Discrete Differential Evolution Algorithm for Solving the Terminal Assignment Problem

Eugénia Moreira Bernardino[1], Anabela Moreira Bernardino[1], Juan Manuel Sánchez-Pérez[2], Juan Antonio Gómez-Pulido[2], and Miguel Angel Vega-Rodríguez[2]

[1] Research Center for Informatics and Communications, Dept. of Computer Science, School of Technology and Management, Polytechnic Institute of Leiria, 2411 Leiria, Portugal
{eugenia.bernardino,anabela.bernardino}@ipleiria.pt

[2] Dept. of Technologies of Computers and Communications, Polytechnic School, University of Extremadura, 10071 CÃ¡ceres, Spain
{sanperez,jangomez,mavega}@unex.es

Abstract. Terminal Assignment is an important problem in telecommunication networks. The main objective is to assign a given collection of terminals to a given collection of concentrators. In this paper, we propose a Discrete Differential Evolution (DDE) algorithm for solving the Terminal Assignment problem. Differential Evolution algorithm is an evolutionary computation algorithm. This method has proved to be of practical success in a variety of problem domains. However, it does not perform well on dealing with Terminal Assignment problem because it uses discrete decision variables. To remedy this, a DDE algorithm is proposed to solve this problem. The results are compared to those given by some existing heuristics. We show that the proposed DDE algorithm is able to achieve feasible solutions to Terminal Assignment instances, improving the results obtained by previous approaches.

Keywords: Communication Networks, Optimisation Algorithms, Discrete Differential Evolution Algorithm, Terminal Assignment Problem.

1 Introduction

In recent years, several combinatorial optimisation problems have arisen in communication networks. This is mainly due to the dramatic growth of communication networks, their increasing complexity and the heterogeneity of the connected equipments. In centralised computer networks, several terminals or workstations are serviced by a central computer. In large networks, concentrators are used to increase the network efficiency. In these networks a set of terminals is connected to a concentrator and each concentrator is connected to the central computer. If the number of concentrators and their locations are known, the problem then is reduced to determine what terminals will be serviced by each concentrator. This is known as the Terminal Assignment (TA) problem. 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 [1], [2], [3]. The optimisation goals are to simultaneously produce feasible solutions, minimise the distances between concentrators and terminals assigned to them and maintain a balanced distribution of terminals among concentrators. The TA problem is a NP-Hard combinatorial optimisation problem. Therefore, finding a polynomial time algorithm to solve them to optimality is highly unlikely. In this paper we propose a Discrete Differential Evolution (DDE) algorithm to solve this problem. Our algorithm is based on the DDE algorithm proposed by Pan et al. [4] for solving the permutation flowshop scheduling problem. The DDE algorithm first mutates a target population to produce the mutant population [4]. 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. A destruction and construction procedure is employed to generate the mutant population. Embedded in the DDE algorithm we use a Local Search (LS) proposed by Bernardino et al. [5], which is used to improve the solutions quality. We compare the performance of DDE with three algorithms: Local Search Genetic Algorithm (LSGA), Tabu Search (TS) algorithm and Hybrid Differential Evolution algorithm with a multiple strategy (MHDE), used in literature.

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

2 Terminal Assignment Problem

The TA problem involves to determine what terminals will be serviced by each concentrator [1]. In the TA problem a communication network will connect N terminals and each with demand (weight) L_i, via M concentrators and each with capacity C_j. 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)*.

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 chromosome specifies the concentrator that terminal i is to be assigned to.

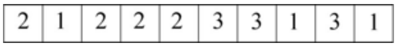

Fig. 1. Terminal-based representation

3 The Proposed DDE Algorithm

Differential Evolution (DE) was introduced by Storn and Price in 1995 [6]. DE is a population-based algorithm using crossover, mutation and selection operators [7]. The crucial idea behind DE is a scheme for generating trial parameter vectors [8], [9]. At the mutation step, distinct individuals (usually three) are selected from population. Mutation adds the weighted difference of two (or more) of the individuals to the third. At the recombination step, new trial individuals are created by combining the mutated individual, with the target individual. Combination takes place according to a strategy (several strategies with different approaches can be found in literature - see [9]). Thereafter, a selection operator is applied to compare the fitness value of both competitive solutions, namely, target and trial solutions to determine who can survive for the next generation. Most of the discrete problems have solutions presented through permutation vectors, while DE usually maintains and evolves floating-point vectors.

In order to apply the DE to solve discrete problems, the most important is to find a suitable encoding scheme, which can transform between floating-point vectors and permutation vectors. DE was first applied to TA problem by Bernardino et al. [5]. They proposed a Hybrid DE (HDE) algorithm and in [10] proposed an improved HDE algorithm with a multiple strategy (MHDE). MHDE combines global and LS by using an evolutionary algorithm to perform exploration while the LS method performs exploitation. MHDE uses the terminal-based representation and not floating-point vectors. After applying the standard equations, the algorithm verifies if the trial solutions contain values outside the allowed range.

In MHDE, if a gene value (concentrator) is outside of the allowed range it is necessary to apply the following transformation:

```
IF concentrator > M THEN        concentrator = concentrator - M
ELSE IF concentrator <=0 THEN   concentrator = concentrator + M
```

This transformation significantly increases algorithm execution time. To solve this problem we use a DDE algorithm based on the algorithm proposed by Pan et al.[4] whose solutions are based on discrete values, which can be applied to all types of combinatorial optimisation problems [11], [12], [13], [14]. The basic model of DDE is different from the DE model.

Main steps of the DDE algorithm:

```
Initialise Parameters
Create initial target Population, $P^0$
Evaluate target Population $P^0$
Find Best Solution in $P^0$, $P_g$
WHILE stop criterion is not reached
  Create Mutant Population, $Pm^t$
  Create Trial Population, $Pt^t$
  Evaluate Trial Population $Pt^t$
  Make Selection and update target population, $P^t$
  Find Best solution in $P^t$, $P^t{}_g$
  Apply Local Search to $P^t{}_g$
```

Initialisation of parameters

The following parameters, must be defined by the user (1) *ni* - number of individuals; (2) *ms* - maximum number of seconds; (3) *pp* - perturbation probability; (4) *np* - number of perturbations and (5) *pc* - crossover probability.

Initial target Population

The initial target population (P^0) can be created randomly or in a Deterministic Form (DF). DF is based in the Geeedy algorithm proposed by Abuali et al. [15]. This algorithm assigns terminals to the closest feasible concentrator.

Evaluation of solutions

To evaluate how good a potential solution is relative to other potential solutions we use a fitness function. The fitness function is based on: (1) the total number of terminals connected to each concentrator (the objective is to guarantee a balanced distribution of terminals among concentrators); (2) the distance between the concentrators and the terminals assigned to them (the objective is to minimise the distances between concentrators and terminals assigned to them); (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). The objective is to minimise the fitness function (equation 4), searching a trade off among the three objectives mentioned.

$$total_c = \sum_{t=1}^{N} \begin{cases} 1 \; if(c(t) = c) \\ 0 \end{cases} \qquad bal_c = \begin{cases} 10 \;\; if(total_c = round(\frac{N}{M}) + 1) \\ 20 * |(round(\frac{N}{M}) + 1 - total_c)| \end{cases} \tag{1}$$

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

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

$$Penalisation = \begin{cases} 0 \qquad if(Feasible) \\ 500 \end{cases} \tag{3}$$

$$fitness = 0.9 * \sum_{c=1}^{M} bal_c + 0.1 * \sum_{t=1}^{N} dist_{t,c(t))} + Penalisation \tag{4}$$

Mutant Population

A mutant individual is obtained by perturbing the generation best solution in the target population. The differential variation is achieved in the form of perturbations of the best solution from the previous generation in the target population. To obtain the mutant individual, the following equation is used:

$$Pm_i^t = \begin{cases} DC_{np}(P_g^{t-1}) \quad if(r < pp) \\ MutationClosestConcentrator(P_g^{t-1}) \end{cases} \tag{5}$$

P_g^{t-1} is the best solution from the previous generation in the target population and DC_{np} is the destruction and construction procedure with the destruction size of *np* as a perturbation operator; and *MutationClosestConcentrator* is a simple mutation operator. With this operator, one gene (terminal) is randomly selected and its value (concentrator) is replaced for a new one (the closest feasible concentrator). A uniform random number r is generated between *0* and *1*. If r is less than *pp* then the *DC* procedure is applied to generate the mutant individual. Otherwise, the best solution from the previous population is perturbed using a simple mutation operator. The *DC* procedure creates a new solution by performing multiple moves whose length is specified as *np*. The algorithm performs *np* perturbations to find a new solution. 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 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 concentrators, *c1* and *c2*. The algorithm repeats this process until at least one exchange is made.

Steps of the DC method:
```
FOR n=1 TO np DO
   t = random(N), closestC=1
   FOR c=1 TO M DO    /*find the closest concentrator*/
        IF distance (t, c) < distance (t, closestC) THEN
                closestC=c
   IF capacityFree(closestC)>=L(t) and mantainBalanced(closestC) THEN
        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) ) THEN
                  Assign t1 to c2 and t2 to c1
                  cond = false
         WHILE cond=true
```

Trial Population

Following the perturbation phase, a trial individual is obtained such that:

$$Pt_i^t = \begin{cases} CR(Pm_i^t, P_i^{t-1}) & if(r < pc) \\ Pm_i^t \end{cases} \tag{6}$$

A uniform random number r is generated between *0* and *1*. If r is lower than *pc* then the crossover operator is applied to generate the trial individual. The crossover operator (*CR*) adopted was 1-point, widely used in literature. CR produces two children. In this study, we selected one of the children randomly. If r is higher or equal to pc then the trial individual is chosen as: $Pt_i^t = Pm_i^t$ The trial individual is made up either from the perturbation operator or from the crossover operator.

Selection
The selection is based on the survival of the fitness among the trial and target individuals such that:

$$P_i^t = \begin{cases} Pt_i^t \;\; if(fitness(Pt_i^t) < fitness(P_i^{t-1})) \\ P_i^{t-1} \end{cases} \tag{7}$$

Local Search
The LS algorithm applies a partial neighbourhood examination. 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 better neighbour improves the actual solution then the LS algorithm replaces the actual solution with the better neighbour. Otherwise, the algorithm creates another set of neighbours. In this case, one neighbour results of 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)*.

Steps of the LS algorithm:
```
c1 = random (M), c2 = random (M)
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) THEN
    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) THEN
       ACTUAL-SOL = SOLUTION
ELSE
    ACTUAL-SOL = SOLUTION
```

The evaluation process is the most time-consuming step of the algorithm, which is usually the case in many real-life problems. Our LS procedure has some important improvements compared to the LS proposed by Bernardino et al. [5]. 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 that were made to create the neighbour. The running time is reduced considerably.

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

Further information on DDE can be found in [4], [11], [12], [13], [14].

4 Results

In order to test the performance of our approach, we use a collection of TA instances of different sizes. We take 9 problems from literature [16]. To compare our results we consider the results produced with LSGA, TS and MHDE. GA was first applied to TA by Abuali et al. [15]. GA is widely used in the literature to make comparisons with other algorithms. TS was applied to this problem by Xu et al. [17] and Bernardino et al. [16]. We compare our algorithm with LSGA [18], TS [16] and MHDE [10] algorithms proposed by Bernardino et al. 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 DDE, LSGA, TS and MHDE. The first column represents the problem number (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.

Table 1. Results

Prob	LSGA		TS		MHDE		DDE	
	BestF	Ts	BestF	Ts	BestF	Ts	BestF	Ts
1	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
3	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
5	335.09	<1s	335.09	<1s	335.09	<1s	335.09	<1s
6	371.12	1s	371.12	<1s	371.12	<1s	371.12	<1s
7	401.21	1s	401.49	1s	401.21	2s	401.21	**<1s**
8	563.19	7s	563.34	1s	563.19	10s	563.19	**2s**
9	642.83	7s	642.86	2s	642.83	15s	642.83	**3s**

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 $\hat{A}\frac{1}{2}$ 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 [16], LSGA [18] and MHDE [10]. 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

Table 2. Average Fitnesses and Standard Deviations

Prob	LSGA		TS		MHDE		DDE	
	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**
2	**134.65**	**0,00**	**134.65**	**0,00**	**134.65**	**0,00**	**134.65**	**0,00**
3	270.69	0.23	270.76	0.3	270.75	**0.15**	**270.47**	0.22
4	286.99	0.13	287.93	0.75	287.17	0.14	**286.89**	**0,00**
5	335.99	0.6	335.99	0.59	336.55	0.39	**335.26**	**0.17**
6	371.68	0.24	372.44	0.45	373.19	0.42	**371.38**	**0.22**
7	402.41	0.5	403.25	0.73	403.61	0.33	**401.62**	**0.28**
8	564.94	0.52	564.5	0.54	572.04	0.76	**564.07**	**0.38**
9	646.52	0.84	644.18	0.48	648.46	0.48	**643.96**	**0.46**

to crossover probability between *0.3* and *0.4*, factor F between *0.9* and *1.6* and strategy="Best1Exp". The parameters of DDE are set to crossover probability between *0.1* and *0.3*, perturbation probability between *0.8* and *0.9* and number of perturbations between *[N/10...N/5]*. The MHDE and LSGA were applied to populations of *200* individuals and DDE to populations of *100* 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.

The four algorithms reach feasible solutions for all test instances. The DDE algorithm can reach the best-known solutions for all instances. MHDE 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 a significant parameter to determine the quality of the algorithms. The differences in terms of execution time are not significant. To establish which is the best algorithm we must observe the average quality of the produced solutions and the standard deviations. As it can be seen in table 2, for larger instances the standard deviations and the average fitnesses for DDE algorithm are smaller. It means that the DDE algorithm is slightly more robust than LSGA, TS and MHDE.

We perform comparisons between all parameters (using the 9 instances) in order to establish the correct parameter setting for the DDE algorithm. We consider the same instance - 7 (a problem with average difficulty) to show the comparisons between parameters. To compute the results we use *1500* iterations.

The better results obtained with DDE use *np* between *N/20* and *N/3*, *pp*>*0.5*, *pc*<*0.5* (Fig. 2) and $ni = \{90, 100\}$. These parameters were experimentally found to be good and robust for the problems tested.

In our experiments we use different population sizes. The number of individuals was set to $\{10, 20, ..., 200\}$. We studied the impact on the execution time, the average fitness and the number of best solutions found. A higher number of initial solutions significantly increases algorithm execution time (Fig. 3).

The results show that the best population sizes are *90* 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 initial solutions the algorithm can reach a better average fitness but it is more time consuming (Fig. 3).

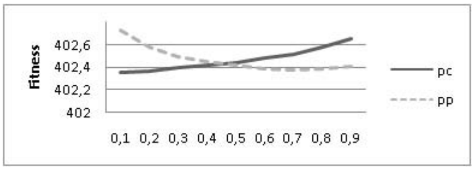

Fig. 2. Influence of parameters, Problem 7

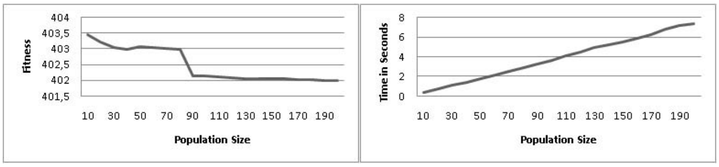

Fig. 3. Number of Individuals, Average Fitness/Execution Time, Problem 7

In our experiments *np* (number of perturbations) was set to {*0, 1, 2, ..., N*} (Fig. 4). In case of a high *np* the resulting permutation tends to be too random, which makes it more difficult to generate new improving solutions. A high *np* has also a significant impact on the execution time (Fig. 4). A small *np* did not allow the system to escape from local minima because the resulting solution was in most cases the same as the starting permutation.

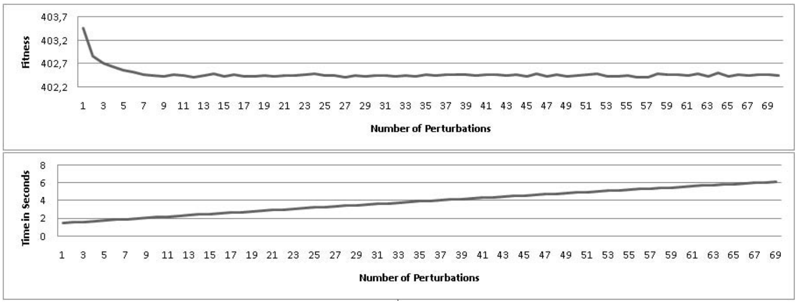

Fig. 4. Number of Perturbations, Average Fitness/Execution Time, Problem 7

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

5 Conclusions

In this paper we present a DDE algorithm to solve the TA problem. The performance of our algorithm is compared with LSGA, TS and MHDE. The computational results show that DDE performed more strongly, improving the results obtained by previous approaches. DDE provides good solutions in a smaller execution time. Moreover, in terms of average fitness and standard deviation, the DDE also proved more robust and stable than the other algorithms. Experimental results demonstrate that the proposed DDE algorithm is an effective and competitive approach in composing fairly satisfactory results with respect to solution quality and execution time for the TA problem.

In literature the application of DDE for this problem is nonexistent, for that reason this article shows its enforceability in the resolution of this problem.

For future work we propose the implementation of parallel algorithms to speed up the optimisation process.

References

1. Khuri, S., Chiu, T.: Heuristic Algorithms for the Terminal Assignment Problem. In: Proc. of the ACM Symposium on Applied Computing, pp. 247–251 (1997)
2. Salcedo-Sanz, S., Yao, X.: A hybrid Hopfield network-genetic algorithm approach for the terminal assignment problem. IEEE Transaction On Systems, Man and Cybernetics, 2343–2353 (2004)
3. Yao, X., Wang, F., Padmanabhan, K., Salcedo-Sanz, S.: 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 (2005)
4. Pan, Q.-K., Tasgetiren, M.F., Liang, Y.-C.: A discrete differential evolution algorithm for the permutation flowshop scheduling problem. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 126–133 (2007)
5. Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J.: 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, Heidelberg (2009)
6. Storn, R., Price, K.: Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report TR-95-012, ICSI (1995)
7. Storn, R., Price, K.: Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359 (1997)
8. Price, K., Storn, R., Lampinen, J.: Differential Evolution - A Practical Approach to Global Optimization. Springer, Berlin (2005)
9. Differential Evolution, `http://www.icsi.berkeley.edu/~storn/code.html`
10. Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J.: A Hybrid Differential Evolution Algorithm with a Multiple Strategy for Solving the Terminal Assignment Problem. In: 6th Hellenic Conference on Artificial Intelligence. Springer, Heidelberg (2010)
11. Tasgetiren, M.F., Pan, Q.-K., Liang, Y.-C.: A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. Computers and Operations Research 36(6), 1900–1915 (2009)
12. Tasgetiren, M.F., Pan, Q.-K., Liang, Y.-C., Suganthan, P.N.: A discrete differential evolution algorithm for the total earliness and tardiness penalties with a common due date on a single machine. In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CISched 2007), pp. 271–278 (2007)

13. Tasgetiren, M.F., Pan, Q.-K., Suganthan, P.N., Liang, Y.-C.: A discrete differential evolution algorithm for the no-wait flowshop scheduling problem with total flow-time criterion. In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CISched 2007), pp. 251–258 (2007)
14. Tasgetiren, M.F., Pan, Q.-K., Liang, Y.-C.: A discrete differential evolution algorithm for single machine total weighted tardiness problem with sequence dependent setup times. In: IEEE Congress on Evolutionary Computation, pp. 2613–2620 (2008)
15. Abuali, F., Schoenefeld, D., Wainwright, R.: Terminal assignment in a Communications Network Using Genetic Algorithms. In: Proc. of the 22nd Annual ACM Computer Science Conference, pp. 74–81. ACM Press, New York (1994)
16. Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J.: Tabu Search vs Hybrid Genetic Algorithm to solve the terminal assignment problem. In: International Conference Applied Computing, pp. 404–409. IADIS Press (2008)
17. Xu, Y., Salcedo-Sanz, S., Yao, X.: Non-standard cost terminal assignment problems using tabu search approach. In: IEEE Conference in Evolutionary Computation, vol. 2, pp. 2302–2306 (2004)
18. Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Vega-Rodríguez, M., Gómez-Pulido, J.: Solving the Terminal Assignment Problem Using a Local Search Genetic Algorithm. In: International Symposium on Distributed Computing and Artificial Intelligence, pp. 225–234. Springer, Heidelberg (2008)