.6)
$\dot{x}$ & T	97439	23.5	99468	38.4	98137	24.1	98973	40.9	98797	26.2
Pr144 (58537)	**58712.0**	(±247.7)	60715.4	(±1753.4)	60662.3	(±2330.9)	61447.3	(±1763.7)	59470.5	(±641.1)
$\dot{x}$ & T	58537	34.1	58922	53.6	58599	32.5	59143	58.3	58538	33.5
Pr152 (73682)	**74320.7**	(±390.3)	76819.0	(±2038.8)	75699.1	(±912.0)	76563.5	(±904.8)	75663.9	(±1253.3)
$\dot{x}$ & T	73818	36.7	74268	68.1	74526	37.5	74613	80.0	74249	35.8

Table 4 Results of GB, GA_1, GA_2, DGA_1 and DGA_2 for the TSP. For each instance results average, standard deviation, best result and time average are shown.

where:

$\overline{X_1}$: Average of GB
SD_1: Standard deviation of GB,
$\overline{X_2}$: Average of the other technique,
SD_2: Standard deviation of the other technique,
n_1: GB size,
n_2: Size of the other technique,

In Table 5, we show a direct comparison between GB and each of the other techniques, GA_1, GA_2, DGA_1 and DGA_2, using the Student's t-test. This comparison is made for each of the instances used in Table 4. The t values shown can be positive, neutral, or negative. The positive value of t indicates that our proposal is significantly better than the technique with which it is facing. In the opposite case, GB obtains significant worse solutions. If t is neutral, the difference between the two algorithms is not significant. We stated confidence interval at the 95% confidence level ($t_{0.05} = 1.96$).

Instance	GB vs. GA_1	GB vs. GA_2	GB vs. DGA_1	GB vs. DGA_2
Oliver30	+ (3.71)	+ (2.71)	+ (7.45)	+ (2.69)
Eilon50	+ (7.22)	+ (10.39)	+ (8.94)	+ (6.56)
Eil51	+ (10.56)	+ (7.32)	+ (7.76)	+ (8.81)
Berlin52	+ (10.70)	+ (5.21)	+ (8.90)	+ (4.88)
St70	+ (4.28)	+ (3.72)	+ (9.40)	+ (7.12)
Eilon75	+ (11.48)	+ (7.61)	+ (10.51)	+ (6.03)
Eil76	+ (10.85)	+ (9.03)	+ (8.48)	+ (11.01)
Eil101	+ (13.48)	+ (12.26)	+ (10.90)	+ (8.77)
KroA100	+ (9.57)	+ (4.68)	+ (13.77)	+ (7.53)
KroB100	+ (9.43)	+ (5.85)	+ (11.29)	+ (6.29)
KroC100	+ (7.55)	+ (6.17)	+ (10.47)	+ (6.61)
KroD100	+ (8.99)	+ (7.25)	+ (8.55)	+ (5.78)
KroE100	+ (9.65)	+ (6.33)	+ (7.55)	+ (7.05)
Pr107	+ (9.69)	+ (2.93)	+ (5.40)	+ (2.71)
Pr124	+ (9.46)	+ (4.61)	+ (8.86)	+ (6.29)
Pr136	+ (7.52)	+ (4.26)	+ (4.71)	+ (4.26)
Pr144	+ (5.05)	+ (3.72)	+ (6.86)	+ (4.93)
Pr152	+ (5.38)	+ (6.21)	+ (10.17)	+ (4.57)

Table 5 Students t-test for TSP. '+' indicates that GB is significantly better (at 95% confidence level).

5.3 Analysis of the results

We can conclude from the Table 4 that the proposed technique gets better results for each and every one of the instances in terms of averages and best solution. Furthermore, thanks to Students test presented in Table 5 we can say that these improvements are significant. The only point in which the GB has been outperformed is in the best result for the instance Pr107, where GA_2 has obtained a better solution (44391 by GB, compared to 44387 by GA_2). Even so, is less of an issue because the average is lower in quality than the average of GB.

The fact of obtaining better results by GB may occur for several reasons. GB is a technique that combines local search with the cooperation and competition between the various players in the system. While our technique gives greater importance to the improvement of players individually, the other algorithms focus more in the cooperation among individuals of the population and subpopulations. Even so, the GB also has mechanisms for cooperation between players, with the custom training. This resource is used only when it is beneficial to the search process and exploration of the solution space. The custom trainings help to avoid local optimums and to explore the solution space more exhaustively. Another advantage over GA_1, GA_2 and DGA_1 is the option that a player can explore different neighborhood structures. This occurs because players can switch teams and receive different trainings methods throughout the execution of the algorithm. This mechanism is another way to avoid local optima and helps players to explore in different ways the solution space. In addition, players can delve into those parts of space which are most interesting to the search process. GA_1, GA_2, DGA_1 and DGA_2, on the other hand, have several mechanisms to avoid local optima, but optimization mechanisms are not as powerful as the GB.

Talking about the execution time, GA_1 and DGA_1 require more execution time than GB, while GA_2 and DGA_2 need similar times to GB. This fact gives an advantage to GB, since in times similar to GA_2 and DGA_2, it can obtain better results than the rest of meta-heuristics.

The reason why our algorithm needs lower runtime than GA_1 and DGA_1 is logical. If individual operators (mutation and individual training) and cooperative operators (crossover and custom training) are compared, the last ones needs more time to execute, since they operate with two different solutions, and their working way is more complex. On the other hand, mutations and individual trainings operate with one solution and they make a simple modification in a solution which can be made in a minimum time. GB makes less cooperative movements than the GA_1 and DGA_1, and this fact is perfectly reflected in the runtimes. Furthermore, GB, GA_2 and DGA_2 obtain similar runtimes because they use their operators similarly.

Another important feature that is obligatory to highlight is the robustness of the GB. The standard deviation of the results obtained by the GB is lower than the deviation of other algorithms. This means that for the GB, the differences between the worst and the best results of every instance are not very large. This characteristic gives robustness and reliability to the algorithm, something that is very important if we want to use our technique in a real environment.

To conclude the analysis of the results, and with the aim of making a deeper analysis, we compare the convergence behavior of the GB and the DGA_2. We have selected the DGA_2 for this comparison because it is the most similar to GB, both in concept and in results average. In Table 6, the average number of objective function evaluations needed to reach the final solution for each instance is shown (in thousands).

Instance	GB	DGA_2
Oliver30	**10.72**	17.06
Eilon50	52.86	**49.74**
Eil51	**51.46**	54.27
Berlin52	**53.04**	54.00
St70	127.62	**104.61**
Eilon75	138.40	**128.78**
Eil76	144.30	**137.77**
Eil101	312.89	**311.03**
KroA100	**232.86**	281.11
KroB100	285.29	**242.91**
KroC100	**277.70**	308.12
KroD100	**199.56**	248.74
KroE100	294.94	**287.33**
Pr107	**338.33**	362.50
Pr124	**408.16**	464.77
Pr136	**616.52**	706.00
Pr144	**771.01**	867.56
Pr152	1195.98	**996.85**

Table 6 Convergence of GB and DGA_2 for TSP, expressed in thousand of objetive function evaluations

The Table 6 shows that both algorithms have similar behavior related to convergence, being slightly better for the GB. This is an advantage for GB, since making a similar number of objective function evaluations can obtain better results.

As a conclusion we can say that using the same functions and the same parameters, our meta-heuristic is more efficient than GA_1, GA_2, DGA_1 and DGA_2 solving the TSP. Now, let's test whether this fact is true also for the next problem.

6 Tests with Capacitated Vehicle Routing Problem (CVRP)

In this section, the details of the tests performed with the CVRP are introduced. First, in Section 6.1, characteristics of the algorithms used are mentioned. Then, results obtained by the techniques are shown in Section 6.2. Finally, in Section 6.3, results are analyzed.

6.1 Characteristics

The parameters and characteristics of the algorithms used to address this problem are the same that we used to solve the TSP, except the number of teams, which increases to 6. This fact makes the system players total amount to 72, therefore, the size of the population of the GA_1 and GA_2 also ascends to 72. Furthermore, for DGA_1 and DGA_2, the number of subpopulations increases to 6. This increase is due to the complexity of the problem, which requires more treatment and further exploration of the solution space.

The objective function for this problem is also the total distance traveled, so that the objective is to minimize it. Solutions are encoded using an adaptation of the Path Representation. In this case, the routes are also represented as a permutation of nodes. To distinguish the routes of one solution, they are separated by zeros. For example, if we have a solution with three routes, for instance [2-4-6], [1-3-7] and [5-8-9], the solution will be coded as follows:

$$[\mathbf{0}, 2, 4, 6, \mathbf{0}, 1, 3, 7, \mathbf{0}, 5, 8, 9, \mathbf{0}]$$

This form of encoding has been widely used in the literature for the VRP and its variants [77–79].

Two successor functions used as training functions are the same that we use for the TSP. These functions are the *2opt* and the *Vertex Insertion*. In this case, both are intra-route functions, which mean that they work only within a specific route of a solution. Remaining functions are, as Savelsbergh called [80], inter-route functions, and they work as follows:

- *Swapping Routes:* In this case two nodes are selected at random from two random routes to swap their positions. Within this function the creation of new paths may happen by selecting a node from a random route and creating a new one with the selected node.

– *Vertex Insertion Routes:* In this case a node is selected randomly from a random route and re-inserted in a random position of another randomly selected route. The creation of new routes is also possible in this function with the selected node.

These functions are used as training functions in the GB algorithm, and as mutation functions in DGA_2. For GA_1, GA_2 and DGA_1, *Vertex Insertion Routes* is used as mutation operator, since it is the one that gets best results. To get to this statement, several tests have been performed with the four mutation functions described above. Anyway, it has been considered that the objectives of the paper is to introduce the proposed technique and to demonstrate that it is a good alternative to solve combinatorial optimization problems. For this reason, and because of the large extension of this work, these tests have not been shown in the paper.

The custom training function for the GB, which is also used as crossover operator for the GA_1, GA_2, DGA_1 and DGA_2, is as follow:

– *Golden Help Funtion:* In this operator take part two players, the player to coach and the captain of the team. Assuming a 17-node instance (including the depot), an example of the 2 players could be as follows:

$$p_{icap} : [\mathbf{0}, 1, 2, 3, 4, \mathbf{0}, 5, 6, 7, 8, \mathbf{0}, 9, 10, 11, 12, \mathbf{0}, 13, 14, 15, 16, \mathbf{0}]$$

$$p_{ij} : [\mathbf{0}, 1, 11, 6, 8, \mathbf{0}, 2, 4, 14, 10, \mathbf{0}, 3, 16, 15, 12, \mathbf{0}, 5, 9, 13, 7, \mathbf{0}]$$

First, the 50% of the best routes of the captain are selected and inserted in the new player.

$$p'_{ij} : [\mathbf{0}, 1, 2, 3, 4, \mathbf{0}, 5, 6, 7, 8, \mathbf{0}]$$

Finally, the remaining nodes are inserted in the same order in the new player, creating new routes. Thus, taking into account the capacity of the vehicles, it is probably the creation of more than one route. Continuing the example, the resulting player could be the next:

$$p'_{ij} : [\mathbf{0}, 1, 2, 3, 4, \mathbf{0}, 5, 6, 7, 8, \mathbf{0}, 11, 14, 10, 16, \mathbf{0}, 15, 12, 9, 13, \mathbf{0}]$$

As we have said in Section 3.3.1, in GB, the player p'_{ij} resulting from this operation replaces p_{ij}.

Table 7 summarizes the characteristics of the 4 algorithms with which we compare our proposal.

Alg.	Population	p_c and p_m	Cross. operators	Mut. operators
GA_1	1 population, 72 individuals	80% & 20%	Golden Help Function	Vertex Insertion Routes
GA_2	1 population, 72 individuals	0.003% & 100%	Golden Help Function	Vertex Insertion Routes
DGA_1	6 subpopulations, each with 12 individuals	Respectively: 95% & 5%, 90% & 10%, 85% & 15%, 80% & 20%, 75% & 25%, 70% & 30%	Golden Help Function	Vertex Insertion Routes (the same for all subpopulations)
DGA_2	6 subpopulations, each with 12 individuals	0.003% & 100%	Golden Help Function	2-opt, Swapping Routes, Vertex Insertion Routes & Vertex Insertion (a different function for each population)

Table 7 summary of the characteristics of GA_1, GA_2, DGA_1 and DGA_2 for the CVRP

6.2 Results

Table 8 shows the results obtained. As in the TSP, for each instance we display the total average ($\bar{x}$) and, in brackets, the standard deviation. We also show the best result ($\dot{x}$) and the average of the execution time (T), in seconds. The number of executions for each instance is 20. As in TSP, we have used the same 20 seeds to generate random initial solutions. The instances were obtained from the CVRP set of Christofides and Eilon, picked up at [81]. The name of each instance has a number that displays the number of nodes it has, and the number inside brackets beside the name of each instance is its optimum value. In Table 9, the Students t-test is shown. The test has been performed in the same way as for the TSP.

Instance	Golden Ball		GA_1		GA_2		DGA_1		DGA_2	
En22k4 (375)	**376.0**	(±2.2)	390.3	(±13.4)	410.0	(±14.4)	401.0	(±15.3)	393.5	(±15.0)
$\dot{x}$ & T	375	0.9	375	2.4	390	1.2	375	3.8	375	1.8
En23k3 (569)	589.7	(±17.7)	625.1	(±31.5)	660.0	(±27.8)	655.1	(±23.1)	**587.6**	(±23.5)
$\dot{x}$ & T	569	0.7	569	3.8	602	1.1	601	3.1	569	1.8
En30k3 (503)	**517.4**	(±15.6)	574.3	(±28.4)	597.4	(±49.3)	560.4	(±49.5)	577.7	(±71.5)
$\dot{x}$ & T	503	2.2	521	5.1	529	2.8	503	4.1	503	2.2
En33k4 (835)	**857.8**	(±9.6)	917.0	(±35.1)	947.7	(±26.5)	921.8	(±27.1)	919.8	(±23.4)
$\dot{x}$ & T	844	2.8	862	7.6	902	2.8	888	6.1	862	2.3
En51k5 (521)	**578.1**	(±10.9)	681.6	(±51.4)	677.9	(±81.7)	635.4	(±31.6)	624.4	(±43.7)
$\dot{x}$ & T	561	9.8	574	17.6	589	10.1	572	23.1	568	7.6
En76k7 (682)	**755.8**	(±13.1)	852.5	(±47.5)	849.9	(±53.5)	819.7	(±31.6)	799.9	(±43.7)
$\dot{x}$ & T	736	25.5	755	59.6	753	24.8	722	67.0	750	23.4
En76k8 (735)	**816.9**	(±14.8)	923.2	(±37.0)	908.9	(±38.4)	883.3	(±53.5)	873.9	(±43.7)
$\dot{x}$ & T	795	31.5	851	64.6	859	32.5	801	69.2	808	28.5
En76k10 (830)	**913.6**	(±15.6)	1002.8	(±32.4)	995.2	(±58.7)	962.6	(±41.5)	959.0	(±50.6)
$\dot{x}$ & T	888	37.8	932	65.4	928	37.6	906	55.2	888	27.2
En76k14 (1021)	**1124.6**	(±11.5)	1198.8	(±20.0)	1186.7	(±47.9)	1177.3	(±52.8)	1172.2	(±37.1)
$\dot{x}$ & T	1107	28.8	1142	60.1	1117	32.7	1104	46.4	1110	29.4
En101k8 (815)	**906.4**	(±16.4)	1104.4	(±44.8)	999.9	(±46.0)	971.7	(±69.1)	991.1	(±41.1)
$\dot{x}$ & T	867	69.8	1042	124.5	908	67.5	893	134.2	933	67.5
En101k14 (1071)	**1191.5**	(±26.1)	1298.0	(±112.4)	1288.8	(±52.8)	1249.9	(±56.5)	1273.5	(±50.7)
$\dot{x}$ & T	1155	77.9	1175	119.4	1187	78.5	1182	134.4	1194	75.4

Table 8 Results of GB, GA_1, GA_2, DGA_1 and DGA_2 for the CVRP. For each instance results average, standard deviation, best result and time average are shown.

Instance	GB vs. GA_1	GB vs. GA_2	GB vs. DGA_1	GB vs. DGA_2
En22k4	+ (4.70)	+ (10.43)	+ (7.23)	+ (5.16)
En23k3	+ (4.38)	+ (9.53)	+ (10.03)	∗ (-1.03)
En30k3	+ (7.85)	+ (6.91)	+ (3.70)	+ (3.68)
En33k4	+ (7.27)	+ (14.26)	+ (9.95)	+ (10.96)
En51k5	+ (8.80)	+ (5.41)	+ (7.66)	+ (4.59)
En76k7	+ (8.77)	+ (7.64)	+ (8.35)	+ (4.32)
En76k8	+ (11.92)	+ (9.99)	+ (5.34)	+ (5.52)
En76k10	+ (11.09)	+ (6.00)	+ (4.94)	+ (3.83)
En76k14	+ (14.38)	+ (5.63)	+ (4.36)	+ (5.48)
En101k8	+ (18.56)	+ (8.56)	+ (4.11)	+ (8.55)
En101k14	+ (4.12)	+ (7.38)	+ (4.19)	+ (6.43)

Table 9 Students t-test for CVRP. '+' indicates that GB is better. '*' indicates that the difference between the two algorithms is not significant (at 95% confidence level)

6.3 Analysis of the results

As in the TSP, the proposed method obtains significantly better results in almost all CVRP instances compared with other techniques. The only case in which GB has been overcome is in En23k3 instance, by DGA_2. Anyway, this improvement over GB is not significant, as can be seen in the Students t-test. Regarding execution times, GB needs less time than GA_1 and DGA_1. Compared to GA_2, the runtimes of both algorithms are similar. Finally, the parametrization and operators used for DGA_2 and GB are the same, which is why their runtimes are also similar. Still, comparing with DGA_2, the GB is a more complex technique, and it has a greater capacity of exploitation and exploration. For this reason, GB performs a more exhaustive search of the solution space, and its runtimes are slightly higher compared with DGA_2. Apart from this, the standard deviation is lower in the GB technique than in the other algorithms. Therefore, solutions are more regular for the GB also in this problem. This feature gives robustness and reliability to the algorithm.

Following the same steps as for the above problem, in Table 10 can be seen the convergence behavior of GB and DGA_2 for the CVRP.

Instance	GB	DGA_2
En22k4	**29.56**	33.16
En23k3	**16.65**	24.04
En30k3	**57.51**	72.39
En33k4	**40.99**	63.09
En51k5	**122.76**	138.45
En76k7	**227.84**	307.19
En76k8	**242.37**	284.69
En76k10	**216.19**	262.75
En76k14	232.66	**227.74**
En101k8	**595.84**	796.99
En101k14	**502.24**	591.75

Table 10 Convergence of GB and DGA_2 for CVRP, expressed in thousand of objetive function evaluations

Unlike the above problem, for the CVRP, the GB shows better convergence behavior than DGA_2 for almost all the instances. This is an advantage for GB, because it demonstrates a better ability of exploitation than the DGA_2, requiring fewer number of objective function evaluations to obtain better results

In conclusion we can say that the GB proves to be better than GA_1, GA_2, DGA_1 and DGA_2 also for CVRP. It is better in terms of quality, convergence behavior and robustness. Regarding runtime, it is similar compared with GA_2 and DGA_2 and better than GA_1 and DGA_1. The reasons why our technique is better are the same that we have mentioned in Section 5.3.

7 Conclusions

In this paper we have presented a new multi-population meta-heuristic for solving combinatorial optimization problems. In this meta-heuristic, various solutions that make up the population are called *players*, which are grouped into different *teams* that together form a *league*. Teams face each other playing soccer matches that serve to guide the process of players exchange between teams and change of coaches. A training phase is used for the improvement of the players and it is different for each team, so that players receive different trainings depending on the team they are in.

In this work, we have tested the quality of our new technique, showing the results that it has obtained with two problems, and comparing them with the results obtained by the basic GA and a Distributed GA. First, we have described in detail the different steps of the GB. Then, we have introduced the problems to solve, which are two well-known problems in the field of combinatorial optimization, the TSP and the CVRP, and finally, we have shown the results of the tests. We have shown that our meta-heuristic is a good alternative to solve these problems, since it improves in every way the GA and DGA.

As future work, we can mention the intention of applying this new meta-heuristic to a real environment, making a more elaborate objective function and creating more complex constraints. Now, the proposed meta-heuristic is used in the PRODIS project (Grant PI2011-58, funded by the Basque Government in Spain). This project combines in one system, the industrial production with the distribution of materials. The GB is used in this work as an algorithm to solve a distribution problem, which is defined as a Capacitated Vehicle Routing Problem with Time Windows and Backhauls. Also, at this time, we are planning the application of GB to a dynamic distribution system of car windscreen repairs. In this case the problem is defined as a dynamic CVRP, wherein the routes may be re-planned according to the needs of customers.

In addition, we are planning to compare our technique with any commercial solver. These techniques are not similar to the GB in terms of concepts

and philosophy. Anyway, we think that the comparisons on results could be interesting.

ACKNOWLEDGMENT

This work is an extension of the two-page late-breaking abstract presented in the fifteenth annual conference on genetic and evolutionary computation (GECCO)[82]. In that short abstract we introduce a preliminary version of our technique in a very concise way.

References

1. Papadimitriou, C.: The new faces of combinatorial optimization. Combinatorial Optimization (2012) 19–23
2. Korte, B., Vygen, J.: Combinatorial optimization: theory and algorithms. Volume 21. Springer-Verlag (2012)
3. Lawler, E., Lenstra, J., Kan, A., Shmoys, D.: The traveling salesman problem: a guided tour of combinatorial optimization. Volume 3. Wiley New York (1985)
4. Coffman, E.G., Bruno, J.L.: Computer and job-shop scheduling theory. John Wiley & Sons (1976)
5. Lenstra, J., Kan, A.: Complexity of vehicle routing and scheduling problems. Networks **11**(2) (1981) 221–227
6. Mattos Ribeiro, G., Laporte, G.: An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research **39**(3) (2012) 728–735
7. Xu, Y., Qu, R.: A hybrid scatter search meta-heuristic for delay-constrained multicast routing problems. Applied Intelligence **36**(1) (2012) 229–241
8. Onieva, E., Naranjo, J., Milanes, V., Alonso, J., Garcia, R., Perez, J.: Automatic lateral control for unmanned vehicles via genetic algorithms. Applied Soft Computing Journal **11**(1) (2011) 1303–1309
9. Zheng, Y.J., Chen, S.Y.: Cooperative particle swarm optimization for multiobjective transportation planning. Applied Intelligence **39**(1) (2013) 202–216
10. Kang, M.H., Choi, H.R., Kim, H.S., Park, B.J.: Development of a maritime transportation planning support system for car carriers based on genetic algorithm. Applied Intelligence **36**(3) (2012) 585–604
11. Masoud, H., Jalili, S., Hasheminejad, S.M.H.: Dynamic clustering using combinatorial particle swarm optimization. Applied Intelligence **38**(3) (2013) 289–314
12. Shin, K.S., Jeong, Y.S., Jeong, M.K.: A two-leveled symbiotic evolutionary algorithm for clustering problems. Applied Intelligence **36**(4) (2012) 788–799
13. Harman, M., McMinn, P., de Souza, J.T., Yoo, S.: Search based software engineering: Techniques, taxonomy, tutorial. In: Empirical Software Engineering and Verification. Volume 7007. Springer (2012) 1–59
14. Gao, J., Sun, L., Gen, M.: A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Computers & Operations Research **35**(9) (2008) 2892–2907
15. Wang, L., Zhou, G., Xu, Y., Wang, S., Liu, M.: An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology **60**(1) (2012) 303–315
16. Zhang, R., Wu, C.: Bottleneck machine identification method based on constraint transformation for job shop scheduling with genetic algorithm. Information Sciences **188**(1) (2012) 236–252
17. Wang, K., Zheng, Y.J.: A new particle swarm optimization algorithm for fuzzy optimization of armored vehicle scheme design. Applied Intelligence **37**(4) (2012) 520–526

18. Rahmati, S.H.A., Zandieh, M., Yazdani, M.: Developing two multi-objective evolutionary algorithms for the multi-objective flexible job shop scheduling problem. The International Journal of Advanced Manufacturing Technology **64**(5-8) (2013) 915–932
19. Kirkpatrick, S., Gellat, C., Vecchi, M.: Optimization by simmulated annealing. science **220**(4598) (1983) 671–680
20. Torres-Jimenez, J., Rodriguez-Tello, E.: New bounds for binary covering arrays using simulated annealing. Information Sciences **185**(1) (2012) 137–152
21. Glover, F.: Tabu search, part i. ORSA Journal on computing **1**(3) (1989) 190–206
22. Hedar, A.R., Ali, A.F.: Tabu search with multi-level neighborhood structures for high dimensional problems. Applied Intelligence **37**(2) (2012) 189–206
23. Goldberg, D.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Professional (1989)
24. De Jong, K.: Analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan, Michigan, USA (1975)
25. Shi, K., Li, L.: High performance genetic algorithm based text clustering using parts of speech and outlier elimination. Applied Intelligence **38**(4) (2013) 511–519
26. Dorigo, M., Blum, C.: Ant colony optimization theory: A survey. Theoretical computer science **344**(2) (2005) 243–278
27. Wu, J., Abbas-Turki, A., El Moudni, A.: Cooperative driving: an ant colony system for autonomous intersection management. Applied Intelligence **37**(2) (2012) 207–222
28. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Techn. Rep. TR06, Erciyes Univ. Press, Erciyes (2005)
29. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: artificial bee colony (abc) algorithm and applications. Artificial Intelligence Review (2012) 1–37
30. Tsai, P.W., Pan, J.S., Liao, B.Y., Chu, S.C.: Enhanced artificial bee colony optimization. International Journal of Innovative Computing, Information and Control **5**(12) (2009) 5081–5092
31. El-Abd, M.: A cooperative approach to the artificial bee colony algorithm. In: IEEE Congress on Evolutionary Computation. (2010) 1–5
32. Banharnsakun, A., Achalakul, T., Sirinaovakul, B.: Artificial bee colony algorithm on distributed environments. In: IEEE Second World Congress on Nature and Biologically Inspired Computing. (2010) 13–18
33. Parpinelli, R.S., Benitez, C.M.V., Lopes, H.S.: Parallel approaches for the artificial bee colony algorithm. In: Handbook of Swarm Intelligence. Springer (2010) 329–345
34. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the IEEE Sixth International Symposium on Micro Machine and Human Science. (1995) 39–43
35. Langdon, W., Poli, R.: Evolving problems to learn about particle swarm optimizers and other search algorithms. IEEE Transactions on Evolutionary Computation **11**(5) (2007) 561–578
36. Hasanzadeh, M., Meybodi, M.R., Ebadzadeh, M.M.: Adaptive cooperative particle swarm optimizer. Applied Intelligence **39**(2) (2013) 397–420
37. Angeline, P.J.: Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In: Evolutionary Programming VII, Springer (1998) 601–610
38. Xu, Y., Wang, Q., Hu, J.: An improved discrete particle swarm optimization based on cooperative swarms. In: IEEE International Conference on Web Intelligence and Intelligent Agent Technology. Volume 2. (2008) 79–82
39. Niu, B., Zhu, Y., He, X., Wu, H.: Mcpso: A multi-swarm cooperative particle swarm optimizer. Applied Mathematics and Computation **185**(2) (2007) 1050–1062
40. Chanj, J., Chu, S.C., Roddick, J.F., Pan, J.S.: A parallel particle swarm optimization algorithm with communication strategies. Journal of Information Science and Engineering **21**(4) (2005) 809–818
41. Manderick, B., Spiessens, P.: Fine-grained parallel genetic algorithms. In: Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers Inc. (1989) 428–433

42. Reeves, C.R.: Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc. (1993)
43. Whitley, D., Rana, S., Heckendorn, R.B.: The island model genetic algorithm: On separability, population size and convergence. Journal of Computing and Information Technology **7** (1999) 33–48
44. Li, C., Yang, S.: An island based hybrid evolutionary algorithm for optimization. In: Simulated Evolution and Learning. Springer (2008) 180–189
45. Calégari, P., Guidec, F., Kuonen, P., Kobler, D.: Parallel island-based genetic algorithm for radio network design. Journal of Parallel and Distributed Computing **47**(1) (1997) 86–90
46. Abbasian, R., Mouhoub, M.: A hierarchical parallel genetic approach for the graph coloring problem. Applied Intelligence **39**(3) (2013) 510–528
47. Cantú-Paz, E.: A survey of parallel genetic algorithms. Calculateurs paralleles, reseaux et systems repartis **10**(2) (1998) 141–171
48. Atashpaz-Gargari, E., Lucas, C.: Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: IEEE Congress on Evolutionary Computation, IEEE (2007) 4661–4667
49. Wang, G.j., Zhang, Y.B., Chen, J.W.: A novel algorithm to solve the vehicle routing problem with time windows: Imperialist competitive algorithm. Advances in Information Sciences and Service Sciences (2011)
50. Yousefikhoshbakht, M., Sedighpour, M.: New imperialist competitive algorithm to solve the travelling salesman problem. International Journal of Computer Mathematics **3**(5) (2013) 108–116
51. Dai, C., Chen, W., Zhu, Y.: Seeker optimization algorithm. In: International Conference on Computational Intelligence and Security, Springer-Verlang (2006) 225–229
52. Dai, C., Chen, W., Song, Y., Zhu, Y.: Seeker optimization algorithm: a novel stochastic search algorithm for global numerical optimization. Journal of Systems Engineering and Electronics **21**(2) (2010) 300–311
53. Dai, C., Chen, W., Zhu, Y., Zhang, X.: Seeker optimization algorithm for optimal reactive power dispatch. IEEE Transactions on Power Systems **24**(3) (2009) 1218–1231
54. Dai, C., Chen, W., Zhu, Y.: Seeker optimization algorithm for digital iir filter design. IEEE Transactions on Industrial Electronics **57**(5) (2010) 1710–1718
55. Lin, S.: Computer solutions of the traveling salesman problem. Bell System Technical Journal **44**(10) (1965) 2245–2269
56. Davis, L.: Applying adaptive algorithms to epistatic domains. In: Proceedings of the international joint conference on artificial intelligence. Volume 1. (1985) 161–163
57. Julstrom, B.A.: Very greedy crossover in a genetic algorithm for the traveling salesman problem. In: Proceedings of the ACM symposium on Applied computing. (1995) 324–328
58. Ochi, L.S., Vianna, D.S., Drummond, L., Victor, A.: A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. Future Generation Computer Systems **14**(5) (1998) 285–292
59. Liefooghe, A., Humeau, J., Mesmoudi, S., Jourdan, L., Talbi, E.: On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics **18**(2) (2012) 317–352
60. Casazza, M., Ceselli, A., Nunkesser, M.: Efficient algorithms for the double traveling salesman problem with multiple stacks. Computers & operations research **39**(5) (2012) 1044–1053
61. Ray, S.S., Bandyopadhyay, S., Pal, S.K.: Genetic operators for combinatorial optimization in tsp and microarray gene ordering. Applied intelligence **26**(3) (2007) 183–195
62. Laporte, G.: The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research **59**(3) (1992) 345–358
63. Ngueveu, S., Prins, C., Wolfler Calvo, R.: An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Computers & Operations Research **37**(11) (2010) 1877–1885
64. Lee, C.Y., Lee, Z.J., Lin, S.W., Ying, K.C.: An enhanced ant colony optimization (eaco) applied to capacitated vehicle routing problem. Applied Intelligence **32**(1) (2010) 88–95

65. Cordeau, J., Maischberger, M.: A parallel iterated tabu search heuristic for vehicle routing problems. Computers & Operations Research **39**(9) (2012) 2033–2050
66. Reinelt, G.: Tspliba traveling salesman problem library. ORSA journal on computing **3**(4) (1991) 376–384
67. Larranaga, P., Kuijpers, C.M.H., Murga, R.H., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review **13**(2) (1999) 129–170
68. Cordeau, J., Laporte, G.: A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological **37**(6) (2003) 579–594
69. Breedam, A.: Comparing descent heuristics and metaheuristics for the vehicle routing problem. Computers & Operations Research **28**(4) (2001) 289–315
70. Tarantilis, C.: Solving the vehicle routing problem with adaptive memory programming methodology. Computers & Operations Research **32**(9) (2005) 2309–2327
71. Tang, H., Miller-Hooks, E.: A tabu search heuristic for the team orienteering problem. Computers & Operations Research **32**(6) (2005) 1379–1407
72. Tarantilis, C., Kiranoudis, C.: A flexible adaptive memory-based algorithm for real-life transportation operations: Two case studies from dairy and construction sector. European Journal of Operational Research **179**(3) (2007) 806–822
73. Bianchessi, N., Righini, G.: Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. Computers & Operations Research **34**(2) (2007) 578–594
74. Osaba, E., Onieva, E., Carballedo, R., Diaz, F., Perallos, A.: An adaptive multi-crossover population algorithm for solving routing problems. In: Nature Inspired Cooperative Strategies for Optimization. Springer (2014) 113–124
75. Alfa, A., Heragu, S., Chen, M.: A 3-opt based simulated annealing algorithm for vehicle routing problems. Computers & Industrial Engineering **21**(1) (1991) 635–639
76. Rocki, K., Suda, R.: Accelerating 2-opt and 3-opt local search using gpu in the travelling salesman problem. In: IEEE International Conference on High Performance Computing and Simulation. (2012) 489–495
77. Toth, P., Vigo, D.: The vehicle routing problem. Volume 9. Society for Industrial and Applied Mathematics (1987)
78. Lee, Z.J.: A hybrid approach for vehicle routing problem with time windows. Advances in Intelligent Transportation Systems **1**(1) (2012) 11–18
79. Osaba, E., Onieva, E., Carballedo, R., Diaz, F., Perallos, A., Zhang, X.: A multi-crossover and adaptive island based population algorithm for solving routing problems. Journal of Zhejiang University SCIENCE C **14**(11) (2013) 815–821
80. Savelsbergh, M.: The vehicle routing problem with time windows: Minimizing route duration. ORSA Journal on Computing **4**(2) (1992) 146–154
81. Diaz, B.: Vrp web. http://neo.lcc.uma.es/radi-aeb/Web-VRP (2012)
82. Osaba, E., Diaz, F., Onieva, E.: A novel meta-heuristic based on soccer concepts to solve routing problems. In: Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion, ACM (2013) 1743–1744