HYBRID POPULATION-BASED INCREMENTAL LEARNING TO ASSIGN TERMINALS TO CONCENTRATORS

Eugénia Moreira Bernardino, Anabela Moreira Bernardino
Research Center for Informatics and Communications, Dep. of Computer Science, School of Technology and Management Polytechnic Institute of Leiria, Leiria, Portugal

Juan Manuel Sánchez-Pérez, Juan Antonio Gómez-Pulido, Miguel Angel Vega-Rodríguez
Dep. of Technologies of Computers and Communications, Polytechnic School, University of Extremadura, Cáceres, Spain

Keywords: Communication networks, Terminal assignment problem, Optimisation algorithms, Population-based incremental learning.

Abstract: In the last decade, we have seen a significant growth in communication networks. In centralised communication networks, a central computer serves several terminals or workstations. In large networks, some concentrators are used to increase the network efficiency. A collection of terminals is connected to a concentrator and each concentrator is connected to the central computer. In this paper we propose a Hybrid Population-based Incremental Learning (HPBIL) to assign terminals to concentrators. We use this algorithm to determine the minimum cost to form a network by connecting a given collection of terminals to a given collection of concentrators. We show that HPBIL is able to achieve good solutions, improving the results obtained by previous approaches.

1 INTRODUCTION

In last years, we have observed tremendous research activities in optimisation methods for communication networks. This is mainly due to the dramatic growth in the use of the Internet (Salcedo-Sanz and Yao, 2004; Yao et al. 2005). The assignment of terminals to concentrators is an important issue in communication networks' optimisation. The number of concentrators and terminals and their 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 a way that no concentrator oversteps its capacity (Khuri and Chiu, 1997; Salcedo-Sanz and Yao, 2004; Xu et al. 2004). This problem is known as Terminal Assignment Problem (TAP). Our purpose is to minimise the cost to form a network between a specified set of terminals and concentrators (Khuri and Chiu, 1997). The objective is to assign terminals to concentrators under three constraints (Bernardino et al. 2009b): (1) each terminal is assigned to one (and only one) concentrator; (2) the total number of terminals assigned to any concentrator does not overload that concentrator, i.e. is within the concentrators' capacity and (3) balanced distribution of terminals among concentrators. Under these constraints, an assignment with the lowest possible cost is sought.

The TAP is a NP-Hard combinatorial optimisation problem (Salcedo-Sanz and Yao, 2004). This means that we cannot guarantee to find the best solution in a reasonable amount of time. The intractability of this problem is a motivation for the pursuits of an algorithm to produce approximate solutions.

Estimation of distribution algorithms (EDAs) are a class of Evolutionary Algorithms (EAs). EDAs use sampling with probabilities instead of traditional crossover and mutation operators.

The Population-Based Incremental Learning (PBIL) algorithm is an EDA, proposed by Baluja (1994). The PBIL uses a stochastic guide search process to obtain new solutions based on the directional information from the previous best solution. The PBIL maintains statistics about the search space (learning probabilities) and uses them

Moreira Bernardino E., Moreira Bernardino A., Sánchez-Pérez J., Gómez-Pulido J. and Vega-Rodríguez M..
HYBRID POPULATION-BASED INCREMENTAL LEARNING TO ASSIGN TERMINALS TO CONCENTRATORS .
DOI: 10.5220/0003076301820189
In *Proceedings of the International Conference on Evolutionary Computation* (ICEC-2010), pages 182-189
ISBN: 978-989-8425-31-7

to direct its exploration (Baluja, 1994, 1996, 1997).

The algorithm produces new solutions according to the learning probabilities. Based on analysing the principle of PBIL algorithm, in this paper we present an improvement version of the PBIL algorithm. We extend the standard PBIL algorithm to work with an integer representation (terminal-based representation). The standard PBIL algorithm uses learning probabilities to build complete solutions. In our Hybrid PBIL (HPBIL) algorithm, we use the learning probabilities to perform modifications on TAP solutions. We incorporate the HPBIL with an intensification mechanism to allow returning to previous best solutions. The HPBIL also uses a diversification mechanism that periodically reinitialises all the learning probabilities.

We compare the performance of HPBIL with five algorithms: Local Search Genetic Algorithm (LSGA), Tabu Search (TS), Hybrid Ant Colony Optimisation (HACO), Hybrid Differential Evolution with a Multiple strategy (MHDE), and Hybrid Scatter Search (HSS), used in literature.

The paper is structured as follows. In Section 2 we describe the TAP; in Section 3 we describe the implemented HPBIL algorithm; in Section 4 we discuss the computational results obtained and, finally, in Section 5 we report about the conclusions.

2 TAP

In the TAP, a communication network will connect N terminals and each with T_i demand (weight) via M concentrators and each with C_j capacity. No terminal's demand exceeds the capacity of any concentrator. A terminal site has a fixed and known location CT_i (x,y). A concentrator site has also a fixed and known location CP_j (x,y).

Problem Instance:

- Terminals - a set N of n distinct terminals;
- Weights - a vector T, with the capacity required for each terminal;
- Terminals' Location - a vector CT, with the location (x,y) of each terminal;
- Concentrators - a set M of m distinct concentrators;
- Capacities - a vector C, with the capacity required for each concentrator;
- Concentrators' Location - a vector CP, with the concentrators' location (x,y).

Each terminal must be assigned to one node of the set of concentrators, in a way that no concentrator oversteps its capacity. To minimise the cost, the distances between concentrators and terminals assigned to them must be minimised. Other objective is to ensure a balanced distribution of terminals among concentrators.

Figure 1 illustrates an assignment to a problem with $N=10$ terminal sites, and $M=3$ concentrator sites. The figure shows the coordinates for the concentrators and terminals and also their capacities.

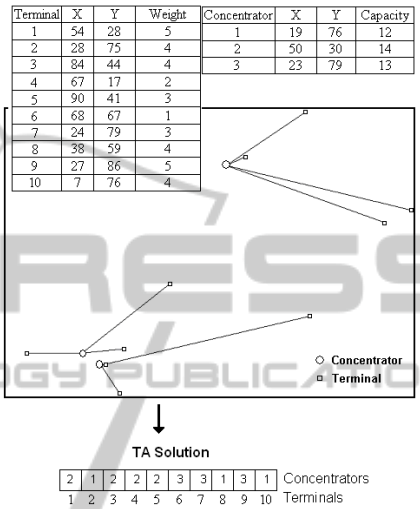

Figure 1: TAP - Example.

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

3 PROPOSED HPBIL

PBIL is a population-based optimisation method to solve hard combinatorial optimisation problems. PBIL combines characteristics from EAs and reinforcement learning (Baluja, 1994).

The PBIL creates a real-valued probability vector P, which is used to generate new solutions. The coding method of Baluja's PBIL uses binary representation (Baluja, 1994; He et al. 1999). Initially, the values of the probability vector are initialised to 0.5. As the search proceeds, the values in the probability vector gradually shift to reflect the search experience of the best solutions found in previous generations.

The probability vector is used to create a number of ns solutions. The probability vector is updated,

taking the best solution B of the previous generation. After the probability vector is updated, the vector is mutated and a new set of solutions is created by sampling from the updated probability vector. The algorithm continues until the stop criterion is reached (He et al. 1994).

The main steps of the standard PBIL are the following:

```
Initialise parameters
Initialise the probability vector P_i
Update the probability vector: P_i=0.5(1<=i<=n)
While stop criterion is not reached:
    Generate ns sample solutions according to
    the probability vector P.
    Evaluate ns solutions.
    Find best solution B.
    Update probability vector towards best
    solution B:
      P_i=P_i*(1.0-lr)+B_i*lr(1<=i<=n)
    Mutate probability vector:
     if (random(0,1)<mp)
       P_i=P_i*(1.0-mutShift)+
       random(0,1)*(mutShift)
                          (1<=i<=n)
```

ns – number of solutions in the population
n - length of encoded solution
lr - learning rate
mp - mutation probability
mutShift - amount for mutation to affect the probability vector

The probability vector in PBIL serves as distributed, numerical information which the algorithm uses to probabilistically construct solutions to the problem being solved and which the algorithm adapts during the algorithm execution, to reflect the search experience. The probabilities induce a probability distribution over the search space and determine which parts of the search space are effectively sampled.

To extend the binary PBIL algorithm for an integer representation, the probabilities need to be maintaned in a matrix. For the TAP, the set of probabilities is maintained in a matrix T of size N*M, where the entry T_{ij} measures the desirability of assigning terminal i to concentrator j.

The standard PBIL algorithm uses the probability vector to construct complete solutions. Our HPBIL algorithm uses the probabilities to perform modifications on TAP solutions.

In HPBIL, we incorporate a parameter q to control the exploration and exploitation processes. We use these two processes to modify the solutions.

The management of the probabilities is the most important component of PBIL. Exploration is a stochastic process, in which the choice of the component used to modify a solution to the problem is made in a probabilistic way. Exploitation chooses the component that maximises a blend of probability values and partial objective function evaluations.

In this paper, we also explore one of the most successful emerging ideas combining Local Search (LS) with a population-based search algorithm. HPBIL uses a modified PBIL to explore several regions of the search space and simultaneously integrates a LS algorithm to intensify the search around some selected regions.

For the best solution B in some generation, the corresponding learning formula to update the probabilities is: $T_{iBi} = T_{iBi} + lr$.

HPBIL uses an intensification mechanism. This mechanism allows returning to previous best solutions. The algorithm also uses a diversification mechanism after a predefined number of nid iterations, without improving the best solution found so far. When combined with appropriate choices for the probabilities update, the diversification mechanism can be very useful to refocus the search on a different search space region and to avoid the early convergence of the algorithm.

The main steps of HPBIL are the following:

```
Initialise parameters
iteration = 0
Generate initial population S
for s=1 to ns do
    Apply Local Search to S_s
Evaluate Solutions in S
Find best global solution G in S
Initialise the probability matrix T_ij:
     T_ij =1/M, (1<=i<=N), (1<=j<=M)
intensification=true
While stop criterion is not reached:
     iteration = iteration+1
     for s=1 to ns do
          Modify s solution according to the
          probability matrix T:
             S'_s=ModifyProcess(S_s)
          Apply Local Search to S'_s
          cond=true
          if (intensification=true)
               if(fitness(S'_s)>
                         fitness(S_s))
                    S'_s=S_s
               else
                    cond=false
     if (cond=true) //no solution improved
```

```
      intensification=false
Find best solution B in S'
if(fitness(B)<fitness(G))
      G=B
      Intensification=true
Update probability matrix towards best solution B:
      T_iBi= T_iBi + lr
Mutate probability matrix:
      if (random(0,1)<mp)
        T_ij=T_ij*(1.0-mutShift)+
         random(0 or 1)*(mutShift)
           (1<=i<=N), (1<=j<=M)
if (iteration % nid =0)
      Apply Diversification Mechanism
S=S'
```

S - population of solutions
lr - learning constant

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

3.1 Initialisation of Parameters

The following parameters, must be defined by the user: (1) ms – number of seconds; (2) ns – number of solutions in the population; (3) mp – mutation probability; (4) mutShift – amount for mutation to affect the probability matrix; (5) nm – number of modifications; (6) q – exploitation/exploration probability and (7) nid – number of iterations without improvement (used for diversification).

3.2 Create Initial Population

The solutions are created using a deterministic form. The deterministic form is based in the Greedy algorithm proposed by Abuali et al. (1994). The Greedy algorithm randomly assigns terminals to the closest feasible concentrators.

3.3 Local Search Procedure

HPBIL uses the LS algorithm proposed by Bernardino et al. (2008b). The evaluation process is the most time-consuming step of the algorithm, which is usually the case in many real-life problems. We improve the LS proposed by Bernardino et al. (2008b). After creating a neighbour, the algorithm does not perform a full examination to calculate the new fitness value; it only updates the fitness value based on the modifications made to create the neighbour. The running time is considerably reduced. We observe 80% of improvement in terms of execution time.

3.4 Evaluation of Solutions

The fitness function is the same used in Bernardino et al. (2008a, 2008b, 2009a, 2009b, 2010a, 2010b).

$$fitness = 0{,}9 * \sum_{c=0}^{M-1} bal_c + \quad (1)$$

$$0{,}1 * \sum_{t=0}^{N-1} dist_{t,c(t)} + \quad (2)$$

$$Penalisation \quad (3)$$

c(t)= concentrator of terminal t
t = terminal, c = concentrator

The fitness function is based on:

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

$$bal_c = \begin{cases} 10 & if\left(total_c = round\left(\frac{N}{M}\right)+1\right) \\ 20*abs\left(round\left(\frac{N}{M}\right)+1-total_c\right) & \end{cases}$$

$$total_c = \sum_{t=0}^{N-1} \begin{cases} 1 & if\,(c(t)=c) \\ 0 & \end{cases}$$

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

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

$$Penalisation = \begin{cases} 0 & if\,(Feasible) \\ 500 & \end{cases}$$

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

The main objective is to minimise the fitness function.

3.5 Initialisation of Probability Matrix

All the values in the probability matrix are initialised with the same probability:

$$T_{ij} = 1/M,\ (1<=i<=N),\ (1<=j<=M)$$

For example, if we have 10 terminals and 4 concentrators, all the values are initialised with the same probability ¼=0.25. All the concentrators have the same probability of being selected.

3.6 Modification of Solutions

It consists of repeating nm modifications. The modification is done assigning a terminal t to a concentrator c. First, a terminal t is randomly chosen (between 1 and N) and then a concentrator c is selected. Then, a random number x is generated between 0 and 1. If x is smaller than q (parameter), the best feasible concentrator c is chosen in a way that T_{tc} will be maximum. If x is higher than q, the feasible concentrator c is chosen with a probability proportional to the values contained in the pheromone trail. We only consider feasible concentrators. This means that we only consider the concentrators that have a free capacity equal or higher than the demand of terminal t.

3.7 Intensification Mechanism

The intensification mechanism allows a more complete exploration of the neighbourhood and allows returning to previous best solutions. The objective is to return towards attractive regions to search them thoroughly. If the intensification is active and the solution S_i in the beginning of the iteration is better, the new solution S'_i returns to the initial solution S_i. The intensification is activated when the best solution found so far has been improved and remains active while at least one solution succeeds on improving its solution during the iteration. In the end of each iteration, if no solution improves its last solution, the intensification is deactivated. The objective is to explore other regions, avoiding the algorithm to become trapped in a local minimum.

3.8 Probability Matrix Update

The probability matrix is updated by taking into account only the best solution of the previous generation. The probability matrix is updated by setting: $T_{iBi} = T_{iBi} + lr$

Based on preliminary observations, we consider the value 0.5 for lr (learning constant).

3.9 Probability Matrix Mutation

PBIL does not use a crossover operator and a selection mechanism like the most EAs. Instead, the values in T are mutated once per iteration. During this step, a random number between 0 and 1 is generated. If this random value is smaller than mp (mutation probability), the probability is mutated by setting:

$$T_{ij} = T_{ij} * (1.0 - mutShift) + random(0 \text{ or } 1) * (mutShift)$$

3.10 Diversification Mechanism

The diversification mechanism restarts the probability matrix and creates new solutions. For the following iteration, we kept the best solution found so far (G).

3.11 Termination Criterion

The algorithm stops when a maximum number of seconds (ms) is reached.

More information on PBIL can be found in (Baluja, 1994, 1996, 1997; Baluja and Caruana, 1995; He et al. 1999).

4 RESULTS

In order to test the performance of our approach, we use a collection of TAP instances of different sizes. We take 9 problems from literature (Bernardino et al. 2008a).

To compare our results we consider the results produced with LSGA, TS, HACO, MHDE and HSS. We compare our algorithm with the algorithms proposed by Bernardino et al. (2008a, 2008b, 2009b, 2010a, 2010b), because they (1) used the same test instances; (2) adopted the same fitness function; (3) implemented the algorithms using the same language (C++), and; (4) adopted the same representation (terminal-based).

Table 1 presents the best-obtained results with HPBIL, LSGA, TS, MHDE, HACO and HSS. The first column represents the number of the problem (Prob) and the remaining columns show the results obtained (BestF – Best Fitness, Ts – Run Times). The initial solutions for all algorithms were created using the Greedy algorithm. The algorithms have been executed using a processor Intel Core Duo T2300. The Ts (Run Time) corresponds to the execution time that each algorithm needs to obtain the best feasible solution.

The HPBIL algorithm can reach the best-known solutions for all instances. MHDE, HACO, HSS and LSGA can also find the best–known solutions, but in a higher execution time. Since we are not trying to dynamically assign terminals to concentrators, the running time is not enough to determine the quality of the algorithms. The best-known solutions are

Table 1: Results.

Prob	LSGA		TS		MHDE		HACO		HSS		HPBIL	
	BestF	Ts	BestF	Ts	BestF	Ts	BestF	Ts	BestF	Ts	BestF	Ts
1	65.63	<1s	65.63	<1s	65.63	<1s	65.63	<1s	65.63	<1s	65.63	<1s
2	134.65	<1s	134.65	<1s	134.65	<1s	134.65	<1s	134.65	<1s	134.65	<1s
3	270.26	<1s	270.26	<1s	270.26	<1s	270.26	<1s	270.26	<1s	270.26	<1s
4	286.89	<1s	286.89	<1s	286.89	<1s	286.89	<1s	286.89	<1s	286.89	<1s
5	335.09	<1s	335.09	<1s	335.09	<1s	335.09	2s	335.09	<1s	335.09	<1s
6	371.12	1s	371.12	<1s	371.12	<1s	371.12	3s	371.12	1s	371.12	<1s
7	401.21	1s	401.49	1s	401.21	2s	401.21	4s	401.21	1s	401.21	**<1s**
8	563.19	7s	563.34	1s	563.19	10s	563.19	14s	563.19	4s	563.19	**3s**
9	642.83	7s	642.86	2s	642.83	15s	642.83	25s	642.83	6s	642.83	**5s**

Table 2: Results – fitnesses and standard deviations.

Prob	LSGA		TS		MHDE		HACO		HSS		HPBIL	
	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**
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**
3	270.69	0.23	270.76	0.30	270.75	0.15	270.42	0.08	270.35	0.06	**270.26**	**0.01**
4	286.99	0.13	287.93	0.75	287.17	0.14	286.96	0.08	286.90	0.01	**286.89**	**0.00**
5	335.99	0.60	335.99	0.59	336.55	0.39	335.79	0.34	335.20	0.14	**335.09**	**0.00**
6	371.68	0.24	372.44	0.45	373.19	0.42	372.45	0.39	371.70	0.24	**371.41**	**0.13**
7	402.41	0.50	403.25	0.73	403.61	0.33	402.28	0.40	401.82	0.34	**401.61**	**0.15**
8	564.94	0.52	564.50	0.54	572.04	0.76	565.64	0.84	563.87	0.37	**563.76**	**0.18**
9	646.52	0.84	644.18	0.48	648.46	0.48	644.82	0.58	643.94	0.51	**643.35**	**0.19**

reached with almost all algorithms (except TS). For that reason, to establish which is the best algorithm, we must observe the average quality of the produced solutions and the standard deviations.

Table 2 presents the average fitnesses and standard deviations. The first column represents the number of the problem (Prob) and the remaining columns show the results obtained (AvgF – Average Fitness, Std – Standard Deviation). To compute the results in table 2, we use `0.5` second for instances 1-3, `1` second for instances 4-5, `2` seconds for instance 6, `5` seconds for instance 7, `10` seconds for instance 8 and `15` seconds for instance 9.

The suggestions from literature helped us to guide our choice of parameter values for TS (Bernardino et al. 2008a), LSGA (Bernardino et al. 2008b), MHDE (Bernardino et al. 2010a), HACO (Bernardino et al. 2009b) and HSS (Bernardino et al. 2010b). For the TS, we consider a number of elements in the tabu list between `5` and `20`. The parameters of LSGA are set to crossover probability between `0.3` and `0.4`, selection operator= "tournament", mutation probability between `0.6` and `0.8`, crossover operator="one-point" and mutation operator= "multiple". The parameters of the MHDE algorithm are set to crossover probability between `0.3` and `0.4`, factor `F` between `0.9` and `1.6` and strategy="Best1Exp". The parameters of the HACO algorithm are set to the number of iterations used for diversification between `200` and `400`, `Q=100`, `q=0.9`, pheromone influence=`0.8`, pheromone evaporation=`0.8` and number of modifications between `2` and `10`. The parameters of the HSS algorithm are set to `ni=100`, `b1=8`, `b2=8`, and `nid` between `N/15` and `N/2`. The parameters of HPBIL are set to `mp=0.3`, `mutShift=0.1`, `nid<N/20` and `q=0.6`. The MHDE and LSGA were applied to populations of `200` individuals, HSS to populations of `100` individuals and HPBIL and HACO to populations of `30` individuals.

The values presented in table 2 have been computed based on `50` different executions (`50` best executions out of `100` executions) for each test instance.

As it can be seen in table 2, for larger instances the standard deviations and the average fitnesses for the HPBIL algorithm are smaller. It means that the HPBIL algorithm is slightly more robust than LSGA, TS, HACO, MHDE and HSS.

The best results obtained with HPBIL use `nm<N/20`, `mp` between `0.2` and `0.7` (Figure 2) and `mutShift>=0.1` (Figure 2), `q` between `0.3 and 0.6` (Figure 2), number of solutions between

30 and 100 and nid in the range [N*3 and N*4]. These parameters were experimentally found to be good and robust for the problems tested.

We perform comparisons between all parameters (using the 9 instances) in order to establish the correct parameter setting for the HPBIL algorithm. We consider the same instance – 7 (a problem with average difficulty) to show the comparisons between parameters. To compute the results we use 1000 iterations.

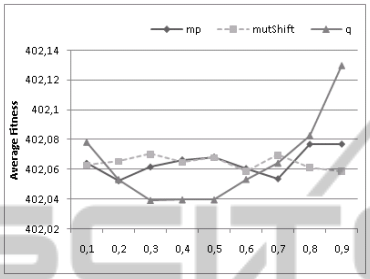

Figure 2: Influence of parameters – Problem 7.

In our experiments we use a growing number of solutions. The number of solutions (ns) was set to {10, 20, 30, 40, …, 200}. We studied the impact on the execution time and the average fitness (Figures 3 and 4). A high number of solutions significantly increases the algorithm execution time (Figure 3). The results show that the best values are in the range [30 and 100]. With these values, the algorithm can reach, in a reasonable amount of time, a reasonable number of good solutions. With a higher number of solutions, the algorithm can reach a better average fitness (Figure 4), but it is more time consuming. We also observe that a small number of solutions allows an initial faster convergence, but a worse final result, following to an increased amount of suboptima values. This can be explained, because the quality of the initial best-located solution previous to the first restart depends highly on the population size: they need more population diversity – it depends on the population size – to avoid premature stagnation.

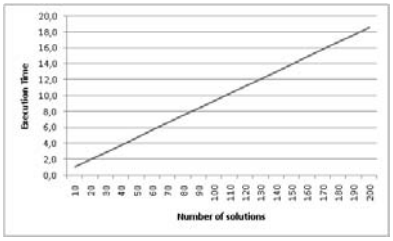

Figure 3: ns – Execution Time – Problem 7.

For nid<N*3 and nid>N*4 we observed phenomena of stagnation and insufficient intensification.

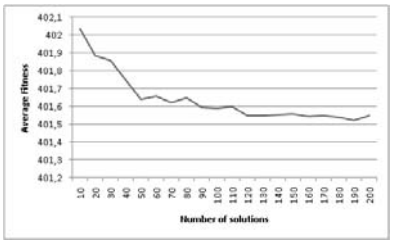

Figure 4: ns – Average Fitness – Problem 7.

For parameter nm, the number of modifications, nm<N/20 has been shown experimentally more efficient (Figure 5). A high nm has a significant impact on the execution time (Figure 6). A small nm did not allow the system to escape from local minima, because after the LS, the resulting solution was, in most cases, the same as the starting solution.

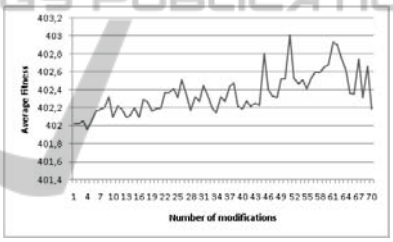

Figure 5: nm – Average Fitness – Problem 7.

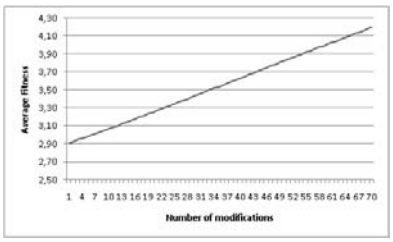

Figure 6: nm – Execution Time – Problem 7.

In general, experiments have shown that the proposed parameter setting is very robust to small modifications.

5 CONCLUSIONS

In this paper we present a HPBIL algorithm to solve the TAP. The performance of' our algorithm is compared with five algorithms: a LSGA, a TS algorithm, a HACO algorithm, a MHDE algorithm

and a HSS algorithm. All algorithms were applied to TAP by the same authors.

HPBIL presents better results for TAP. The experimental results show that the proposed algorithm is an effective and competitive approach in composing satisfactory results with respect to solution quality and execution time for TAP. Moreover, in terms of standard deviation, the algorithm also proved to be more stable and robust than the other algorithms.

For future work we suggest the implementation of Evolutionary Swarm Intelligence algorithms. The combination of EAs and SI algorithms can unify the fast speed of EAs for global solutions and good precision of SI algorithms for good solutions by the feedback information.

REFERENCES

Abuali, F., Schoenefeld, D., Wainwright, R., 1994. Terminal assignment in a Communications Network Using Genetic Algorithms. In *Proc. of the 22nd Annual ACM Computer Science Conference*, pp. 74–81. ACM Press.

Baluja, S., 1994. *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.

Baluja, S., 1996. Genetic Algorithms and Explicit Search Statistics. In *Advances in Neural Information Processing Systems*, pp. 319-325. MIT Press.

Baluja, S., 1997. Prototyping Intelligent Vehicle Modules Using Evolutionary Algorithms. In E*volutionary Algorithms in Engineering Applications*, pp. 24 1-257. Springer-Verlag.

Baluja, S., Caruana, R.. 1995. Removing the genetics form the standard genetic algorithm. In *Proceeding of the International Conference on Machine Learning*, pp. 38-46.

Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J., 2008a. Tabu Search vs Hybrid Genetic Algorithm to solve the terminal assignment problem. In *IADIS International Conference Applied Computing*, pp. 404–409. IADIS Press.

Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J., 2008b. Solving the Terminal Assignment Problem Using a Local Search Genetic Algorithm. In *International Symposium on Distributed Computing and Artificial Intelligence*, pp. 225-234. Springer.

Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J., 2009a. A Hybrid Differential Evolution Algorithm for solving the Terminal assignment problem. In *International Symposium on Distributed Computing and Artificial Intelligence 2009*, pp. 178–185. Springer.

Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J., 2009b. A Hybrid Ant Colony Optimization Algorithm for Solving the Terminal Assignment Problem. In *International Conference on Evolutionary Computation, 2009*, pp. 144-151. Springer.

Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J., 2010a A Hybrid Differential Evolution Algorithm with a Multiple Strategy for Solving the Terminal Assignment Problem. In *6th Hellenic Conference on Artificial Intelligence 2010*, pp. 303-308. Springer.

Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J., 2010b. A Hybrid Scatter Search Algorithm to assign terminals to concentrators. In *Proc. of the 2010 IEEE Congress on Evolutionary Computation*, pp. 1-8. IEEE Computer Society. Los Alamitos, CA, USA.

He, Z., Wei, C., Jin, B., Pei, W., Yang, L., 1999. A new population-based incremental learning method for the traveling salesman problem. In *Proc. of the 1999 Congress on Evolutionary Computation*, vol. 2, pp. 1152-1156. IEEE.

Khuri, S., Chiu, T., 1997. Heuristic Algorithms for the Terminal Assignment Problem. In *Proc. of the ACM Symposium on Applied Computing*, pp. 247–251. ACM Press.

Salcedo-Sanz, S., Yao, X., 2004. A hybrid Hopfield network-genetic algorithm approach for the terminal assignment problem. *IEEE Transaction On Systems, Man and Cybernetics*, 2343–2353.

Xu, Y., Salcedo-Sanz, S., Yao, X. 2004 Non-standard cost terminal assignment problems using tabu search approach. In *IEEE Conference in Evolutionary Computation*, vol. 2, pp. 2302–2306.

Yao, X., Wang, F., Padmanabhan, K., Salcedo-Sanz, S., 2005. Hybrid evolutionary approaches to terminal assignment in communications networks. In *Recent Advances in Memetic Algorithms and related search technologies*, vol. 166, pp. 129–159. Springer, Berlin.