International Journal of Computational Intelligence Systems, Vol. 5, No. 4 (August, 2012), 745-761

Evolutionary Swarm based algorithms to minimise the link cost in Communication Networks

Eugénia Moreira Bernardino, Anabela Moreira Bernardino
Research Center for Informatics and Communications, Department of Computer Science, School of Technology and Management, Polytechnic Institute of Leiria
Leiria, 2411-901, Portugal
E-mail: {eugenia.bernardino;anabela.bernardino}@ipleiria.pt
www.ipleiria.pt

Juan Manuel Sánchez-Pérez, Juan Antonio Gómez-Pulido, Miguel Ángel Vega-Rodríguez
Department of Technologies of Computers and Communications, Polytechnic School, University of Extremadura
Cáceres, 10071, Spain
E-mail: {sanperez;jangomez;mavega}unex.es
www.unex.es

Received 6 June 2011; accepted 6 June 2012

Abstract

In the last decades, nature-inspired algorithms have been widely used to solve complex combinatorial optimisation problems. Among them, Evolutionary Algorithms (EAs) and Swarm Intelligence (SI) algorithms have been extensively employed as search and optimisation tools in various problem domains. Evolutionary and Swarm Intelligent algorithms are Artificial Intelligence (AI) techniques, inspired by natural evolution and adaptation. This paper presents two new nature-inspired algorithms, which use concepts of EAs and SI. The combination of EAs and SI algorithms can unify the fast speed of EAs to find global solutions and the good precision of SI algorithms to find good solutions using the feedback information. The proposed algorithms are applied to a complex NP-hard optimisation problem - the Terminal Assignment Problem (TAP). The objective is to minimise the link cost to form a network. The proposed algorithms are compared with several EAs and SI algorithms from literature. We show that the proposed algorithms are suitable for solving very large scaled problems in short computational times.

Keywords: Evolutionary Algorithms, Swarm Intelligence, Terminal Assignment Problem, Genetic algorithm with a new swarm mutation operator, Queen-bee Evolutionary Algorithm.

1. Introduction

A great number of engineering models and algorithms have been used to solve complex optimisation problems. The organisms and natural systems, which are working and developing in nature, are interesting and valuable sources for designing and inventing new systems and algorithms to be applied to different fields of science and technology. Among them, EAs and SI algorithms have been extensively applied to solve complex optimisation problems. EAs are a subset of evolutionary computation. They are bio-inspired population-based meta-heuristic optimisation algorithms [1]. SI algorithms are also bio-inspired techniques involving the study of collective behaviour in decentralised systems [2]. Ant Colony Optimisation (ACO), Particle Swarm Optimisation (PSO), Bees Algorithm (BA) and Artificial Bee Colony (ABC) algorithm are some of the most known SI approaches. These algorithms can be used in real-world optimisation problems.

In this paper, we propose two bio-inspired algorithms, which combine characteristics of EAs and SI algorithms. We propose a Genetic Algorithm with a new "Swarm" mutation operator (GAS) and a Queen-Bee Evolutionary Algorithm (QBEA) to optimise a communication network problem - the Terminal Assignment Problem (TAP). The algorithms were tested using small, medium and large TAP instances. The algorithms are used to minimise the link cost to form a network by connecting a given set of terminals to a

given set of concentrators. TAP is a NP-hard combinatorial optimisation problem [3-5].

Several heuristics have been used to optimise TAP: Local Search (LS) methods [6-9], EAs [3-5, 9-18], SI techniques [19, 20], among others. Note however, that this paper presents the first attempt (to the authors' knowledge) to use evolutionary swarm based algorithms to optimise TAP.

We compare the performance of GAS and QBEA with: Genetic Algorithm (GA), Tabu Search (TS) [9], Hybrid GA (HGA) [9], Local Search GA (LSGA) [11], GA with Multiple Operators (GAMO) [12], Hybrid Differential Evolution (HDE) [13], BA [20], Hybrid ACO (HACO) [19], Hybrid Scatter Search (HSS) [16], Discrete Differential Evolution (DDE) [17] and Hybrid Population Based Incremental Learning (HPBIL) [18].

The paper is structured as follows: in Section 2 we describe the TAP; in Section 3 we present the previous work; in Section 4 we describe the proposed algorithms; in Section 5 we discuss the computational results obtained and, in Section 6 we report about the conclusions.

2. TAP

In large centralised networks, concentrators are used to increase the network efficiency: a set of terminals is connected to a concentrator and each concentrator is connected to the central computer.

In TAP the number of concentrators and their capacities and locations are known. Each concentrator is limited in the amount of traffic that it can accommodate. For that reason, each terminal must be assigned to one node of the set of concentrators, in such a way that any concentrator does not overstep its capacity [3-5].

In TAP, a communication network will connect N terminals, each with L_i demand (weight) to M concentrators, each of C_j capacity. Capacities are given by positive integers and each L_i must be small or equal to min (C_j ... C_M). The terminals $CT_i(x, y)$ and concentrators $CP_j(x, y)$ sites have fixed and known locations placed on a Euclidean grid.

Problem Instance (see Fig. 1):

- N Terminals;
- Weights - a vector L, with the capacity required for each terminal;
- Terminals Location - a vector CT, with the location (x, y) of each terminal;
- M Concentrators;
- Capacities - a vector C, with the capacity required for each concentrator;
- Concentrators Location - a vector CP, with the location (x, y) of each concentrator.

The optimisation goals are to simultaneously produce feasible solutions, minimise the distances between concentrators and terminals assigned to them and to maintain a balanced distribution of terminals among concentrators.

In this work, the solutions are represented using integer vectors. We use the terminal-based representation (see Fig. 1). Each position corresponds to a terminal. The value carried by position i of the vector specifies the concentrator to which the terminal i is to be assigned to.

TAP is a NP-hard optimisation problem [3-5] and to deal with its difficulty many researchers proposed in the last decades, several optimisation algorithms to solve TAP (see Section 3). Nowadays, we observe an increasing size and consequently an increasing complexity of communication networks, and for that reason finding an optimal solution for TAP continues to be a hard task.

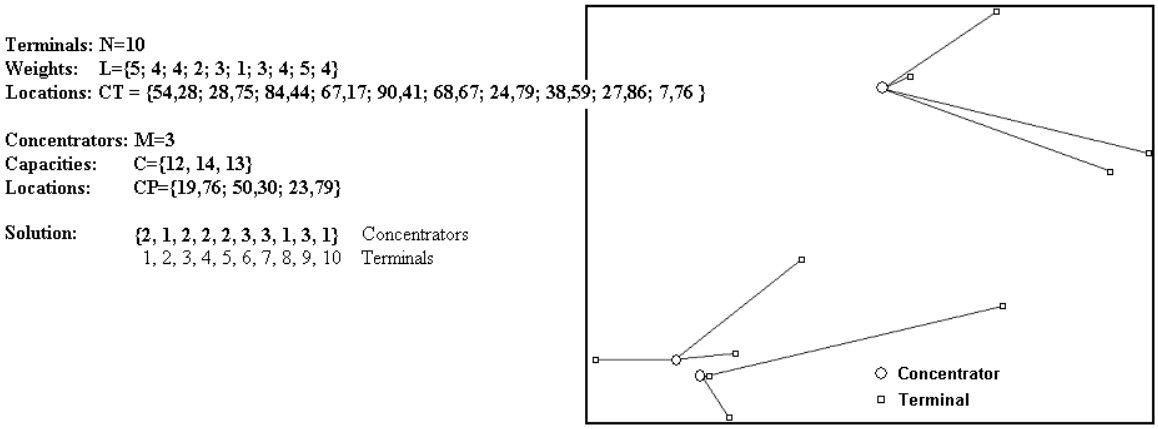

Fig. 1.TAP Example.

3. Previous Work

Many problems in combinatorial optimisation are NP-hard. To solve this type of problems are used approximate methods, because classical heuristics have failed to be efficient. The existing, successful methods in approximate optimisation fall into two classes: Local Search (LS) and population-based search.

There are in literature different approximation algorithms, which use different concepts, derived from: classical heuristics, AI, biological evolution, neural systems, SI and statistical mechanics. These approaches include Simulated Annealing (SA), Tabu Search (TS), Greedy Randomised Adaptive Search Procedure (GRASP), Evolutionary Algorithms (EAs), SI algorithms, their hybrids and others.

Iterated Local Search (ILS) [21] is a simple and powerful stochastic LS method that creates a sequence of solutions generated by an embedded heuristic. ILS is simple, easy to implement, robust, and highly effective. The essential idea of ILS lies in focusing the search on a smaller subspace, defined by locally optimal solutions for a given optimisation engine.

GRASP [22] is a meta-heuristic belonging to the class of LS techniques. It typically consists of iterations made up from successive constructions of a greedy randomised solution and subsequent iterative improvements of it through LS.

SA [23-25] exploits an analogy between the metal annealing process and the search for a minimum value in a more general system. In each run, it attempts to search the entire region of interest for the global minimum rather than performing multiple downhill optimisation runs, in which the selection of the various starting points is automated.

TS [26] is a meta-heuristic algorithm that belongs to the class of LS techniques. TS allows the search of solutions that decrease the objective function value only in those cases where these solutions are not forbidden [27].

Some interesting LS techniques to solve TAP can be found in literature. Atiqullah and Rao [28] proposed SA to find the optimal design of small-scale networks. Pierre et al. [29] proposed SA to find solutions for packet switched networks. Zhang and Ke [30] studied the capability of SA Arithmetic to solve terminal allocation problems in communication networks. Glover et al. [31], and Koh and Lee [32] adopted TS to find an appropriate design of communication networks. Kapantow [6] proposed SA to solve concentrator location and terminal assignment problems. Xu et al. [7] proposed a TS algorithm to solve TAP and compared the results with the ones found with GA and Greedy algorithm. Bernardino [8] proposed ILS and GRASP to find solutions to TAP. Bernardino et al. [9] proposed a TS algorithm and compared the results with the ones found with Hybrid GA.

Low [33] proposed an algorithm to solve Minimum Cost Min-Max Load Terminal Assignment Problem (MCMLTAP) proving that the problem is optimally solvable in polynomial time using MCMLTAP.

EAs use mechanisms inspired by biological evolution [1]. EAs have been successfully used to solve complex combinatorial optimisation problems.

GAs are EAs inspired in the genetic inheritance and the Darwinian strife for survival [34]. Metaphors as chromosomes and population stand for solutions and solution set, respectively. Mechanisms as recombination and mutation give rise to new offspring by manipulating the current population of solutions. Following a standard Darwinian approach, the selection extracts the most promising individuals from the current population [35].

Differential Evolution (DE) [36] is an EA. DE uses the mutation operation as a search mechanism and the selection operation to direct the search toward prospective regions in the search space [37]. Using the members of existing population to build trial vectors, the recombination operator efficiently shuffles information about successful combinations, enabling the search for a better solution space [38, 39].

SS is an useful methodoly to solve combinatorial optimisation problems. It was first introduced in 1977 by Fred Glover [40] and extensive contributions have been made by Manuel Laguna [41]. The SS operates on a small set of solutions and makes only limited use of randomisation as a proxy for diversification when searching for an optimal solution.

DDE was proposed by Pan et al. [42] to solve the permutation flowshop scheduling problem. DDE first mutates a target population to produce a mutant population [42]. Then the target population is recombined with the mutant population in order to generate a trial population. Finally, a selection operator is applied to both target and trial populations to determine who will survive for the next generation.

PBIL algorithm is an EA proposed by Baluja [43]. It uses a stochastic guide search process to obtain new solutions based on the directional information from the previous best solution. PBIL maintains statistics about the search space (learning probabilities) and uses them to direct its exploration [43].

Some interesting evolutionary approaches for TAP can be found in literature. Abuali et al. [10] proposed a Greedy algorithm and a Hybrid Greedy-GA to solve TAP. Khuri and Chui [3] proposed a GA with a penalty function as alternative method to solve TAP and compared the results with the Greedy algorithm. Salcedo-Sanz and Yao [4] proposed two different GAs, using Hopfield Neural Network and compare the results with the GA [3]. Salcedo-Sanz et al. [44] proposed to solve TAP with Groups Encoding. Yao et al. [5] proposed Hybrid GAs and compared the concentrator and terminal-based representations. Bernardino et al. [9] proposed a Hybrid GA (HGA) with a repair procedure and compared the HGA with the TS. Bernardino et al. [12] proposed a GA with multiple operators (GAMO) for crossover and mutation and compared it with the traditional methods. Bernardino et al. [11] proposed a Local Search GA (LSGA) and compared the LSGA with TS, HGA and GAMO. Julstrom [15] proposed three permutation-coded GAs and compared the results between them.

Bernardino et al. [13, 14] proposed HDE and MHDE algorithms to solve TAP and compared these algorithms with the traditional methods. Bernardino et al. [16] proposed an application of a SS algorithm combined with a TS algorithm to solve TAP and compared the results with LSGA, TS, HDE and HACO. Embedded in a DDE algorithm, Bernardino et al. [17] used a LS to improve the TAP solutions quality. The authors compare the performance of DDE with LSGA, TS and MHDE.

A SI algorithm is initialised with a population (i.e., potential solutions), whose individuals are modified over many iteration steps by imitating the social behaviour of insects or animals, in order to find the optimal solution in the problem solution space. A potential solution "flies" through the search space by modifying itself according to its past experience and its relationship with other individuals in the population and the environment [2].

ACO is a SI algorithm. It is based on the indirect communication of a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails [45-49].

BA is also a population-based optimisation method and it has been successfully applied to different optimisation problems. BA is inspired by the food foraging behaviour of honey bees [50] and uses a neighbourhood search method and a LS method to be able to locate the global minimum.

Some interesting SI techniques to solve TAP can be found in literature. Bernardino et al. [19] proposed a Hybrid ACO (HACO) algorithm to solve TAP and compared the results with the traditional methods. Bernardino et al. [20] proposed a BA to solve TAP and compared the results with LSGA, TS, HDE and HACO.

In this paper, we compare the proposed algorithms with the algorithms proposed by Bernardino et al. [8, 9, 11-13, 16-20], because they: (1) use the same 9 small test instances; (2) adopt the same fitness function; (3) implement the algorithms using the same language (C++), and; (4) adopt the same representation (terminal-based).

4. Proposed Algorithms

In order to optimise TAP, we implement two EAs which use crossover, mutation and selection operators and we hybridise them with concepts of SI.

4.1. *GAS*

GA involves a search from a "population" of individuals (potential solutions), in order to find the optimal solution in the problem solution space [35]. Each generation of a GA involves a competitive selection that weeds out poor solutions. Selection is a genetic operator that chooses a solution from the current generation's population for inclusion in the next generation's population. Before making it into the next generation's population, selected solutions may undergo crossover and/or mutation (depending on the probability of crossover and mutation) in which case the offsprings are actually the ones that make it into the next generation's population [35].

The selected solutions are "recombined" with other solutions by swapping parts of a solution with another. Solutions are also "mutated" by making small changes. Recombination and mutation are used to generate new solutions that are biased towards regions of the space for which good solutions have already been seen.

The GA consists in the following steps:

```
Generate initial population
Evaluation
WHILE TerminationCriterion()
        Selection
        Crossover
        Mutation
        Evaluation
```

In [11], Bernardino et al. proposed a Local Search GA (LSGA). It combines global and LS by using a GA to perform exploration, while the LS method performs exploitation.

Combining global and local search is a strategy used by many successful global optimisation approaches, and this type of algorithms has in fact been recognised as a powerful algorithmic paradigm for evolutionary computing [11].

The LSGA consists in the following steps:

```
Generate initial population
Evaluation
WHILE TerminationCriterion()
   Selection
   Crossover
   Mutation
   FOR each solution in population
    Perform local search to get a new solution
   Evaluation
```

We adopted the model proposed by Bernardino et al. [11]. The basic difference is that we created a new mutation operator based on the ACO algorithm.

ACO algorithm was proposed by Dorigo et al. [45, 46], and Dorigo [47]. In real life, ants indirectly communicate among them by depositing pheromone trails on the ground, influencing the decision processes of other ants. This simple communication form among individual ants causes complex behaviours and capabilities on the colony as a whole.

The pheromone trails in ACO serve as distributed numerical information used by the ants in order to probabilistically built solutions to the problem to be solved and which they adapt during the algorithm execution to reflect their search experience.

The standard ACO algorithm uses pheromone trail information to build complete solutions. Gambardella et al. [48] proposed a Hybrid Ant Colony System coupled with a LS (HAS_QAP), applied to the Quadratic Assignment Problem (QAP). HAS-QAP uses pheromone trail information to perform modifications on QAP solutions.

In GAS, the pheromone trail information is also used to perform mutations on TAP solutions.

For solving the TAP, the set of pheromone trails is held in a matrix T of size $N*M$, where each T_{ij} measures the desirability of assigning the terminal i to the concentrator j.

Only the best solution found during the search process contributes to update the pheromone trails. This makes the search process more aggressive and requires less time to reach good solutions [48].

The GAS consists in the following steps:

```
Generate initial population
Evaluation
Initialise pheromone trails
WHILE TerminationCriterion()
   Selection
   Crossover
   Mutation (using pheromone trail matrix or a
   "neighbourhood search" algorithm)
   FOR each solution in population
    Perform local search to get a new solution
   Evaluation
   Pheromone trail updating
```

The next subsections describe each step of the algorithm in detail.

Initialisation of Parameters

The following parameters must be defined by the user: (1) *mi* – number of iterations; (2) *ni* – number of initial solutions; (3) *cr* – crossover operator; (4) *pm* – mutation probability; (5) *pcr* – crossover probability; (6) *nm* – number of modifications (used for mutation); (7) *Q* – value for booting the pheromone trails; (8) *q* – probability of exploration/exploitation; (9) *x1* – pheromone evaporation rate; (10) *x2* – pheromone influence rate, and; (11) *ms* – number of seconds.

Generation of Solutions

The solutions are created using a deterministic form. The deterministic form is based on the Greedy algorithm proposed by Abuali et al. [10]. The Greedy algorithm randomly assigns terminals to the closest feasible concentrators.

Evaluation of Solutions

To evaluate the quality of a potential solution in relation with other potential solutions we use a fitness function. This function returns a number (fitness value) that reflects how optimal the solution is.

A simple and feasible method for optimising TAP is to allocate each terminal to one concentrator without exceeding the capacity of any concentrator. We use a vector of integers:

$$X = X(1), X(2), X(3), ..., X(N) \quad (1)$$

Where $X(t)=c$ means that terminal t is allocated to concentrator c, and $1 \leq X(t) \leq M$; $1 \leq t \leq N$.

In other words, whole terminals must be assigned to one concentrator, and

$$k(c) = \sum_{t=1}^{N} \begin{cases} L(t) & if\ (X(t)=c) \\ 0 \end{cases} \quad (2)$$

$$feasible = \begin{cases} 1 & if\ \ for\ \ all\ \ c;\ 1 \leq c \leq M;\ k(c) \leq C(c) \\ 0 \end{cases} \quad (3)$$

It means that the capacity of one concentrator must not be exceeded by the capacity requirements of the terminals assigned to that concentrator.

The fitness function is the same used in [8, 9, 11-13, 16-20], and it is based on:

(i) the total number of terminals connected to each concentrator (the purpose is to guarantee a balanced distribution of terminals among concentrators);

$$sum_c = \sum_{t=1}^{N} \begin{cases} 1 & if\ (X(t)=c) \\ 0 \end{cases} \quad bal_c = \begin{cases} 10 & if\left(sum_c = round\left(\frac{N}{M}\right)+1\right) \\ 20*abs\left(round\left(\frac{N}{M}\right)+1-sum_c\right) \end{cases} \quad (4)$$

X(t)- concentrator assigned to terminal t

(ii) the distances between concentrators and terminals assigned to them (the goal is to minimise the distances);

$$dist_{t,X(t)} = \sqrt{(CP[X(t)].x - CT[t].x)^2 + (CP[X(t)].y - CT[t].y)^2} \quad (5)$$

(iii) the penalisation if a solution is not feasible (the purpose is to penalise the solutions when the total capacity of one or more concentrators is overloaded).

$$Penalisation = \begin{cases} 0 & if\ (feasible=1) \\ 500 \end{cases} \quad (6)$$

The purpose is to minimise the fitness function (7).

$$fitness = w1 * \sum_{c=1}^{M} bal_c + w2 * \sum_{t=1}^{N} dist_{t,X(t)} + Penalisation \quad (7)$$

This fitness function (7) encourages solutions with a balanced distribution of terminals among concentrators (4). The Euclidean distances between terminals and concentrator are also considered and must be as small as possible (5). The parameters *w1* and *w2 (w1+w2=1)* are used to control the importance of each term (balance/ distances) in the fitness function. In this paper, parameters *w1* and *w2* are set to *0.9* and *0.1*, respectively, which is the same used in [4, 8, 9, 11-13, 16-20]. The fitness function penalises also infeasible solutions (6).

Pheromone trail initialisation

All pheromone trails T_{ij} are set to the same value $T_0=1/(Q*f(S^*))$ [48]. S^* is the best solution found so far and Q a parameter.

Selection

For selection we implement the Tournament operator. This operator randomly selects a subset of individuals (size = *4*). The best individual in this subset is then chosen as the selected individual.

Crossover

For recombination we implement seven crossover operators: one point, 2-points, 4-points, uniform, "reciprocal translocation", "exchange positions" and "exchange terminals of two concentrators".

One point, 2-points, 4-points and uniform are very well-known and widely used in practice.

In "reciprocal translocation", randomly located and arbitrary-length segments are exchanged between parents, as it is illustrated in Fig. 2.

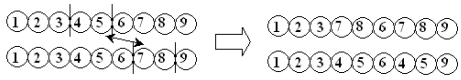

Fig. 2. Reciprocal Translocation.

In "exchange positions", one segment is randomly selected for each parent and the segments are exchanged between parents, as it is illustrated in Fig. 3.

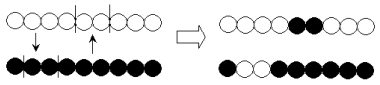

Fig. 3. Exchange Positions.

Mutation

We use a mutation operator, which can perform two types of mutation: (1) "neighbourhood search" mutation and (2) mutation using pheromone trail matrix.

The mutation operator can perform *nm* modifications and it is applied with a probability *pm*. A modification consists on assigning a terminal *t* to a concentrator *c*.

First, a random number *k* is generated (*0* or *1*). If *k* is equal to *0,* the mutation is done using a "neighbourhood search" algorithm.

The "neighbourhood search" mutation consists in the following steps:

```
FOR n=1 TO nm DO
  t = random(N)
  closestC=1
  FOR c=1 TO M DO
    IF distance(t, c) < distance(t, closestC)
      closestC=c
  IF capacityFree(closestC)>=L(t) and
                      mantainBalanced(closestC)
    Assign terminal t to concentrator closestC
  ELSE
    cond=true
    REPEAT
      t1 = random(N)
      t2 = random(N)
      c1 = solution (t1)
      c2 = solution (t2)
      IF (capacityFree(c2)-L(t2)>=L(t1) and
             capacityFree(c1)-L(t1)>=L(t2)) and
           (distance(t2,c1)<=distance(t1,c1) or
            distance(t1,c2)<=distance(t2,c2))
        Assign t1 to c2 and t2 to c1
        cond = false
    WHILE cond=true
```

In the "neighbourhood search" mutation, a mutant individual is obtained by performing multiple moves which length is specified as *nm* (number of modifications).

First, the algorithm chooses a random terminal *t* and searches the closest concentrator. If the concentrator has enough capacity and maintains a balanced distribution of terminals, then the terminal *t* is assigned to the closest concentrator, *closestC*. Otherwise, the algorithm generates two random terminals, *t1* and *t2*. The algorithm verifies the two concentrators, *c1* and *c2*, assigned to them. If the concentrators have enough capacities and at least one of the concentrators is closest to the terminal that will be assigned, then the algorithm exchanges the terminals, *t1* and *t2*, between the two concentrators, *c1* and *c2*. The algorithm repeats this process until terminals, *t1* and *t2*, are interchanged between concentrators, *c1* and *c2*.

If *k* is equal to *1,* the mutation is done using the pheromone trail matrix. A mutant individual is also obtained by performing multiple moves which length is specified as *nm*. First, a terminal *t* is randomly chosen (between *1* and *N*) and after, a concentrator *c* is chosen. A random number *x* is generated between *0* and *1*. If *x* is smaller than *q* (parameter), the best concentrator *c* is selected in such a way that T_{tc} is maximum. This policy consists in exploiting the pheromone trail. If *x* is equal or higher than *q,* the concentrator *c* is chosen with a probability proportional to the values contained in the pheromone trail. This consists in exploring the solution space.

Local Search Algorithm

After recombination and mutation, the solutions go through the improvement phase. A LS algorithm is applied to each solution in the new population to reduce its cost, if possible. We adopt the improved LS algorithm proposed by Bernardino et al. [17]. The LS algorithm applies a partial neighbourhood search. We generate a neighbour by swapping two terminals between two concentrators - *c1* and *c2* (randomly chosen). The algorithm searches for a better solution in the initial set of neighbours. If the best neighbour improves the actual solution, then LS replaces the current solution by the best neighbour. Otherwise, the algorithm creates another set of neighbours. In this case, one neighbour results on assigning one terminal of *c1* to *c2,* or *c2* to *c1*.

The neighbourhood size is *N(c1)*N(c2),* or *N(c1)*N(c2) + N(c1)+N(c2)*.

The LS algorithm consists in the following steps:

```
c1 = random (number of concentrators)
c2 = random (number of concentrators)
NN = neighbours of ACTUAL-SOL (one neighbour
     results of interchange one terminal of c1
     or c2  with one terminal of c2 or c1)
SOLUTION = FindBest (NN)
IF fitness (ACTUAL-SOL) < fitness(SOLUTION)
   NN = neighbours of ACTUAL-SOL (one neighbour
        results of assign one terminal of c1 to
        c2 or c2 to c1)
   SOLUTION = FindBest (NN)
   IF fitness (SOLUTION) < fitness(ACTUAL-SOL)
        ACTUAL-SOL = SOLUTION
ELSE
   ACTUAL-SOL = SOLUTION
```

Pheromone trail updating

To speed-up the convergence, the pheromone trails are updated by taking into account only the best solution found so far [19, 48]. The pheromone trails are updated by setting:

$T_{ij}=(1-x1)*T_{ij}$, where $0<x1<1$ is a parameter that controls the evaporation of the pheromone trail.

$T_{iSi*} = T_{iSi*} + x2/fitness(S^*)$, where $0<x2<1$ is a parameter that controls the influence of the best solution S^* in the pheromone trail.

Termination criterion

The algorithm stops when a maximum number of iterations (*mi*) or a maximum number of seconds (*ms*) is reached.

4.2. *QBEA*

Currently, researchers are studying the behaviour of social insects in an effort to use the SI concepts to create algorithms with the ability to explore the solution search space of the problem in a way similar to the behaviour of social insects [51, 52, 53]. Some algorithms inspired by the behaviour of insects can be called meta-heuristic algorithms, because they provide a high-level framework, which can be adapted to solve optimisation, search, and related problems, as opposed to providing a stringent set of guidelines to solve a particular problem.

A review of literature on algorithms inspired by the behaviour of bees [51] suggests that the topic is evolving and that there is no consensus on a single descriptive title for algorithms based on bees' behaviour. In literature, it is possible to find several bee inspired algorithms that use different algorithm models: Bee System, BeeHive, Virtual Bee algorithm, Bee Swarm Optimisation, Bee Colony Optimisation, Artificial Bee Colony, Bees algorithm and Honey Bees Mating Optimisation algorithm.

Honey-bees live in hives around the world in well organised colonies. These colonies are characterised by the division of labour, where specific bees perform specific tasks [54, 55].

In this paper we consider the Honey-Bees marriage process and the GA model as an inspiration for creating a new bio-inspired algorithm.

QBEA can be considered as a typical swarm-based approach to optimisation, in which the search algorithm is inspired by the process of marriage in real honey bees and it is also inspired by the process of selection and mutation used in GAs. In a real honey bee, the queen crossbreeds with some other bees of the population (selected as fathers) to create new bees for the next generation.

In QBEA, the queen (the best solution in the population) is recombined with some individuals of the population to create new individuals for the next generations. The individuals (fathers) are selected using a selection operator.

QBEA uses two different mutation operators to perform mutations on TAP solutions. Depending on a random number (*0* or *1*), the algorithm can choose the first or second mutation operator.

The first mutation operator only affects a single gene of the individual – simple mutation operator.

The second can affect several genes of the individual – multiple mutation operator.

The QBEA consists in the following steps:

```
Generate initial population
Evaluation
Select queen
WHILE TerminationCriterion()
   Selection (selects the individuals that will
              be recombined with the queen)
   Crossover
   Mutation:
     For each solution in population
        If random(0 or 1) = 0
          Apply simple mutation operator
        Else
          Apply multiple mutation operator
   FOR each solution in population
     Perform local search to get a new solution
   Evaluation
   Update queen
```

The next subsections describe each step of the algorithm in detail.

Initialisation of Parameters

The following parameters, must be defined by the user: (1) *mi* – number of iterations; (2) *ni* – number of initial solutions; (3) *mut* – simple mutation operator; (4) *cr* – crossover operator; (5) *pm1* – mutation probability; (6) *pm2* – mutation probability; (7) *pcr* – crossover probability; (8) *nm* – number of modifications (used in the second mutation operator), and; (9) *ms* – number of seconds..

Generation of Solutions

The solutions are created using the deterministic form proposed by Abuali et al. [10].

Evaluation of Solutions

To evaluate how good a potential solution is in relation to other potential solutions, we use the fitness function (7).

Selection

To create a new population of individuals, the queen is recombined with *ni/2* individuals from the previous population.

We use the Tournament operator to select the individuals that will be recombined with the queen. For each individual that will be selected, this selection operator randomly selects a subset of individuals (size = *4*). The best individual in this subset is then chosen as the selected individual.

Crossover

For recombination we implement the same seven crossover operators used in GAS: one point, 2-points, 4-points, uniform, "reciprocal translocation", "exchange positions" and "exchange terminals of two concentrators".

Mutation

A random number *k* is generated (*0* or *1*). If *k* is equal to *0,* the simple mutation operator is applied with a probability *pm1*. A random number *pr*, between *0* and *1*, is generated. If *pr* is less than *pm1*, then the simple mutation operator is applied to generate a perturbed individual.

We implemented and studied three simple mutation operators: "change order", "change concentrator" and "change closest concentrator". The mutation operator is chosen by the user. In "change order", two terminals are randomly selected and exchanged. In "change concentrator", a terminal is randomly selected and its value (concentrator) is replaced by a new random value (concentrator). In "change closest concentrator", a terminal is randomly selected and its value (concentrator) is replaced by a new value (closest concentrator).

If *k* is equal to *1*, the multiple mutation operator is applied with a probability *pm2*. A uniform random number *pr* is generated between *0* and *1*. If *pr* is less than *pm2,* then the multiple mutation operator is applied to generate the mutant individual. The general mechanism of the multiple mutation operator is equal to the mechanism of the neighbourhood search method used in GAS ("neighbourhood search" mutation – see Section 4.1 - Mutation).

Local Search Algorithm

After recombination and mutation, the solutions go through the improvement phase. We use the same LS algorithm used in GAS to improve the new solutions' quality.

Termination criterion

The algorithm stops when a maximum number of iterations (*mi*) or a maximum number of seconds (*ms*) is reached.

5. Results

In order to test the performance of our approaches, we use a collection of TAP instances of different sizes. We choose 9 small instances from literature [20] and 3 extreme large instances with *1000* terminals and *300* concentrators.

We compare our results with those found previously using GA, TS, HGA, LSGA, GAMO, HDE, BA, HACO, DDE, HSS and HPBIL.

The suggestions from literature helped us to guide our choice of parameter values for TS [9], HGA [9], GAMO [12], HDE [13], LSGA [11], HACO [19], BA [20], HSS [16], DDE [17] and HPBIL [18]. We use the same values proposed by Bernardino et al. [8, 9, 11-13, 16-20] (see Tables 1 and 2).

We performed different tests using different instances of different sizes in order to establish the best parameter values for GAS and QBEA. We select the values presented in Table 1 to produce the results for GAS and QBEA.

Table 3 and Table 4 present the best-obtained results with classical GA, TS, HGA, LSGA, GAMO, HDE, BA, HACO, DDE, HSS and HPBIL. In both tables, the first columns represent the number of the problem (*P*) and the fitness of the Best-Known (*BK*) solution and the remaining columns show the results (*BK*, *Ts* – Run Time) obtained with the mentioned algorithms.

The algorithms have been executed using a processor Intel Core Duo T2300.

The initial solutions were created using the Greedy algorithm. *Ts* (Run Time) corresponds to the execution time that each algorithm needs to obtain the best-known feasible solution.

The values presented in tables 3 and 4 have been computed based on *100* different executions for each test instance.

Table 1. Parameter values.

HACO	Number of ants	30
	Number of iterations for diversification	[N*2 ... N*4]
	Q	100
	Probability exploitation/exploration	0.9
	Pheromone influence	{0.7, 0.8}
	Pheromone evaporation	0.8
	Number of modifications	[3 ... 15]
BA	Number of scout bees	10
	Number of selected sites	10
	Number of best sites	5
	Number of bees sent to selected sites	10
	Number of bees sent to best sites	30
	Number of modifications	6
GAS	Number of individuals	40 (instances 10-12), 100 (instances 1-9)
	Crossover operator	one-point
	Mutation probability	>=0.7
	Crossover probability	<=0.4
	Number of modifications (used for mutation)	<4
	Q	100
	Probability exploitation/exploration	[0.5 … 0.8]
	Pheromone influence	0.8
	Pheromone evaporation	0.8
QBEA	Number of individuals	40 (instances 10-12), 100 (instances 1-9)
	Simple mutation operator	"change concentrator"
	Crossover operator	one-point
	Simple mutation probability	[0.4 … 0.8]
	Multiple mutation probability	>=0.7
	Crossover probability	<=0.4
	Number of modifications (used for mutation)	<4

Tables 5 and 6 present the average fitnesses and standard deviations. The first column represents the number of the problem (*P*) and the remaining columns show the results obtained (*AvgF* – Average Fitness, *Std* – Standard Deviation).

To compute the results in tables 5 and 6 we used *300* iterations/generations for instances 1-4, *500* for the instance 5, *1000* for the instance 6, *1500* for the instance 7 and *2000* for instances 8-9. For instances 10, 11 and 12 we stop the executions when a maximum number of *10000* iterations/generations is reached or when a maximum number of *5000* seconds is reached.

The values presented in tables 5 and 6 have been computed based on *50* different executions (*50* best executions out of *100* executions) for each test instance.

In a first stage, we use a different stop condition for the algorithms. We use a predefined number of evaluations. We observe that EAs present a very poor performance (higher average fitnesses and standard deviations) if we compare them in terms of number of evaluations. This happens because the tested EAs need more population diversity. For this reason, in each iteration, the EAs perform a higher number of evaluations in comparison with SI algorithms. We observe that SI algorithms can obtain very good results using small populations (see Fig. 4). The improvements using large populations are not significant. Large populations are more time consuming and do not provide significant better results using the SI algorithms (see Fig. 4 and Fig. 5).

Table 2. Parameter values.

TS	Number of elements in the tabu list	[5 … 20]
GA	Number of individuals	200
	Crossover probability	{0.3, 0.4}
	Selection operator	"tournament"
	Mutation probability	[0.6 … 0.8]
	Crossover operator	one-point
	Mutation operator	"change order"
HGA	Number of individuals	200
	Crossover probability	{0.3, 0.4}
	Selection operator	"tournament"
	Mutation probability	[0.6 … 0.8]
	Crossover operator	"exchange terminals of two concentrators", one-point
	Mutation operator	"multiple"
GAMO	Number of individuals	200
	Crossover probability	{0.3, 0.4}
	Selection operator	"tournament"
	Mutation probability	[0.6 … 0.8]
	Crossover operator	"exchange terminals of two concentrators", one-point
	Mutation operator	"multiple"
LSGA	Number of individuals	200
	Crossover probability	{0.3, 0.4}
	Selection operator	"tournament"
	Mutation probability	[0.6 … 0.8]
	Crossover operator	"exchange terminals of two concentrators", one-point
	Mutation operator	"multiple"
HDE	Number of individuals	200
	Crossover probability	{0.3, 0.4}
	Factor F	{0.9, 1.6}
	Strategy	"Best1Exp"
HSS	Number of individuals	100
	Number of best solutions in the reference set	8
	Number of most different feasible solutions in the reference set	8
	Number of iterations for diversification	[N/15 … N/2]
DDE	Number of individuals	100
	Crossover probability	[0.1 … 0.3]
	Perturbation probability	{0.8, 0.9}
	Number of perturbations	[N/10...N/5]
HPBIL	Number of individuals	30
	Mutation probability	0.3
	Mutation Shift	0.1
	Number of iterations for diversification	N/20
	Probability exploitation/exploration	0.6

When using the same number of individuals in the population, we observe that SI algorithms are more time consuming, because they use probability vectors/matrices (see Fig. 5). Since SI algorithms can produce good results with smaller populations, the differences between SI and EAs (which need more

Table 3. Results – Evolutionary Algorithms.

P	BK	GA		HGA		GAMO		HDE		LSGA		HSS		DDE		HPBIL	
		BK	***Ts***	***BK***	***Ts***	***BK***	***Ts***	***BK***	***Ts***	***BK***	***Ts***	***BK***	***Ts***	***BK***	***Ts***	***BK***	***Ts***
1	65.63	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
2	134.65	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
3	270.26	-	-	Yes	1s	Yes	1s	Yes	<5s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
4	286.89	Yes	<1s	Yes	1s	Yes	1s	Yes	<5s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
5	335.09	Yes	<1s	Yes	1s	Yes	1s	Yes	<5s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
6	371.12	-	-	Yes	1s	Yes	1s	Yes	58s	Yes	1s	Yes	1s	Yes	<1s	Yes	<1s
7	401.21	-	-	Yes	2s	Yes	2s	Yes	118s	Yes	1s	Yes	1s	Yes	<1s	Yes	<1s
8	563.19	-	-	Yes	8s	Yes	8s	Yes	274s	Yes	7s	Yes	4s	Yes	2s	Yes	3s
9	642.83	-	-	Yes	8s	Yes	8s	Yes	456s	Yes	7s	Yes	6s	Yes	3s	Yes	5s
10	4892.09	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	4897.28	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	4883.67	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

P- Problem, *BK*- Best Known Solution Fitness, *Ts* - Execution time.

Table 4. Results – TS, SI and Proposed Algorithms.

P	BK	TS		HACO		BA		GAS		QBEA	
		BK	***Ts***	***BK***	***Ts***	***BK***	***Ts***	***BK***	***Ts***	***BK***	***Ts***
1	65.63	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
2	134.65	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
3	270.26	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
4	286.89	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s	Yes	<1s
5	335.09	Yes	<1s	Yes	2s	Yes	<1s	Yes	<1s	Yes	<1s
6	371.12	Yes	<1s	Yes	3s	Yes	<1s	Yes	<1s	Yes	<1s
7	401.21	-	-	Yes	4s	Yes	<1s	Yes	<1s	Yes	<1s
8	563.19	-	-	Yes	14s	Yes	3s	Yes	1s	Yes	1s
9	642.83	-	-	Yes	25s	Yes	4s	Yes	2s	Yes	2s
10	4892.09	-	-	-	-	-	-	Yes	300s	Yes	250s
11	4897.28	-	-	-	-	-	-	Yes	300s	Yes	250s
12	4883.67	-	-	-	-	-	-	Yes	300s	Yes	250s

Table 5. Results – Evolutionary Algorithms – Average Fitnesses and Standard Deviations.

P	GA		HGA		GAMO		HDE		LSGA		HSS		DDE		HPBIL	
	AvgF	***Std***	***AvgF***	***Std***	***AvgF***	***Std***	***AvgF***	***Std***	***AvgF***	***Std***	***AvgF***	***Std***	***AvgF***	***Std***	***AvgF***	***Std***
1	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**
2	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**
3	283.13	5.62	270.52	0.24	270.52	0.23	270.35	0.06	270.32	0.06	**270.26**	**0.00**	270.40	0.09	270.28	0.03
4	295.08	1.42	287.26	0.48	287.18	0.42	286.97	0.09	286.90	0.02	**286.89**	**0.00**	**286.89**	**0.00**	**286.89**	**0.00**
5	350.69	2.67	335.75	0.60	336.00	0.67	335.42	0.16	335.34	0.25	**335.09**	**0.01**	335.11	0.02	**335.09**	**0.01**
6	388.21	1.80	371.95	0.33	371.95	0.30	371.60	0.17	371.57	0.22	371.26	0.13	371.51	0.29	371.43	0.13
7	441.56	3.51	402.36	0.43	402.42	0.49	401.58	0.12	401.87	0.24	**401.36**	0.15	401.57	0.26	401.62	0.14
8	623.16	2.86	564.03	0.41	564.14	0.36	564.03	0.21	563.59	0.24	**563.37**	**0.10**	563.68	0.32	563.70	0.18
9	784.68	2.91	643.88	0.45	643.98	0.53	646.65	0.61	643.83	0.41	643.53	0.35	643.77	0.44	644.00	0.34
10	6891.50	28.1	4959.69	3.26	4966.05	3.24	5167.41	3.83	4915.46	2.24	4937.87	4.24	4900.03	1.62	5053.37	3.41
11	6736.08	33.37	4969.92	2.59	4972.09	3.17	5158.72	3.44	4917.96	2.12	4942.99	3.17	4902.33	1.47	5063.75	4.64
12	6673.56	33.21	4956.34	2.56	4961.78	2.53	5143.74	4.11	4902.12	2.05	4925.28	4.11	4888.33	1.26	5042.69	3.63

P- Problem, *AvgF*- Average Fitnesses, *Std* - Standard Deviations.

population diversity) in terms of execution time are not relevant. For these reasons, we consider that the number of evaluations is not suitable to perform comparisons between EAs, SI and hybrid algorithms.

We establish the number of iterations based on preliminary observations on the convergence of the algorithms.

All algorithms reach feasible solutions for all test instances. GAS, QBEA, HACO, BA, HSS, HPBIL and DDE algorithms can reach the best-known solutions for the 9 smaller instances (see Table 3 and Table 4). HDE, HGA, GAMO, LSGA and HACO can also find the best–known solutions, but in a higher execution time.

Table 6. Results – TS, SI and Proposed Algorithms – Average Fitnesses and Standard Deviations.

P	TS		HACO		BA		GAS		QBEA	
	AvgF	*Std*	*AvgF*	*Std*	*AvgF*	*Std*	*AvgF*	*Std*	*AvgF*	*Std*
1	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**	**65.63**	**0.00**
2	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**	**134.65**	**0.00**
3	270.48	0.15	270.32	0.06	270.29	0.04	270.35	0.07	270.38	0.10
4	287.93	0.75	286.91	0.04	**286.89**	**0.00**	**286.89**	**0.00**	**286.89**	**0.00**
5	336.00	0.66	335.11	0.03	335.11	0.02	335.13	0.07	335.11	0.02
6	372.35	0.51	371.55	0.17	**371.21**	**0.10**	371.30	0.15	371.33	0.16
7	403.29	0.76	401.61	0.15	401.45	**0.12**	401.56	0.16	401.70	0.22
8	564.34	0.59	563.55	0.16	**563.37**	0.11	563.50	0.19	563.45	0.14
9	644.04	0.53	643.67	0.38	**643.41**	**0.25**	643.68	0.39	643.63	0.37
10	5090.97	7.64	4913.02	3.80	4910.30	2.42	4898.83	1.40	**4898.58**	**1.30**
11	5105.23	8.08	4917.80	6.23	4914.10	1.45	4902.07	**1.35**	**4901.83**	**1.35**
12	5075.97	6.48	4904.65	8.56	4893.37	1.18	**4887.29**	1.17	4887.31	**1.12**

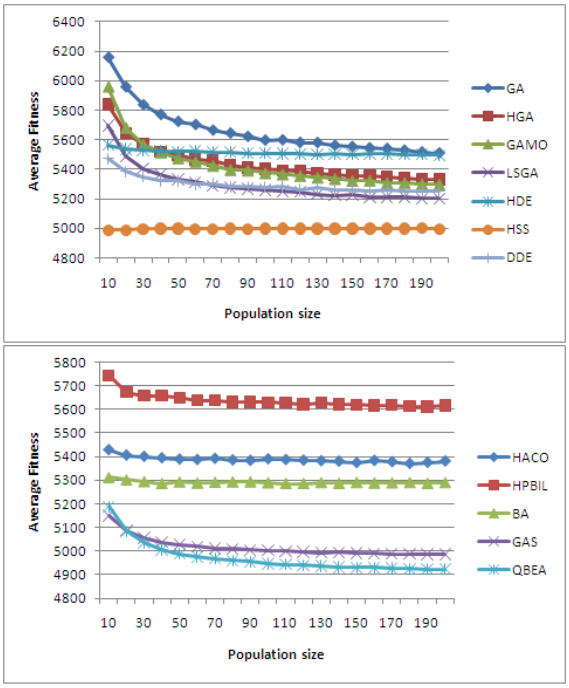

Fig. 4. Results – average fitness – instance 10. The values have been computed based on *30* different executions for each population size {*10*, *20*, ..., *200*}. For stop criterion we use *1000* iterations.

The TS algorithm is the fastest algorithm and can find good solutions in a reasonable running time. For the harder instances (10-12), GAS and QBEA are the best algorithms.

As it can be seen in table 6, for large instances (10-12), the standard deviations and the average fitnesses for GAS and QBEA are smaller. It means that these algorithms are more robust to solve large instances than GA, TS, HGA, GAMO, LSGA, HDE, HSS, DDE, HPBIL, HACO and BA. For smaller instances, BA and HSS are the best algorithms.

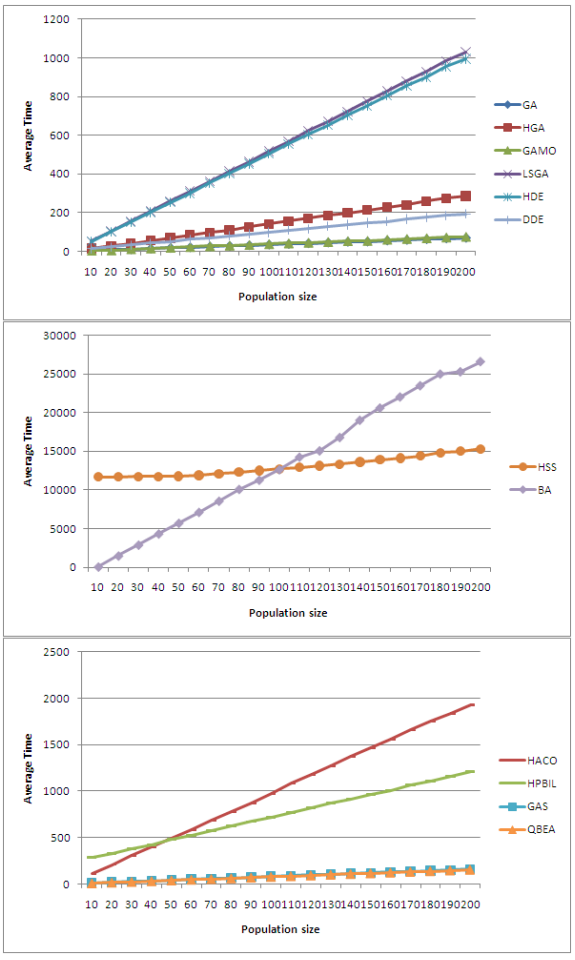

Fig. 5. Results – average time – instance 10. The values have been computed based on *30* different executions for each population size {*10*, *20*, ..., *200*}. For stop criterion we use *1000* iterations.

Comparing with GAS, QBEA presents a better standard deviation and it is less time consuming.

The size of the initial population in HSS does not have a significant influence on the execution time (see Fig. 4). However, when a number of seconds for the stop criterion is used, we observe that HSS has a poor performance (see Fig. 6). We observe that HSS can find good solutions, but it needs more time. In comparison with HSS, GAS and QBEA are faster. GA, HGA and GAMO are faster and can produce good results if the stop criterion is a limited number of seconds (see Fig. 6). However, using a large amount of time or a large number of iterations, we observe an early convergence of the algorithms. GA, HGA and GAMO often get trapped in local minimums. The LS method used in LSGA, HDE, BA, DDE, HSS, HPBIL, QBEA and GAS avoids getting trapped in local minimums.

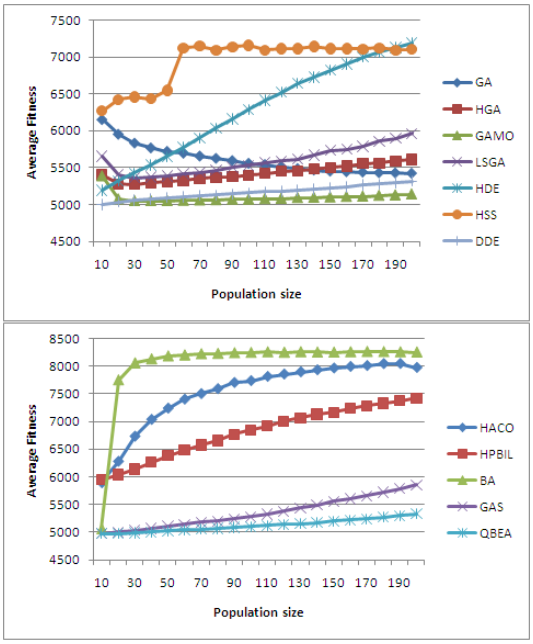

Fig. 6. Results – average fitness – instance 10. The values have been computed based on *30* different executions for each population size {*10*, *20*, ..., *200*}. For stop criterion we use *60* seconds.

Using different population sizes we observe that QBEA and GAS work better with smaller populations (see Fig. 4 and Fig. 5). With a small initial population these algorithms can find faster better solutions. With large populations the algorithms can reach better results but the algorithms slow down (see Fig. 4 and Fig. 5). For large instances, the better algorithms in terms of execution time/quality of solutions are GAS and QBEA. We observe that GAS and QBEA present the best characteristics of EAs and SI algorithms - the proposed algorithms are faster and can find with a good precision good solutions. The algorithms have a good performance with small populations. The feedback information increases the precision of the algorithms, and the LS method avoids the early convergence of the algorithms.

6. Conclusions

In this paper, we proposed a Genetic Algorithm with a new swarm mutation operator and a Queen-bee Evolutionary Algorithm to optimise the Terminal Assignment Problem. These algorithms are a combination between Evolutionary Algorithms and Swarm Intelligence techniques. The algorithms were tested in small, medium and large TAP instances. The results were compared with the ones obtained with Genetic Algorithm, Tabu Search, Hybrid GA, Local Search GA, GA with Multiple Operators, Hybrid Differential Evolution, Bees Algorithm, Discrete Differential Algorithm, Hybrid Scatter Search Algorithm and Hybrid Population Based Incremental Learning. The results indicate that the proposed algorithms have a better performance for large instances.

This paper indicates that the combination of Evolutionary Algorithms with Swarm Intelligence can be very effective. The proposed algorithms are easy to apply and we suggest their application to other assignment problems. Furthermore, as a relatively straightforward extension, the algorithms can be modified to optimise multi-objective optimisation problems.

Acknowledgements

This work has been partially supported by the Polytechnic Institute of Leiria (Portugal) and the MSTAR Project. Reference: TIN 2008-06491-C04-04/TIN (MICINN Spain).

References

1. E. Eiben and J. Smith, *Introduction to Evolutionary Computing* (Springer-Verlag, New York, 2003). ISBN: 3-540-40184-9.
2. J. Kennedy, R. C. Eberhart and Y. Shi, *Swarm intelligence*, 1st edn. (Morgan Kaufmann, San Francisco, CA, 2001). ISBN: 1558605959.

3. S. Khuri and T. Chiu, Heuristic Algorithms for the Terminal Assignment Problem, in Proc. *ACM Symposium on Applied Computing* (ACM, New York, 1997), pp. 247–251. ISBN:0-89791-850-9.
4. S. Salcedo-Sanz and X. Yao, A hybrid Hopfield network-genetic algorithm approach for the terminal assignment problem, *IEEE Transaction On Systems, Man and Cybernetics*, **34**(6) (2004) 2343–2353. DOI: 10.1109/TSMCB.2004.836471.
5. X. Yao, F. Wang, K. Padmanabhan and S. Salcedo-Sanz, Hybrid evolutionary approaches to terminal assignment in communications networks, in *Recent Advances in Memetic Algorithms and related search technologies* (Springer, Berlin / Heidelberg, 2005), pp. 129–159. DOI: 10.1007/3-540-32363-5_7.
6. G. H. M. Kapantow, *Solving concentrator location and terminal assignment problems using simulated annealing*, Masters thesis (Concordia University, Canada, 1996).
7. Y. Xu, S. Salcedo-Sanz and X. Yao, Non-standard cost terminal assignment problems using tabu search approach, in *IEEE Congress on Evolutionary Computation* (IEEE, Portland, Oregon, USA, 2004), vol. 2, pp. 2302–2306. DOI: 10.1109/CEC.2004.1331184.
8. E. Bernardino, *Minimización de Interferencias y Asignación de Terminales en Telecomunicaciones utilizando Métodos Heurísticos*, Diploma de Estudios Avanzados (Universidad de Extremadura, Spain, 2007).
9. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Tabu Search vs Hybrid Genetic Algorithm to solve the terminal assignment problem, in *IADIS International Conference Applied Computing* (IADIS, 2008), pp. 404–409. ISBN: 978-972-8924-56-0.
10. F. Abuali, D. Schoenefeld and R. Wainwright, Terminal assignment in a Communications Network Using Genetic Algorithms, in Proc. *22nd Annual ACM Computer Science Conference* (ACM, New York, 1994), pp. 74–81. DOI: 10.1145/197530.197559.
11. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Solving the Terminal Assignment Problem Using a Local Search Genetic Algorithm, in *International Symposium on Distributed Computing and Artificial Intelligence* (Springer, Berlin / Heidelberg, 2008), vol. 50, pp. 225–234. DOI: 10.1007/978-3-540-85863-8_27.
12. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, A Genetic Algorithm with Multiple Operators for Solving the Terminal Assignment Problem, in *New Challenges in Applied Intelligence Technologies*, Studies in Computational Intelligence, eds. N.T. Nguyen, R. Katarzyniak (Springer, Berlin / Heidelberg, 2008), pp. 279–288. DOI: 10.1007/978-3-540-79355-7_27.
13. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, A Hybrid Differential Evolution Algorithm for solving the Terminal assignment problem, in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living* (Springer, Berlin / Heidelberg, 2009), vol. 5518, pp. 179–186. DOI: 10.1007/978-3-642-02481-8_25.
14. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, A Hybrid Differential Evolution Algorithm with a multiple strategy for solving the Terminal assignment problem, in *Artificial Intelligence: Theories, Models and Applications*, Lecture Notes in Computer Science (Springer, Berlin / Heidelberg, 2010), pp. 303–308. DOI: 10.1007/978-3-642-12842-4_34.
15. B. A. Julstrom, Evolutionary codings and operators for the terminal assignment problem, in *Proc. 11th Annual conference on Genetic and evolutionary computation* (ACM, New York, 2009) pp. 1805–1806. DOI: 10.1145/1569901.1570171.
16. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, A Hybrid Scatter Search Algorithm to assign terminals to concentrators, in Proc. *IEEE Congress on Evolutionary Computation* (IEEE Computer Society, Los Alamitos, CA, USA, 2010), pp. 329–336. ISBN: 978-1-4244-6909-3.
17. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Discrete Differential Evolution Algorithm for solving the Terminal Assignment Problem, in *Parallel Problem Solving from Nature – PPSN XI*, Lecture Notes in Computer Science (Springer, Berlin / Heidelberg, 2010), vol. 6239, Part II, pp. 229–239. DOI: 10.1007/978-3-642-15871-1_24.
18. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Hybrid Population-Based Incremental Learning to assign terminals to concentrators, in *International Conference on Evolutionary Computation* (INSTIC, Portugal, 2010). ISBN: 978-989-8425-31-7.
19. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, A Hybrid Ant Colony Optimization Algorithm for Solving the Terminal Assignment Problem, in *International Conference on Evolutionary Computation* (INSTIC, Portugal, 2009), pp. 144–151. ISBN: 978-989-674-014-6.
20. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Using the Bees Algorithm to assign terminals to concentrators, in *Trends in Applied Intelligent Systems*, Lecture Notes in Computer Science (Springer, Berlin / Heidelberg, 2010), pp. 267–276. DOI: 10.1007/978-3-642-13025-0_29.
21. H. R. Lourenço, O. Martin and T. Stutzle, Iterated local search, in *Handbook of Metaheuristics*, eds. F. Glover and G. Kochenberger (Kluwer Academic Publishers, 2003), pp. 321–353. ISBN: 1402072635.
22. T. A. Feo and M. G. C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, *Operations Research Letters*, **8**(1989) 67–71. DOI: 10.1016/0167-6377(89)90002-3.

23. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, *Science J.*, **220**(4598) (1983) 671–680. DOI: 10.1126/ science.220.4598.671.
24. V. Cerny, A Thermodynamical Approach to the Travelling Salesman Problem: an efficient Simulation Algorithm. *J. Optimization Theory and Applications*, Springer, **45**(1) (1985) 41–51. DOI: 10.1007/ BF00940812.
25. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, Equations of State Calculations by Fast Computing Machines, *J. Chemical Physics*, **21**(6) (1953) 1087–1092. DOI: 10.1063/1.1699114.
26. F. Glover, Future paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research*, **13**(5) (1986) 533–549. DOI: 10.1016/0305-0548(86)90048-1.
27. F. Glover and M. Laguna, *Tabu Search* (Kluwer Academic Publishers, 1997). ISBN: 0-7923-8187-4.
28. M. Atiqullah and S. Rao, Reliability optimization of communication networks using simulated annealing, *Microelectronics and Reliability*, **33**(1993) 1303–1319. DOI: 10.1016/0026-2714(93)90132-I .
29. S. Pierre, M. A. Hyppolite, J. M. Bourjolly and O. Dioume, Topological design of computer communication networks using simulated annealing, *Engineering Applications of Artificial Intelligence*, **8**(1995) 61–69. DOI: 10.1016/0952-1976(94)00041-K.
30. Z. Zhang and X. Ke, Solving terminal allocation problem using simulated annealing arithmetic, *Wseas Transactions on Systems*, **7**(12) (2008) 1412–1422. http://portal.acm.org/citation.cfm?id=1503532.1503537.
31. F. Glover, M. Lee and J. Ryan, Least-cost network topology design for a new service: and application of a tabu search, *Annals of Operations Research*, **33**(1991) 351–362. DOI: 10.1007/BF02073940.
32. S. J. Koh and C. Y. Lee, A tabu search for the survivable fiber optic communication network design, *Computers and Industrial Engineering*, **28**(1995) 689–700. DOI: 10.1016/0360-8352(95)00036-Z .
33. C. P. Low, An Efficient Algorithm for the Minimum Cost Min-Max Load Terminal Assignment Problem, *IEEE Communications Letters*, **9**(11) (2005) 1012–1014. DOI: 10.1109/LCOMM.2005.11011.
34. J. H. Holland, *Adaptation in Natural and Artificial Systems* (The University of Michigan Press, 1975). ISBN: 0-262-08213-6.
35. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1 edn. (Addison-Wesley, Boston, 1989). ISBN: 0201157675.
36. R. Storn and K. Price, *Differential Evolution: a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*, Technical Report TR-95-012 (ICSI, 1995)
37. R. Storn and K. Price, Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *J. Global Optimization*, **11**(1997) 341–359. DOI: 10.1023/A:1008202821328.
38. K. Price, R. Storn and J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, Natural Computing Series (Springer-Verlag, Berlin, 2005). ISBN: 3-540-20950-6.
39. Differential Evolution Website, *http://www.icsi.berkeley.edu/~storn/code.html*
40. F. Glover, Heuristics for integer programming using surrogate constraints, *Decision Sciences*, **8**(1) (1977), pp. 156–166. DOI: 10.1111/j.1540-5915.1977.tb01074.x.
41. M. Laguna, Scatter search, in *Handbook of Applied Optimization*, eds. P. M. Pardalos and M. G. C. Resende (Oxford University, 2002), pp. 183–193. ISBN: 978-0-19-512594-8.
42. Q-K. Pan, M. F. Tasgetiren, Y-C. Liang, A discrete differential evolution algorithm for the permutation flowshop scheduling problem, J. *Computers & Industrial Engineering*, **55**(4) (2008) 795-816. DOI: 10.1016/j.cie.2008.03.003.
43. S. Baluja, *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning*. Technical report CMU-CS-95-163 (School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1994).
44. S. Salcedo-Sanz, J. A. Portilla-Figueras, F. García-Vázquez and S. Jiménez-Fernández, Solving terminal assignment problems with groups encoding: the wedding banquet problem, *Engineering Applications of Artificial Intelligence*, **19**(2006) 569–578. DOI: 10.1016/j.engappai.2005.10.003.
45. M. Dorigo, V. Maniezzo and A. Colorni, P*ositive feedback as a search strategy*, Technical Report 91-016 (Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 1991).
46. M. Dorigo, V. Maniezzo and A. Colorni, The ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man, and Cybernetics*, **26**(1996) 29–41. DOI: 10.1109/3477.484436.
47. M. Dorigo, *Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale (Optimisation, learning and natural algorithms)*, Doctoral dissertation (Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy, 1991).
48. L. M. Gambardella, E. D. Taillard and M. Dorigo, Ant colonies for the quadratic assignment problem, *J. Operational Research Society*, **50**(2) (1999) 167–176. DOI: 10.1057/palgrave.jors.2600676.
49. Ant Colony Optimization Website, *http://iridia.ulb.ac.be/dorigo/ACO/ACO.html*
50. D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim and M. Zaidi, T*he Bees Algorithm*, Technical Note, Manufacturing Engineering Centre (Cardiff University, UK, 2005).
51. A. Baykasoğlu, L. Özbakır and P. Tapkan, Artificial Bee Colony Algorithm and Its Application to Generalized

Assignment Problem, in *Swarm Intelligence, Focus on Ant and Particle Swarm Optimization*, eds. Felix T.S. Chan and Manoj Kumar Tiwari (I-Tech Education and Publishing, Vienna, Austria, 2007), pp. 113–144. ISBN: 978-3-902613-09-7.

52. D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, *J. Artificial Intelligence Review* **31**(1, 4) (2009) 61–85. DOI: 10.1007/s10462-009-9127-4.
53. A. Bernardino, E. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Solving ring loading problems using Bio-inspired algorithms, *Journal of Network and Computer Applications*, **34**(2) (2011) 668–685. DOI: 10.1016/j.jnca.2010.11.003.
54. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Solving large-scale SONET Network Design Problems using Bee-inspired Algorithms, *Optical Switching and Networking*, **9**(2) (2012) 97–117. DOI: 10.1016/j.osn.2011.11.001.
55. E. Bernardino, A. Bernardino, J. Sánchez-Pérez, M. Vega-Rodríguez and J. Gómez-Pulido, Swarm optimisation algorithms applied to large balanced communication networks, *Journal of Network and Computer Applications*, (2012). DOI: 10.1016/j.jnca.2012.04.005.