Genetic and Local Search algorithms applied to balanced communication networks

Eugénia Moreira Bernardino
Anabela Moreira Bernardino
Computer Science and Communication Research Centre, School of Technology and Management, Polytechnic Institute of Leiria
Leiria, Portugal
{eugenia.bernardino, anabela.bernardino}@ipleiria.pt

Juan M. Sánchez-Pérez
Juan A. Gómez-Pulido
Miguel A. Vega-Rodríguez
Department of Technologies of Computers and Communications, Polytechnic School, University of Extremadura
Cáceres, Spain
{sanperez, jangomez, mavega}@unex.es

ABSTRACT
In this paper we describe the application of different heuristics to optimise large communication networks. We use Iterated Local Search (ILS), Tabu Search (TS), Simulated Annealing (SA) and Genetic Algorithm (GA) to minimise the link cost to form balanced communication networks. This paper makes a comparison among the effectiveness of ILS, TS, SA and GA on solving large communication networks. Simulation results verify the effectiveness of these algorithms.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *Centralized networks, Network communications.*

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Heuristic methods.*

General Terms
Algorithms, Design.

Keywords
Communication Networks, Iterated Local Search, Tabu Search, Simulated Annealing, Genetic Algorithm, Terminal Assignment Problem.

1. INTRODUCTION
Nowadays, we observe an increasing size and consequently an increasing complexity of computer networks and for that reason finding an optimal assignment of terminals to concentrators in large communication networks continues to be a hard task. The terminal assignment is an important task in communication networks [1, 8, 12-14]. The Terminal Assignment Problem (TAP) is a NP-hard optimisation problem [1, 8, 12-14] and for dealing with its difficulty, we propose GA, ILS, SA and TS. In this paper we consider a balanced distribution of terminals among concentrators. In this case, the cost of a single assignment is not known in advance, and only the cost associated with candidate solutions can be calculated. In this work, 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. To address these optimisation goals we propose Local Search (LS) methods and GA.

LS heuristics involve repeatedly going from one solution to another through a local move. They start from an initial solution and iteratively attempt to improve upon it by a series of local changes. The main drawback of LS heuristics is that the search might terminate at a local optima which may be far from a global optima. The quality of the final solution depends on the starting solution and the rules used to generate neighbouring solutions. Recently, a great deal of attention has focussed on three LS heuristics when solving complex combinatorial optimisation problems: ILS, SA and TS.

ILS is a simple and powerful stochastic LS method that creates a sequence of solutions generated by an embedded heuristic [10]. The essential idea of ILS lies in focusing the search on a smaller subspace, defined by locally optimal solutions for a given optimisation engine.

SA exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum value in a more general system [3, 9, 11]. 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 is a meta-heuristic algorithm, belonging to the class of LS techniques [5]. TS allows the search of solutions that decrease the objective function value only in those cases where these solutions are not forbidden [6].

GAs are Evolutionary Algorithms (EAs), inspired by the natural process of reproduction [4, 7]. 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 individuals. Following a standard Darwinistic approach, the selection extracts the most promising individuals from the current population [7].

We use different instances with different sizes to prove the efficiency of ILS, SA, TS and GA algorithms.

Isabel'11, October 26 - 29 2011, Barcelona, Spain.

The paper is structured as follows: in Section 2 we describe the TAP; in Section 3 we describe ILS, TS, SA and GA algorithms; in Section 4 we discuss the computational results obtained and, in Section 5 we report about the conclusions.

2. TERMINAL ASSIGNMENT

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 an Euclidean grid.

A simple and feasible method for solving this problem is to allocate each terminal to one concentrator without exceeds the capacity of any concentrator. We use a vector of integers:

$$X = X(1), X(2), X(3),..., X(N) \tag{1}$$

Where $X(t)=c$ means that terminal t is allocated to the concentrator c, and $1 \le X(t) \le M$, $1 \le t \le N$. In other words, whole terminals must be assigned to one concentrator, and

$$k(c) = \sum_{t=1}^{N} \begin{cases} L(t) & if\ (X(t)=c) \\ 0 & \end{cases}$$
$$feasible = \begin{cases} 1 & if\ \ for\ \ all\ \ c;\ 1 \le c \le M;\ k(c) \le C(c) \\ 0 & \end{cases} \tag{2}$$

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

In this work we consider TAP in the context of balanced networks. Our goal is to minimise the link cost and maintain a balanced distribution of terminals among concentrators. In this case, the fitness function depends on the entire solution provided to the computer network, and the cost of a single assignment cannot be calculated. Thus, some of the previous algorithms for TAP are no longer applicable. Salcedo-Sanz and Yao [12] propose a new fitness function to algorithms that work with binary representation. This fitness function considers the distances between terminals and concentrators assigned to them and also the number of terminals assigned to each concentrator. We adjust this fitness function to work with an integer-representation.

In this work we explore infeasible solutions. In many cases performing only a move can be enough to transform an infeasible solution into a feasible solution. The fitness function adopted gives a penalisation to the infeasible solutions.

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);

$$NTC(c) = \sum_{t=1}^{N} \begin{cases} 1 & if\ (X(t)=c) \\ 0 & \end{cases}$$
$$BAL(c) = \begin{cases} 10 & if\left(NTC(c)=round\left(\frac{N}{M}\right)+1\right) \\ 20*abs\left(round\left(\frac{N}{M}\right)+1-NTC(c)\right) & \end{cases} \tag{3}$$

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

$$D(t, X(t)) = \sqrt{(CP(X(t)).x - CT(t).x)^2 + (CP(X(t)).y - CT(t).y)^2} \tag{4}$$

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).

$$Penalisation = \begin{cases} 0 & if\ (feasible=1) \\ 500 & \end{cases} \tag{5}$$

The target of TAP is to minimise the fitness function (Eq. 6).

$$fitness = 0.9 * \sum_{c=1}^{M} BAL(c) + 0.1 * \sum_{t=1}^{N} D(t, X(t)) + Penalisation \tag{6}$$

This fitness function encourages solutions with a balanced distribution of terminals among concentrators. The fitness function penalises infeasible solutions. The Euclidean distances between terminals and concentrator are also considered and must be as small as possible. Note also, that the fitness value cannot be known before a solution is provided to the communication network. For this reason, blind search procedures are needed in order to obtain a solution to the problem.

3. LOCAL SEARCH ALGORITHMS

LS meta-heuristics are an emerging class of methods, which recently have been shown to be very effective for a large number of combinatorial problems. The LS techniques are based on the iterative exploration of a solution space: at each iteration, a LS algorithm steps from one solution to one of its neighbours (solutions that are close to the starting one). In this paper we explore several LS methods applied to TAP: ILS, TS and SA. The next subsections describe each step of ILS, TS and SA in detail.

3.1 Iterated Local Search

ILS is a meta-heuristic, which can be used to solve complex combinatorial optimisation problems [10]. ILS is a LS method that explores a sequence of solutions created as perturbations of the current best solution, the result of which is refined using an embedded heuristic. It is an extension of Mutli-Restart Search and may be considered a parent of many two-phase search approaches such as the Greedy Randomised Adaptive Search Procedure and the Variable Neighbourhood Search. ILS is based on building a sequence of locally optimal solutions by: (1) perturbing the current local minimum and (2) applying LS to the modified solution.

The main steps of the ILS applied to TAP are detailed below:

ILS Algorithm

```
Initialise Parameters
Generate initial candidate solution X
Evaluate Solution
Apply LS Procedure
WHILE TerminationCriterion()
        Apply Perturbation Method
        Apply LS Procedure
        Verify Acceptance Criterion
```

Initialisation of Parameters - The following parameters must be defined by the user: (1) *MI* = maximum number of iterations; and (2) *NM* = number of modifications (perturbations).

Solution Generation - The initial solution can be created randomly or using a deterministic form. The deterministic form is based on the Abuali et al. Greedy algorithm [1]. The Greedy algorithm assigns terminals to the closest feasible concentrators.

Evaluation of Solutions - To evaluate how good a potential solution is we use a fitness function (see Eq. 6).

Perturbation Method - As perturbation method we use a neighbourhood search mechanism. A new solution is obtained by performing multiple moves whose length is specified as *NM*.

The perturbation method consists on the following steps:

Neighbourhood Search method

```
FOR n=1 TO NM DO
   t = random(N)
   closestC=1
   FOR c=1 TO M DO //find the closest concentrator
        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
```

First, the algorithm chooses a random terminal *t* and searches the closest concentrator, *closestC*. If the concentrator has enough capacity and maintains a balanced terminal distribution, then terminal *t* is assigned to the closest concentrator. 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*.

Local Search - The LS method adopted in this study applies a partial neighbourhood search.

The LS method consists on the following steps:

LS method

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

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 current solution, then the best neighbour replaces this solution. 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*. If the fitness of the best neighbour is smaller than the fitness of the current solution, then the best neighbour replaces this solution.

Acceptance Criterion - The algorithm only accepts the solutions, which decrease the fitness value.

Termination criterion - ILS stops when a maximum number of iterations (*MI)* is reached.

3.2 Tabu Search

TS is a higher level heuristic for solving combinatorial optimisation problems. TS is an iterative improvement method that starts from a initial candidate solution and attempts to find better solutions. The basic concept of TS was described by Glover [5]. The TS algorithm explores solutions that decrease the objective function value only in those cases where these solutions are not forbidden. This is usually obtained by keeping track of the action used to transform one solution into the next. When an action is performed it is considered tabu for the next *ntl* iterations, where *ntl* is the tabu status length. A solution is forbidden if it is obtained by applying a tabu action to the current solution [6].

The main steps of the TS applied to TAP are detailed below:

TS Algorithm

```
Initialise Parameters
Initialise Tabu List
Generate initial candidate solution X
Evaluate Solution
auxElem = 0 //number of the last element updated
in the tabu list
numElem = 0 //number of elements in the tabu list
WHILE TerminationCriterion()
   REPEAT
        c1=random(M)
        c2=random(M)
   WHILE VerifyTabuList(c1,c2, numElem)
   Generate Neighbourhood N
   Select best solution X' in N    > LS method
   If fitness (X') < fitness(X)
        X = X'
   Update Tabu List:
        UpdateTabuList(c1, c2, auxElem, numElem)
```

Initialisation of Parameters - The following parameters must be defined by the user: (1) *MI* = maximum number of iterations; and (2) *ntl* = number of elements in the tabu list.

Solution Generation - The initial solution can be created randomly or using a deterministic form.

Evaluation of Solutions - We use the fitness function described in Section 2 (see Eq. 6) to evaluate the solutions' quality.

Tabu List - The solutions admitted to the new neighbourhood, are determined through the use of memory structures. The search then progresses by iteratively moving from a solution *X* to a solution *X'* in the new neighbourhood. The most important type of memory structure used to determine the solutions admitted to the new neighbourhood is the tabu list. In its simplest form, a tabu list is a short-term memory, which contains attributes of solutions that have been visited in the recent past (less than *ntl* iterations ago). TS excludes solutions whose attributes are in the tabu list.

In our implementation, a candidate can be chosen as a new current solution if the concentrators which terminals are exchanged do not match with those in the tabu list. TS only

explores neighbours when the two concentrators *c1* and *c2* used to create the neighbourhood are not in the tabu list.

Tabu List – Method to verify tabu attributes

```
VerifyTabuList(c1, c2, numElem)
   FOR i=1 TO numElem DO
        IF (TabuList(i).attribute1=c1 AND
                      TabuList(i).attribute2=c2) OR
            (TabuList(i).attribute1=c2 AND
                        TabuList(i).attribute2=c1)
             RETURN true
   RETURN false
```

numElem – number of elements in the tabu list.

The two concentrators (*c1* and *c2*) whose terminals are exchanged are classified as tabu attributes.

Tabu List – Method to update the tabu list

```
UpdateTabuList(c1, c2, auxElem, numElem)
        IF auxElem = ntl
                auxElem = 1
        ELSE
                auxElem = auxElem + 1
        IF numElem < ntl
                numElem = numElem + 1
        TabuList(auxElem).attribute1 = c1
        TabuList(auxElem).attribute2 = c2
```

numElem – number of elements in the tabu list; auxElem – number of the last element updated in the tabu list.

Generate Neighbourhood and Select best candidate solution - TS was first applied to this problem by Xu et al. [13] and it only explores a part of the neighbourhood. Our algorithm exploits also a part of the neighbourhood [2]. The search method is similar to the LS method used in ILS. The only difference is that the two concentrators chosen to create the neighbourhood cannot belong to the tabu list. The LS method searches for a better solution in the neighbourhood. The current solution is replaced if the best neighbour improves this solution. Otherwise, the current solution is maintained for the next iteration.

Aspiration - Normally in TS algorithm if a neighbour is the best solution found so far it could be selected as a move, even when it is tabu. In our implementation the algorithm does not explore neighbours when the two concentrators chosen are in the tabu list. In aspiration, just the best neighbour not tabu with a fitness value lower than the best is selected.

Termination criterion - TS stops when a maximum number of iterations (*MI*) is reached.

3.3 Simulated Annealing

SA has been used in various combinatorial optimisation problems (see [9]). The term SA derives from the roughly analogous physical process of annealing in solids. SA mimics the natural process by which crystal lattices of glass or metal relax when heated. Kirkpatrick et al. [9] took this idea and applied it to optimisation problems. The physical annealing turn into an algorithm establishing a mapping between: (1) the system states and the feasible solutions; (2) the energy and the cost (fitness value); (3) change of state and the neighbouring solution; (3) the temperature and the control parameter and (4) the frozen state and the heuristic solution. Using these mappings any optimisation problem can be converted into an annealing algorithm [3, 9] by sampling the neighbourhood randomly and accepting worse solutions.

The main steps of the SA algorithm are:

1. Select starting temperature and initial parameter values;
2. Create an initial solution *X*;
3. Randomly select a new solution *X'* in the neighbourhood of the current solution *X*. In practice, a perturbation method can be used to perform one or more moves in the solution *X*.
4. Compare the two solutions using the Metropolis criterion:

Metropolis criterion

```
delta = fitness(X)-fitness(X')
IF delta>0
    X=X'
ELSE
    r= random number between [0…1]
    m= exp(delta/temperature)
    IF (r<m)
          X=X'
```

   Downhill moves are always accepted and some uphill moves may also be accepted. Thus, the algorithm can escape from local minima. When temperature is very high, uphill moves are mostly likely acceptable. As temperature cools, fewer uphill moves are acceptable;
5. Repeat steps 3 and 4 until system reaches equilibrium state. In our implementation, the process is repeated *it* times for each temperature;
6. Decrease temperature and repeat the above steps, stop when system converges to a frozen state. In our implementation, SA stops when a maximum number of iterations is reached

To the basic model of SA we added the same LS algorithm used in ILS to intensify the search around some selected regions. This LS algorithm is applied to the initial solution. It is also applied to the current solutions after the acceptance criterion step.

The main steps of the SA applied to TAP are detailed below:

SA Algorithm

```
Initialise Parameters
temperature=ti
Generate initial candidate solution X
Evaluate Solution
Apply LS Procedure
WHILE TerminationCriterion()
    FOR it=1 to itT do
        Apply Perturbation Method to generate X'
        Evaluate Solution X'
        delta= fitness(X)-fitness(X')
        IF delta>0
           X=X'
        ELSE                                  > Acceptance
           r= random[0…1]                       Criterion
           m= exp(delta/temperature)
           IF (r<m)
                X=X'
        Apply LS Procedure to X
    Update Temperature:
        temperature = alpha*temperature
```

Initialisation of Parameters - The following parameters must be defined by the user: (1) *MI* = maximum number of iterations; (2) *ti* = initial temperature; (3) *it*= times for temperature; and (4) *alpha* = annealing temperature reduction factor.

Solution Generation - The initial solution can be created randomly or using a deterministic form.

Evaluation of Solutions - To evaluate how good a potential solution is we use a fitness function (see Eq. 6).

Initial Temperature - In our implementation the initial temperature is defined by the user.

Perturbation Method - The algorithm chooses a random terminal and assigns it to a random concentrator.

Local Search - The LS method is applied aiming at improving the current solution. The LS method is described in Section 3.1.

Acceptance Criterion - The algorithm uses the Metropolis [11] criterion. This criterion permits small uphill moves and rejects large uphill moves. This is to prevent the algorithm from becoming trapped in a local minimum.

The algorithm accepts the solutions, which decrease the fitness value (*delta>0*). Worse solutions are also accepted with a probability given by:

$$m = \exp(delta / T) \quad (7)$$

A random number *r* in the range *[0...1]* is generated and compared with the factor *m*. Since *delta/T* is negative, the value of *exp* is calculated raised to a negative power so that *m* will also be in the range *[0...1]*. The more negative the power, the closer to *0* the value of *m*. Large negative values of *delta* and decreasing values for temperature make acceptance of an uphill move more unlikely. If the temperature is zero then only better moves will be accepted which effectively makes SA act like hill climbing.

Update Temperature - A certain number of iterations are carried out at each temperature and then the temperature is decreased. As the temperature of the system decreases the probability of accepting a worse move is decreased. This is the same as gradually moving to a frozen state in physical annealing.

Termination criterion - SA stops when a maximum number of iterations (*MI*) is reached.

4. GENETIC ALGORITHM

GA involves a search from a "population" of individuals (potential solutions), in order to find the optimal solution in the problem solution space [7]. 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 population for inclusion in the next population. Before making it into the next generation, 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 [7]. 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:

GA Algorithm

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

Initialisation of Parameters - The following parameters, must be defined by the user: (1) *MI* = maximum number of iterations; (2) *ni* = population size; (3) *pm* = mutation probability; and (4) *pcr* = crossover probability.

Initial Population - The initial population can be created randomly or in a deterministic form.

Evaluation of solutions - To evaluate the solutions' quality we use the fitness function described in Section 2 (see Eq. 6).

Selection - The selection operator chooses individuals from the current population for inclusion in the next population. We implement the Tournament selection operator. Tournament operator selects *d* individuals to produce a tournament subset of individuals. The best individual in this subset is then chosen as the selected individual. In our implementation, this operator randomly selects a subset of individuals (*d = 4*) and the best individual is selected to form the next population.

Before making it into the next population and depending upon the probability of crossover and mutation the selected individuals may undergo crossover and/or mutation.

Crossover - GA uses the "One point" crossover operator. In "One point" one crossover point is selected, the genes from beginning of chromosome to the crossover point are copied from one parent, the rest are copied from the second parent.

Mutation - GA uses the "change order" mutation operator. In "change order", two genes are randomly selected and exchanged.

Termination criterion - GA stops when a maximum number of iterations (*MI*) is reached.

5. RESULTS

In order to test the performance of ILS, SA, TS and GA, we use a collection of instances of different sizes. We selected 9 instances from literature [2] and we also used 3 randomly generated large instances with *1000* terminals and *300* concentrators.

Table 1 shows our choice of parameter values for ILS, TS, SA and GA.

Table 1. Parameter values.

ILS	Number of perturbations	[1 … 3]
SA	Number of iterations for temperature Initial temperature Alpha	100 10 0.95
TS	Number of elements in the tabu list	[5 … 20]
GA	Number of individuals Crossover probability Mutation probability	200 {0.3, 0.4} [0.6 … 0.8]

We run each algorithm *100* times for each test instance. The values presented in Table 2 were computed based on the *50* best executions. We record the best value (*Best*), average value (*AvgF*), standard deviation value (*Std*) and average computation time (*AT*) produced by each algorithm (see Table 2). The algorithms were executed using a processor Intel Quad Core with 2.84 GHz and 3.5GB of RAM. The initial solutions were created using the Greedy algorithm. To compute the results in Table 2, we use *300* iterations/generations for instances 1-4, *500* for the instance 5, *1000* for the instance 6, *1500* for the instance 7, *2000* for instances 8-9, and *10000* for instances 10-12. We establish the number of iterations based on preliminary observations on the convergence of the algorithms.

Table 2. Results.

P	ILS				SA				GA				TS			
	Best	*AvgF*	*Std*	*AT*	*Best*	*AvgF*	*Std*	*AT*	*Best*	*AvgF*	*Std*	*AT*	*Best*	*AvgF*	*Std*	*AT*
1	65.63	65.63	0.00	<1s	65.63	65.63	0.00	<1s	65.63	65.63	0.00	<1s	65.63	65.63	0.00	<1s
2	134.65	134.65	0.00	<1s	134.65	134.65	0.00	<1s	134.65	134.65	0.00	<1s	134.65	134.65	0.00	<1s
3	270.41	271.24	0.45	<1s	270.26	270.26	0.00	<1s	272.36	283.13	5.62	<1s	270.26	270.48	0.15	<1s
4	286.89	287.77	0.73	<1s	286.89	286.97	0.12	<1s	290.37	295.08	1.42	<1s	586.89	287.93	0.75	<1s
5	335.09	335.89	0.64	<1s	335.09	335.09	0.00	<1s	341.59	350.69	2.67	2s	335.09	336.00	0.66	<1s
6	371.43	372.30	0.42	<1s	371.21	371.49	0.17	<1s	382.69	388.21	1.80	3s	371.12	372.35	0.51	<1s
7	401.52	402.78	0.64	1s	401.71	403.00	0.69	1s	426.72	441.56	3.51	7s	401.49	403.29	0.76	1s
8	563.19	565.40	0.76	2s	563.50	564.35	0.42	2s	615.99	623.16	2.86	12s	563.34	564.34	0.59	2s
9	647.26	649.25	0.85	2s	643.13	644.14	0.39	2s	777.11	784.68	2.91	12s	642.86	644.04	0.53	2s
10	5031.56	5047.03	5.90	10s	5392.59	5393.68	1.40	10s	6824.73	6891.50	28.1	70s	5067.64	5090.97	7.64	10s
11	5043.58	5068.04	7.44	10s	5393.83	5396.84	0.90	10s	6645.31	6736.08	33.37	70s	5087.33	5105.23	8.08	10s
12	5009.13	5025.48	6.21	10s	5380.58	5383.22	0.84	10s	6593.29	6673.56	33.21	70s	5058.35	5075.97	6.48	10s

All algorithms reach feasible solutions for all test instances. SA algorithm is fast and can find good solutions in a reasonable running time for the smallest instances. GA presents a poor performance and is the slowest algorithm. LS algorithms present similar average computation times.

We must observe the average quality of the produced solutions and the standard deviations to establish which is the best algorithm. As it can be seen in table 2, the standard deviations for SA are smaller. However, for the harder instances (7-12), ILS and TS are the best algorithms. ILS and TS can find better solutions for larger instances.

6. CONCLUSION

In this paper, we explore Genetic and Local Search algorithms to optimise large balanced communication networks. Our purpose is to minimise the link cost to form a balanced communication network. Genetic Algorithm, Iterated Local Search, Tabu Search and Simulated Annealing were tested in small, medium and large communication networks. The results show that ILS and TS have a better performance for larger instances.

This paper shows that GA, ILS, TS and SA are able to deal with terminal assignment instances. All algorithms can find feasible solutions. The proposed algorithms are easy to apply and we suggest the application of them to other assignment problems.

GA, ILS, TS and SA can be modified to solve multi-objective optimisation problems. The application of these algorithms in combination with other algorithms may also form an exciting area for further research.

7. REFERENCES

[1] Abuali, F., Schoenefeld, D. and Wainwright, R. 1994. Terminal assignment in a Communications Network Using Genetic Algorithms. In *Proceedings of the 22nd Annual ACM Computer Science Conference* (Phoenix, Arizona, United States, 1994). ACM, New York, NY, 74–81.

[2] Bernardino, E., Bernardino, A., Sánchez-Pérez, J., Gómez-Pulido, J. and Vega-Rodríguez, M. 2008. Tabu Search vs Hybrid Genetic Algorithm to solve the terminal assignment problem. In *Proceedings of the IADIS International Conference Applied Computing* (Algarve, Portugal, April 10-13, 2008). IADIS Press, 404-409.

[3] Cerny, V. 1985. A Thermodynamical Approach to the Travelling Salesman Problem: an efficient Simulation Algorithm. *J. Optimization Theory and Applications*. 45, 1 (Jan. 1985), 41-51.

[4] Gen, M. and Chung, R. 1999. *Genetic algorithms and engineering optimization*. Wiley, New York.

[5] Glover, F. 1986. Future paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*. 13, 5 (May 1986), 533-549.

[6] Glover, F. and Laguna, M. 1997. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

[7] Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA.

[8] Khuri, S. and Chiu, T. 1997. Heuristic Algorithms for the Terminal Assignment Problem. In *Proceedings of the1997 ACM Symposium on Applied Computing* (San Jose, California, USA, 1997). ACM, New York, NY, 247-251.

[9] Kirkpatrick, S. Gelatt, C. D. and Vecchi, M. P. 1983. Optimization by Simulated Annealing. *J. Science*. 220, 4598 (May 1983), 671-680.

[10] Lourenço, H. R., Martin, O. and Stutzle, T. 2003. Iterated local search. In *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Kluwer Academic Publishers, Boston, MA, USA, 321–353.

[11] Metropolis, N. Rosenbluth, A. W. Rosenbluth, M. N. Teller, A. H. and Teller, E. 1953. Equations of State Calculations by Fast Computing Machines. *J. Chemical Physics*. 21, 6 (June 1953), 1087-1092.

[12] Salcedo-Sanz, S. and Yao, X. 2004. A hybrid Hopfield network-genetic algorithm approach for the terminal assignment problem. *IEEE Transaction On Systems, Man and Cybernetics*. 34, 6 (Dec. 2004), 2343-2353.

[13] Xu, Y., Salcedo-Sanz, S. and Yao, X. 2004. Non-standard cost terminal assignment problems using tabu search approach. In *IEEE Congress on Evolutionary Computation*, (June 19 - 23, 2004). 2302-2306.

[14] Yao, X., Wang, F., Padmanabhan, K. and Salcedo-Sanz, S. 2005. Hybrid evolutionary approaches to terminal assignment in communications networks. In *Recent Advances in Memetic Algorithms and related search technologies*, W. Hart, J. Smith and N. Krasnogor, Eds. Studies in Fuzziness and Soft Computing. Springer, Berlin / Heidelberg, 129-159.