

Assignment of cells to switches in a cellular mobile network using a hybrid Hopfield network-genetic algorithm approach[☆]

Sancho Salcedo-Sanz^{a,*}, Xin Yao^b

^aDepartment of Signal Theory and Communications, Universidad de Alcalá, 28871 Alcalá de Henares, Madrid, Spain

^bCentre for Research in Computational Intelligence and Applications (CERCIA), School of Computer Science, The University of Birmingham and Nature Inspired Computation and Applications Laboratory (NICAL), The University of Science and Technology of China, Hefei, Anhui 230027, PR China

Received 16 January 2006; received in revised form 18 December 2006; accepted 9 January 2007

Available online 7 February 2007

Abstract

Handoff and cabling cost management plays a key role in the design of cellular telecommunications networks. The efficient assignment of cells to switches in this type of networks is an NP-complete problem which cannot be solved efficiently unless $P = NP$. This paper presents a hybrid Hopfield network-genetic algorithm approach to the cell-to-switches assignment problem, in which a Hopfield network manages the problem's constraints, and a genetic algorithm searches for high quality solutions with the minimum possible cost in terms of handoff and cable displayed. We show, by means of computational experiments, the good performance of our approach to this problem.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Cellular networks; Cell-to-switch assignment; Hopfield neural networks; Genetic algorithms

1. Introduction

In a cellular mobile telecommunications network, the area of coverage is often divided in hexagonal *cells*, normally hierarchically set to reduce link costs. The cells usually communicate through stationary base stations to *switches*, which route calls either to another base station or to a public switched telephone network [1,2,15,22].

Due to users' mobility, the signals between the mobile unit and the base station may become weaker while interference from neighboring cells increases [2,21]. *Handoffs* are used to avoid problems related to weak signals and interference during a call. A handoff occurs when mobile networks automatically transfer a call from a radio channel in one base station to another radio channel in an adjacent base station, as the user crosses into the adjacent base station's cell area.

If a handoff occurs between two cells associated to the same switch (cells A and B in Fig. 1 for example) it is called *simple*

handoff. This type of handoff is relatively easy to perform, and does not involve any location update in the databases that record the position of the user [1,3]. Imagine now that a user moves from cell B to cell C. In this case, the process of handoff involves the execution of a complicated protocol between switches 1 and 2 (see Fig. 1). It involves, in addition, the updating of the location of the user in the position databases, and also, if the original switch (2 in this case) is the responsible for maintaining the billing information, it may not be able to remove itself from the connection. Note that the handoff process in the case of cells connected to different switches consume much more network resources than handoffs between cells connected to the same switch. Consequently, it is usually advantageous to connect cells which have a high frequency of handoffs between them to the same switch.

The cell-to-switch assignment problem (CTSAP) was introduced in [4], and further studied in [1]. These papers present some heuristic algorithms to solve the problem. Several other heuristics techniques have been applied to the CTSAP, such as tabu search [3,5], simulated annealing [2], genetic algorithms [6] and hybrid algorithms which mix genetic algorithms and local search heuristics [7,8].

In this paper we present a novel approach to the CTSAP based on a hybrid Hopfield network-genetic algorithm scheme. In our approach, the problem's constraints are managed by the

[☆] This paper has been partially supported by Comunidad de Madrid and Universidad de Alcalá through the Research Projects UAH PI2005/078 and CAM-UAH 2005/019. Xin Yao's work was partially supported by a National Natural Science Foundation of China grant (No. 60428202).

* Corresponding author. Tel.: +34 91 885 67 31; fax: +34 91 885 66 99.

E-mail address: sancho.salcedo@uah.es (S. Salcedo-Sanz).

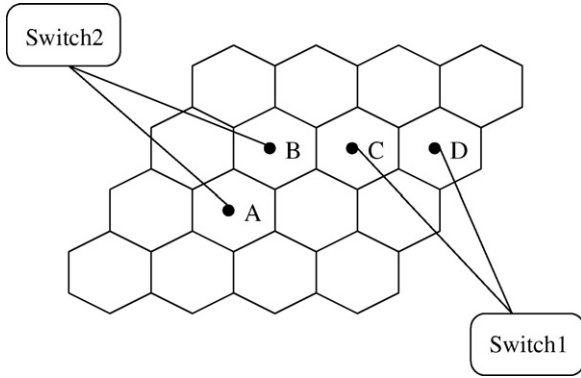


Fig. 1. An example of a simple handoff (from A to B or from C to D) and a complex handoff (from B to C).

Hopfield neural network (HNN), whereas the quality of the solution obtained is improved by a genetic algorithm (GA). The novelty of our approach is in the mix of a binary Hopfield network and a GA to form the hybrid HNN-GA. We propose two different GAs to be hybridized with the Hopfield network; a GA with binary encoding and an GA with integer encoding. The performance of this hybrid scheme, using the two GAs, has been tested in several test problems and compared with an existing GA [17] which manages the problem's constraints by a penalty function, and with a memetic algorithm formed by a GA and a tabu search (TS) heuristic [7]. The effectiveness of our algorithm has been shown by means of comprehensive experiments in several instances of the CTSAP. The results obtained in these experiments show that our method is significantly better (statistically) than previous approaches to the CTSAP.

The rest of the paper is structured as follows: in the next section, we provide a formal definition of the CTSAP. Section 3 provides a detailed overview of the most important existing approaches, whereas in Section 4 we describe the proposed Hopfield network-genetic algorithm approach. In Section 5 we present our experimental results and analyze the performance our algorithm. Finally, Section 6 concludes the paper with some remarks.

2. Problem definition

Consider a cellular mobile network with n cells and m switches, all located in known coordinates. Let c_{ik} be the amortized cost of cabling per unit time between cell i and switch k . Following Merchant and Sengupta [1], a handoff cost h_{ij} is incurred per unit time when cell i and j are assigned to different switches, and no handoff cost is considered when they are assigned to the same switch. Let λ_i be the number of calls handled by cell i per unit time, and M_k be the call handling capacity of switch k .

The problem's variables are defined as follows: let \mathbf{X} be an $n \times m$ binary matrix, in which the element x_{ik} is 1 if cell i is assigned to switch k , and 0 otherwise; z_{ijk} is 1 if cells i and j are both assigned to switch k and 0 otherwise; y_{ij} is defined as 1 if cells i and j are assigned together to any switch and 0 otherwise. The CTSAP is then defined as follows:

find \mathbf{X} which minimizes

$$Z = \sum_{i=1}^n \sum_{k=1}^m c_{ik} x_{ik} + \sum_{i=1}^n \sum_{j=1}^n h_{ij} (1 - y_{ij}), \quad (1)$$

subject to:

$$\sum_{k=1}^m x_{ik} = 1 \quad \forall i, \quad (2)$$

$$\sum_{i=1}^n \lambda_i x_{ik} \leq M_k \quad \forall k, \quad (3)$$

$$z_{ijk} = x_{ik} x_{jk} \quad \forall i, j, k, \quad (4)$$

$$y_{ij} = \sum_{k=1}^m z_{ijk} \quad \forall i, j. \quad (5)$$

The objective function to be minimized (Eq. (1)) comprises the amortized cost of cabling between the cells and the switches and the handoff cost associated with handoffs between cells connected to different switches. Constraints 2 ensure that every cell is assigned to one and only one switch. Constraints 3 ensure that the switch capacities (M_k) are not exceeded. The nonlinear constraints 4 capture the appropriate values for z_{ijk} ($z_{ijk} = 1$ if and only if $x_{ik} = x_{jk} = 1$), whereas constraints 5 capture the correct values of variable y_{ij} ($y_{ij} = 1$ if $z_{ijk} = 1$ for any switch k).

We can obtain a more compact problem definition by substituting Eq. (4) in (5), to get the equivalent expression $y_{ij} = \sum_{k=1}^m x_{ik} x_{jk}$, which can be substituted into the objective function. The CTSAP can be stated then as:

find \mathbf{X} which minimizes

$$Z = \sum_{i=1}^n \sum_{k=1}^m c_{ik} x_{ik} + \sum_{i=1}^n \sum_{j=1}^n h_{ij} - \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^m h_{ij} x_{ik} x_{jk}, \quad (6)$$

subject to:

$$\sum_{i=1}^n x_{ik} = 1 \quad \forall i, \quad (7)$$

$$\sum_{i=1}^n \lambda_i x_{ik} \leq M_k \quad \forall k. \quad (8)$$

3. Previous approaches

The CTSAP was introduced by Merchant and Sengupta in [4]. The same authors studied the problem further in [1]. These papers presented a valid linear integer programming formulation for the CTSAP. In these articles is also suggested that exact approaches failed even with relatively small CTSAP instances, and a heuristic algorithm which provide better results than exact approaches for large CTSAP instances is proposed.

The formulation introduced in [1] was extended by Menon and Gupta in [2]. In this paper the authors also presented a hybrid heuristic which integrates ideas from

linear programming into a simulated annealing framework. The results in this work show the performance of the proposed heuristic against the linear programming relaxation based on lower bound.

There are other works about the CTSAP based on different heuristic techniques: in [6] Hedible and Pierre presented a GA for tackling the CTSAP, in [9] Hung and Song presented an approach based on scatter search, whereas Bhattacharjee et al. in [10] presented a comparative study of several heuristic techniques. In [3], Pierre and Houéto discussed an approach based on TS. In this paper the authors showed that their TS approach outperforms a simulated annealing (SA) approach for the CTSAP.

Recently, some interesting papers applying novel hybrid techniques to the CTSAP have been published: Quintero and Pierre [7,8], introduced a multi-population memetic algorithm which use a GA as a global search heuristic and a TS and a SA as local search procedures to get high quality solutions for the CTSAP. In their study, they provide a comparison of the memetic algorithms' performance GA-TS and GA-SA with the standard GA.

In [7] the authors also introduced a novel way of encoding the CTSAP in a GA. Instead of using an $n \times m$ binary matrix \mathbf{X} , the authors proposed to use an integer vector $\tilde{\mathbf{x}}$ for encoding the solution of the problem. Consider a CTSAP instance with n cells and m switches. The solution of the problem is then encoded as a $\tilde{\mathbf{x}}$ integer vector of length n , where elements \tilde{x}_i fulfil $1 \leq \tilde{x}_i \leq m$. This encoding provides a more compact way of representing the assignment of cells to switches, since the vector $\tilde{\mathbf{x}}$ has length n , whereas matrix \mathbf{X} requires a binary string $n \times m$. On the other hand, the encoding method presented in [7] involves integer numbers instead of binary ones, which may complicate the operators in a GA.

Transformation between \mathbf{X} and $\tilde{\mathbf{x}}$ is straightforward:

If we have a matrix \mathbf{X} , the corresponding vector $\tilde{\mathbf{x}}$ can be calculated as:

$$\tilde{x}_i = j \text{ if } x_{ij} = 1. \tag{9}$$

If we have a vector $\tilde{\mathbf{x}}$, the corresponding binary matrix \mathbf{X} can be calculated as:

$$x_{ij} = \begin{cases} 1, & \text{if } \tilde{x}_i = j, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

4. Our proposed approach

In this section we present our hybrid Hopfield neural network-genetic algorithm approach to the CTSAP. In the next subsections we describe the HNN and two GAs with different encoding methods.

4.1. The Hopfield neural network

We use an HNN as a local search algorithm for satisfying the CTSAP constraints. Our HNN belongs to a class of binary HNNs [11] where the neurons can only take values 1 or 0. The

dynamics of this network depends on a matrix C which defines the minimum distance in rows between two 1s in the network, and on the initial state of the neurons. See [11] for further details. The structure of the HNN can be described as a graph, where the set of vertices are the neurons, and the set of edges define the connections between the neurons. We map a neuron to an element in the solution matrix \mathbf{X} . In order to simplify our notations, we shall use matrix \mathbf{X} to denote the neurons in the HNN. The HNN dynamics can then be described in the following way: After a random initialization of every neuron with binary values, the HNN operates in the serial mode. This means that only one neuron is updated at a time. Denoting by $x_{ik}(t)$ the state of a neuron on time t , the updating rule is described by

$$x_{ik}(t) = \text{isgn} \left(\sum_{p=1}^n \sum_{q=\max(1, k-\hat{c}_{i,p}+1)}^{\min(m, k+\hat{c}_{i,p})} x_{pq} \right) \forall i, k, \tag{11}$$

where the isgn operator is defined by:

$$\text{isgn}(a) = \begin{cases} 0, & \text{if } a > 0 \text{ or } \sum_{i=1}^n \lambda_i x_{ik} > M_k \forall k, \\ 1, & \text{otherwise.} \end{cases}$$

Note that the updating rule only takes into account neurons x_{pq} with value 1 and within a distance of c_{ip} in columns of the element x_{ik} being updated [12,13]. The matrix \hat{C} is an $n \times n$ matrix which encodes the problem's constraints given by Eq. (7). Its elements are defined as follows:

$$\hat{c}_{ij} = \begin{cases} m & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Note that this matrix forces one and only one 1 per row, whereas there may be several 1s in the same column.

In the updating rule defined above, neurons x_{ik} are updated in their natural order, i.e., $i = 1, 2, \dots, n, k = 1, 2, \dots, m$. We introduce a modification of this rule by performing the updating in a random ordering of the rows (variable i). This way the variability in the feasible solution found is increased. Let $\pi(i)$ be a random permutation of $i = 1, 2, \dots, n$. The new updating rule of the HNN will be

$$x_{\pi(i)k}(t) = \text{isgn} \left(\sum_{p=1}^n \sum_{q=\max(1, k-\hat{c}_{\pi(i),p}+1)}^{\min(m, k+\hat{c}_{\pi(i),p})} x_{pq} \right) \forall i, k. \tag{13}$$

The resulting updating rule runs over the rows of \mathbf{X} in the order given by the permutation $\pi(i)$, but the columns are updated in the natural order $k = 1, 2, \dots, m$.

A *cycle* is defined as the set of $N \times M$ successive neuron updates in a given order. In a cycle, every neuron is updated once following the given order $\pi(i)$, which is fixed during the execution of the algorithm [13,14]. After every cycle, the convergence of the HNN is checked. The HNN is considered converged if none of the neurons have changed their state during the cycle. The final state of the HNN is a potential solution to the CTSAP, which fulfils the problem's constraints

given by Eqs. (7) and (8). However, the solution found may be infeasible if not all the cells are assigned to a switch.

4.2. Example of the HNN convergence in the CTSAP

A small example of the HNN convergence can help to understand its performance. Consider a CTSAP instance with 6 cells and 2 switches. We define $\lambda_i = [0.3, 0.8, 0.2, 0.1, 0.9, 1.6]$ and $M_k = [2.5, 2.5]$, note that $\sum_i \lambda_i = 3.9$ and $\sum_k M_k = 5$, and the instance is therefore solvable. The constraints matrix associated to this problem is given by Eq. (12), which in this case takes the following form:

$$\hat{c}_{ij} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix} \quad (14)$$

This matrix forces one 1 per row in matrix X since the number of columns is 2 (number of switches), and the minimum columns separation between 1s in matrix X is also 2, so only one 1 can be located in a given row.

We will show that the HNN is able to obtain feasible solutions for the problem (the optimality of the solution would depend on cells and switches positions, and on hand off matrix h_{ij} , not considered in this example). Fig. 2 shows the dynamics of the neural network until the convergence. The first initial solution has been obtained at random.¹ Note that this initial solution is not feasible, since cell #2 is assigned to both switches. Note also that $\sum_{i=1}^n \lambda_i x_{i2} = 1.2 < M_1$ (the capacity constraint is fulfilled in switch #1) but $\sum_{i=1}^n \lambda_i x_{i2} = 2.6 > M_2$ (the capacity constraint is not fulfilled in switch #2).

Let us consider a natural order of neurons updating, so the first neuron to be updated is x_{11} , which state does not change, and then neuron x_{12} which state is changed from 1 to 0 in order to avoid constraint of assignment to more than one switch. Neuron x_{15} also changes its state from 0 to 1, since there are not constraints of capacity or assignment to more than one switch in this neuron. In a second step, the neurons of the second row are updated, and in this case neuron x_{22} changes its state from 1 to 0 due to the capacity constraint. The rest of the neurons remain unchanged. In a final step, neuron x_{12} is updated from 0 to 1 since no constraints affect it, and thus a final feasible solution is obtained. This solution fulfills constraint Eq. (7), each cell is assigned to one and only one switch, and also constraint Eq. (7), capacity constraint, since $\sum_{i=1}^n \lambda_i x_{i1} = 2.1 < M_1$ and $\sum_{i=1}^n \lambda_i x_{i2} = 2.4 < M_2$.

4.3. Genetic algorithm I

The first GA we propose (GA I hereafter) to be hybridized with the HNN, encodes a population of potential solutions for

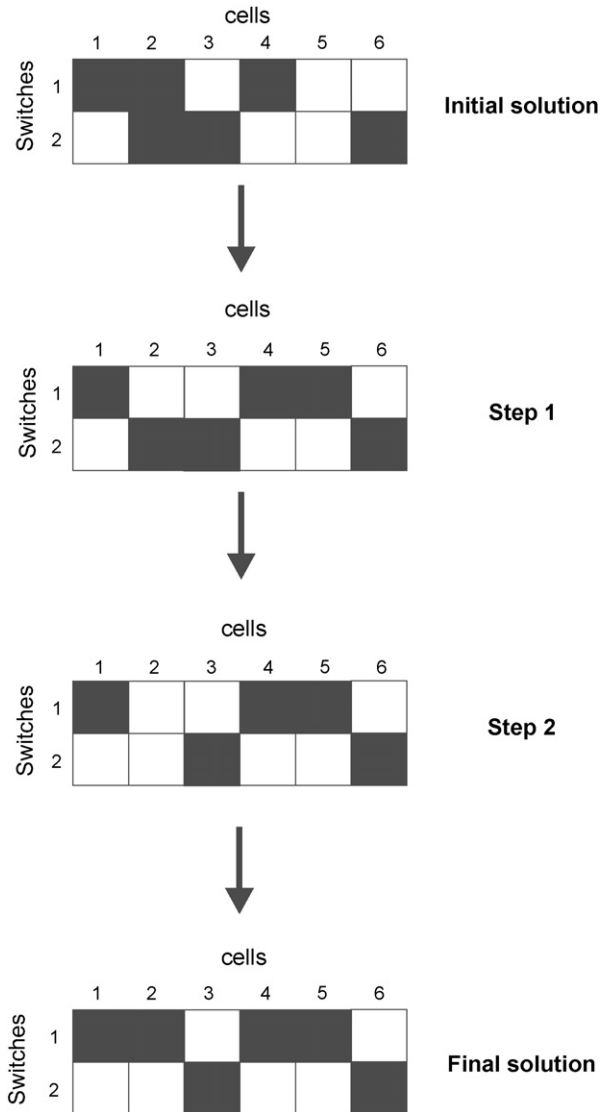


Fig. 2. Example of the binary Hopfield neural network’s dynamics in a 2 switches 6 cells instance.

the CTSAP as binary strings of length $n \times m$. Each string represents a different matrix \mathbf{X} , i.e., this GA uses the notation given in the definition of the problem (see Section 2). Fig. 3(a) shows an example of this encoding for a problem with 10 cells and 4 switches. The population is then evolved through successive generations using selection, crossover and mutation operators [16]. The selection operator is the roulette wheel selection, in which individuals are randomly sampled with probabilities inversely proportional to their fitness values (in order to minimize the objective function). In this case, values of fitness are given by the problem’s cost function Z in Eq. (6). An elitist strategy, which always passes the highest fitness string to the next generation, is applied in order to preserve the best solution encountered thus far in the evolution. We consider two-point crossover with probability P_c and random flip mutation with probability P_m . Since crossover and mutation operators may cause the offspring strings to be infeasible, the offspring will be set into the HNN and replaced by its result. In the case

¹ Please note that all along the paper we depict matrix X^T (transpose) instead of X , for space reasons.

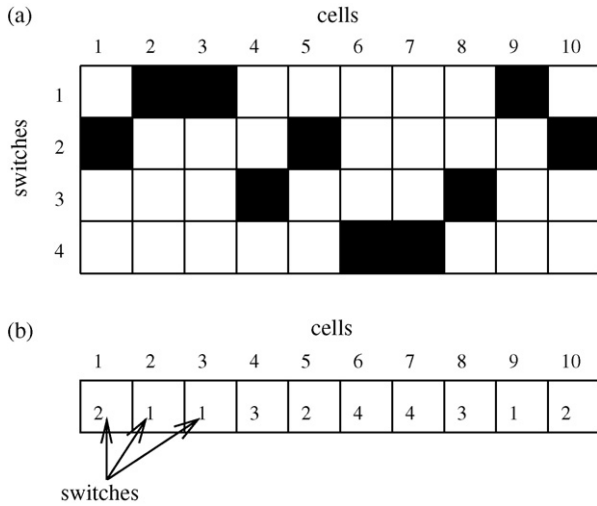


Fig. 3. Example of the two representations used: (a) Hybrid I. The black squares represent 1s and the white squares represent 0s; (b) Hybrid II.

that even the solution found by the HNN is infeasible (one or more cells not assigned to any switch), a penalty term proportional to the number of cells not assigned to switches is added to the cost function.

In pseudo-code, this GA for the CTSAP can be summarized as follows:

Genetic algorithm I

```

Initialize GA population at random
while(Stopping criteria are not satisfied) do
  for(every individual X)
    Run the HNN in order to obtain a feasible X.
    Calculate the fitness value (Z) of the individual.
    if X is not feasible, apply a penalty to Z.
    Substitute the GA individual by the new X obtained through the HNN.
  endfor
  selection
  crossover
  mutation
end(while)

```

4.4. Genetic algorithm II

The second GA (GA II) we investigate in this work uses the encoding method proposed in [7]: every individual is a string of n integers between 1 and m . An example of this encoding can be seen in Fig. 3(b) for an CTSAP formed by 10 cells and 4 switches. The selection and crossover are applied in the same way as that in GA I. The mutation operator consists of substituting every position in the string with a different integer value, with a very small probability P_m . In order to obtain feasible solutions, every individual in the population is passed to the HNN: First, the string of integers \tilde{x} is passed to a binary matrix X using Eq. (10); once a feasible solution has been obtained by the HNN and the fitness value calculated, the resulting matrix X matrix is passed back to a vector of integers \tilde{x} using Eq. (9). In the case that the solution found by the HNN is infeasible (one or more cells not assigned to any switch) the corresponding position in the string is filled at random, and a

penalty term similar to the one for GA I is added to the cost function.

In pseudo-code, this algorithm for the CTSAP can be summarized as:

Genetic algorithm II

```

Initialize GA population at random
while(The stopping criteria are not satisfied) do
  for(every individual x)
    x → X
    Run the HNN in order to obtain a feasible X.
    Calculate the fitness value (Z) of the individual.
    if X is not feasible, apply a penalty to Z.
    X → x
    Substitute the individual in the GA by x.
  end(for)
  selection
  crossover
  mutation
end while

```

5. Computational experiments

5.1. Generation of CTSAP test instances

In order to test the performance of our hybrid approaches, we have tackled some test problems of different sizes (see Table 1). Test problems were generated following the information in [1,7]. We consider an hexagonal grid such as in Fig. 1. We assume that the antenna for each cell is located in the center of the cell and switches are uniformly distributed randomly outside the grid (see Fig. 1). The cable cost between a switch and a cell antenna is taken to be proportional to the geometric distance between the two.

The rate of calls (λ_i) originated in each cell was generated from a Gamma distribution with mean one and coefficient of variation 0.25. The holding time of a call within a cell is distributed according to an exponential law of mean 1. If a cell has k neighbors, then we divide the range [0,1] into $k+1$ intervals by selecting k random numbers from a uniform distribution between 0 and 1.

At the end of the service period in a cell j , the call could be either transferred to the i th neighbor ($i = 1, \dots, k$) with a handoff probability r_{ij} equal to the length of i th interval, or ended with a probability equal to the length of the $k+1$ th interval. The handoff rate h_{ij} is defined as:

$$h_{ij} = \lambda_i r_{ij}. \quad (15)$$

All the switches have the same capacity M_k calculated as:

$$M_k = \frac{1}{m} \left(1 + \frac{K}{100} \right) \sum_{i=1}^n \lambda_i, \quad (16)$$

where K is uniformly chosen between 10 and 50, which ensures a global excess of 10–50% of the switches' capacity compared to the cells' volume of calls [7,8].

All test instances were run in a simulation platform under UNIX system, with a Dual Xeon/2.8 GHz processor.

Table 1
Comparison of the results obtained by the different algorithms considered (best/average/standard deviation)

P #	C/S	GA	M	HI	HII
1	15/3	184.9/189.4/4.7	184.9/187.4/8.5	184.9/185.9/1.1	184.9/185.3/0.4
2	15/4	369.1/379.2/11.7	369.1/374.9/6.7	369.1/371.3/2.2	369.1/369.9/0.85
3	15/5	374.5/394.8/16.1	373.7/376.6/5.5	373.7/379.4/5.1	373.7/376.7/2.6
4	30/3	807.9/840.0/23.2	801.1/824.6/11.4	801.1/832.3/21.4	801.1/811.3/10.0
5	30/4	730.0/770.2/21.0	724.9/757.2/16.0	724.0/741.8/11.9	721.2/744.2/12.3
6	30/5	692.7/740.2/19.0	693.8/754.5/36.1	688.6/733.4/25.2	687.6/721.7/11.9
7	50/3	1421.3/1457.2/29.1	1419.3/1447.8/30.7	1422.2/1450.6/17.3	1418.7/1436.2/14.5
8	50/4	1145.2/1205.4/29.4	1158.2/1182.5/15.8	1152.4/1221.6/48.4	1139.7/1178.9/21.0
9	50/5	948.7/994.6/31.2	945.2/980.4/35.4	947.3/979.7/15.9	948.1/976.7/13.0
10	75/3	2466.0/2635.5/93.2	2439.0/2588.3/141.5	2451.8/2613.0/79.8	2437.9/2529.6/40.6
11	75/4	1791.7/1884.8/47.0	1774.8/1880.8/70.9	1770.3/1870.1/50.3	1767.9/1841.7/41.6
12	75/5	2931.6/3066.0/72.8	2881.5/3062.9/143.5	2925.3/3062.3/66.3	2874.4/3017.2/54.8
13	100/3	3002.3/3118.5/75.0	2914.3/3080.4/98.2	2969.0/3114.8/94.1	2913.8/3030.9/66.4
14	100/4	3263.2/3309.0/32.2	3239.0/3337.1/96.1	3250.6/3301.3/31.7	3238.7/3287.6/31.2
15	100/5	2817.1/2942.3/68.6	2814.6/2910.6/77.8	2732.1/2857.5/52.4	2712.7/2799.7/41.8

Symbols P #, C/S, GA, M, HI and HII stand for problem number, cells/switches of the instance, genetic algorithm, memetic algorithm, Hybrid I and Hybrid II algorithms, respectively.

5.2. Results

We compare the results obtained by our hybrid algorithms² with the results obtained by the GA with a penalty function [17], and with a *memetic* algorithm proposed in [7].

The use of penalty functions within GAs is a well known method for managing constrained combinatorial optimization problems [18]. We consider a GA with integer encoding as proposed in [7] (see Section 3). Using this encoding method will be no infeasible solutions due to unassigned cells to switches, but only due to an excessive load of switches. The penalty function used to cope with the problem's constraints is the sum of two parts. First, a constant term is added to the fitness function in such a way that the fitness value of the best infeasible solution is greater than the fitness value of the worse feasible solution. The second term of the penalty is proportional to the excessive load in every overloaded switch. The genetic operators used in this algorithm are the same as those used in the GA II (see Section 4.4).

The memetic algorithm we use for comparison has been described in [7]. A memetic algorithm is a hybrid algorithm formed by a population-based global search and a local search heuristic carried on for each of the individuals. In this case, it is formed by a GA as global search algorithm and a TS as a local search heuristic. TS is a meta-heuristic that guides the local exploration of a given search space by means of a "Tabu list", or forbidden moves to previous low-quality solutions, to avoid local optimum solutions [19,20]. A search begins with a feasible solution usually produced with another heuristic. For each feasible solution, a class of neighbor solutions is generated by a move or a perturbation. A move is generally obtained by changing some of the attributes of the current solution. In order to prevent the algorithm from deterministically cycle among solutions already visited, one or more of the attributes of the

current solution are kept in the tabu list as *forbidden moves*. The number of moves in the list is referred to as the *tabu length*, and the exclusion period that a move remains in tabu is called the *tabu tenure* of it. In the memetic algorithm we consider, the GA encodes each solution as an integer string, just like our GA II. Each individual is passed through a TS procedure, in which a move consists of the swapping between two assigned switches. A maximum number of swaps are allowed for each individual. The best solution found during the TS process is incorporated to the GA substituting the previous assignment. More details about memetic algorithms applied to the CTSAP are given in [7].

The parameters of all GAs compared are the following: population size of 50 individuals, $P_c = 0.6$, $P_m = 0.01$, number of generations equal to 1000, two-point crossover, and roulette wheel selection with probabilities of survival inversely proportional to the fitness function in order to minimize it. The parameters of the memetic algorithm used for comparison purposes are population size of 25 individuals, maximum number of swaps in the TS of 20 (with a tabu length equal to 7,

Table 2
 t values obtained by a two-tailed t -test for problems #1 to #15

Problem #	HII-GA t -test	HII-HI t -test	HII-M t -test
1	-4.70 [†]	-2.40 [†]	-1.28
2	-4.67 [†]	-2.93 [†]	-4.59 [†]
3	-6.22 [†]	-2.56 [†]	-0.88
4	-5.81 [†]	-5.35 [†]	-4.41 [†]
5	-5.79 [†]	-0.72	-3.32 [†]
6	-4.19 [†]	-2.71 [†]	-4.71 [†]
7	-3.70 [†]	-4.00 [†]	-1.77
8	-4.07 [†]	-4.2 [†]	-0.74
9	-2.57 [†]	-0.84	-0.53
10	-6.02 [†]	-5.40 [†]	-2.12 [†]
11	-4.26 [†]	-2.45 [†]	-2.42 [†]
12	-2.77 [†]	-2.59 [†]	-1.69
13	-5.44 [†]	-4.05 [†]	-2.04 [†]
14	-2.68 [†]	-2.53 [†]	-2.80 [†]
15	-9.22 [†]	-5.31 [†]	-7.91 [†]

² Hereafter we call Hybrid I to the HNN working with GA I and Hybrid II to the HNN working with GA II.

[†] Stands for values of t with 29 degrees of freedom which are significant at $\alpha = 0.05$.

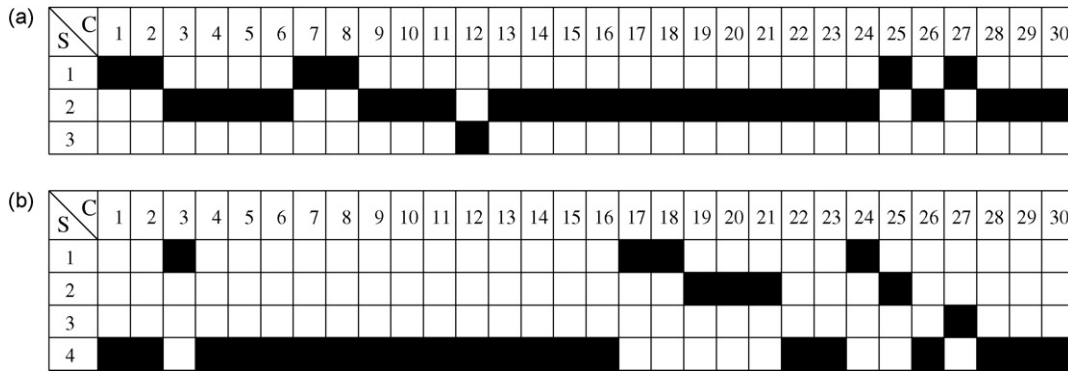


Fig. 4. Best assignments obtained by the Hopfield network-genetic algorithm approach in problems #4 (a) and #5 (b), respectively.

the standard value proposed by Glover in [19]), and a maximum of 100 generations. Note that with these parameters, the number of function evaluations is equal for all algorithms considered. We ran every algorithm 30 times, keeping the best, average and standard deviation values obtained.

Table 1 shows the results obtained by our approaches Hybrid I and II, compared with the GA with penalty function proposed in [17] and the memetic algorithm in [7]. Table 2 shows the results of a *t*-test performed over the data obtained by the compared algorithms. This *t*-test involves comparison of the Hybrid II algorithm with the rest of the approaches tested. In these tables it is possible to see that our Hybrid II algorithm outperforms the other algorithms considered. It performs better than the GA with penalty function in all the test problems, and also the improvements over the first approach proposed in this paper (Hybrid I) are clear. The comparison with the memetic algorithm proposed in [7] shows that our approach outperforms it in 9 out of 15 problems, and in the rest our Hybrid II obtains better results, but they are not statistically significant.

5.3. Discussion

Fig. 4 shows the best assignments obtained by the Hybrid II algorithm, for the problems #4 and #5. In order to further study the performance of this heuristic, Fig. 5 shows the cellular network and the assignment for problem #4 (Fig. 4(a)). The

Table 3
Values of λ_i for the example depicted in Fig. 5

Cell #	x_coord	y_coord	λ_i
1	0	0	0.33
2	2	0	0.66
3	4	0	0.50
4	6	0	0.03
5	8	0	0.19
6	10	0	1.15
7	1	1	0.53
8	3	1	2.04
9	5	1	1.37
10	7	1	0.22
11	9	1	0.40
12	11	1	4.30
13	2	2	0.58
14	4	2	0.79
15	6	2	0.10
16	8	2	1.26
17	10	2	2.73
18	12	2	0.74
19	3	3	0.02
20	5	3	0.08
21	7	3	0.58
22	9	3	0.43
23	11	3	0.26
24	13	3	2.24
25	4	4	6.83
26	6	4	0.61
27	8	4	4.98
28	10	4	0.79
29	12	4	1.63
30	14	4	0.24

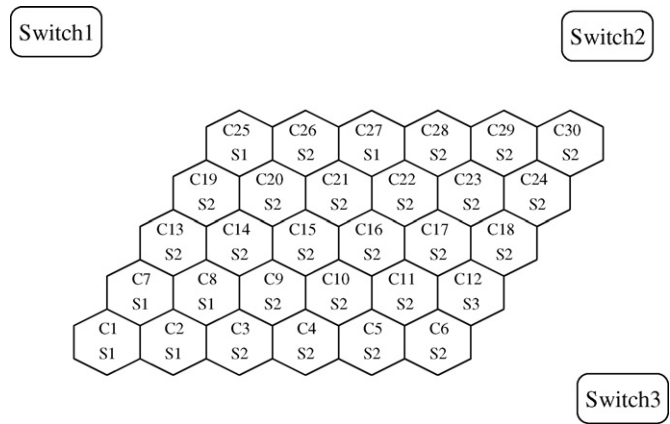


Fig. 5. Assignment obtained by the Hopfield network-genetic algorithm approach in problem #4 depicted together with the cellular network considered.

values of λ_i , M_k and the positions of the cells and switches for this problem are given in Tables 3 and 4. Note that the trend is to assign as many cells as possible³ to the nearest switch (switch 2 in this case). Cells in the center of the network, which have more probability of handoff than other cells, are always assigned to the same switch. In this example, switch 3 only handles one cell, cell 12, which is nearer to switch 3 than to switch 2. On the other hand, switch 1 handles six cells (cells 1, 2, 7, 8, 25 and 27). All these cells are closer to switch 2 than to switch 3 and are not in the center of the network.

³ Note that this depends on the capacity of switches.

Table 4
Values of λ_i for the example depicted in Fig. 5

Switch #	x_coord	y_coord	M_k
1	−3.63	7.41	17.21
2	15.97	5.84	17.21
3	19.02	−5.46	17.21

Table 5
Relative distance (in %) between algorithms' best solution found and the lower bound, for test problems #13, #14 and #15

Problem #	C	S	GA	M	HI	HII
13	100	3	11.1	8.5	10.1	8.5
14	100	4	9.1	8.5	8.7	8.5
15	100	5	10.6	10.6	8.0	6.9

The symbols C, S, GA, M, HI and HII stand for cells, switches, genetic algorithm, memetic algorithm, Hybrid I and Hybrid II algorithms, respectively.

Note that the solutions provided by our approach do not necessarily correspond to the global optimum of the problem. In order to evaluate the quality of the solutions achieved by our approach, we compare them with a lower bound for the CTSAPs. An intuitive lower bound is defined in [7] as:

$$LB = \sum_{i=1}^n \min_k (c_{ik}) \quad (17)$$

This is the cabling cost of the solution obtained by assigning each cell i to the nearest switch k . It is supposed that the capacity of switches is infinity and all cells can be assigned to a single switch. Note that we have a lower bound whatever the values of M_k and λ_k . LB does not include handoff cost, and therefore no solution could equal it. However, we look for solutions close to this LB, as a measure of quality.

Table 5 shows the relative distance (in %) between the best solutions obtained by the considered algorithms and LB, for the CTSAP test instances involving 100 cells. All the analyzed algorithms provide solutions within 10% of the LB, and the Hybrid II algorithm is able to achieve solutions around 8% of the LB.

Table 6
Computational time (in s) of the algorithms tested

Problem #	GA	Memetic	Hybrid I	Hybrid II
1	2	2	5	5
2	2	3	5	5
3	2	3	5	5
4	5	7	15	15
5	5	7	15	15
6	5	7	15	15
7	15	16	20	20
8	15	18	20	20
9	15	18	25	25
10	20	22	30	30
11	20	25	30	30
12	20	25	35	35
13	25	35	40	40
14	25	35	40	40
15	25	37	45	45

The computational time of the algorithms tested in this paper is given in Table 6. Round figures are provided, only for given an idea of the algorithms' running times. The GA with penalty function is faster than Hybrid I, Hybrid II and memetic schemas, as expected. The memetic algorithm tested is slightly faster than the hybrid approaches of this paper. Note that there are not differences in computational time between our approaches Hybrid I and Hybrid II. This shows that the increase of computational cost in our algorithm is mainly due to the HNN. Nevertheless, the computational cost of the proposed approaches is good, solving all test problems in less than 1 min.

6. Conclusions

In this paper we have proposed a novel approach for assigning cells to switches in a mobile telecommunications network. Our algorithm hybridizes a binary Hopfield neural network (HNN) with a genetic algorithm (GA). The HNN solves the problem's constraints, giving feasible solutions to the problem, and the GA performs a global search for high quality feasible solutions in terms of handoff and cabling costs. The novelty of our approach is the hybridization of the HNN and the GA to form the hybrid HNN-GA algorithm. Two GAs with different encoding have been proposed to be mixed with the HNN.

We have tested our approach on a set of CTSAP assignment instances and compared our results with those obtained by a GA with a penalty function and a memetic algorithm formed by a GA and a tabu search heuristic. We have shown, by means of these computational experiments, that our algorithm is able to achieve very good results in terms of the number of feasible solutions provided and minimum cost found in all test instances considered, and improving the results obtained by the other considered techniques.

References

- [1] A. Merchant, B. Sengupta, Assignment of cells to switches in PCS networks, *IEEE/ACM Trans. Networking* 3 (5) (1995) 521–526.
- [2] S. Menon, R. Gupta, Assigning cells to switches in cellular networks by incorporating a pricing mechanism into simulated annealing, *IEEE Trans. Syst. Man Cybern. B* 34 (1) (2004) 558–565.
- [3] S. Pierre, F. Houéto, Assigning cells to switches in cellular mobile networks using taboo search, *IEEE Trans. Syst. Man Cybern. B* 32 (3) (2002) 351–357.
- [4] A. Merchant, B. Sengupta, Multiway graph partitioning with applications to PCS, in: *Proceedings of the IEEE INFOCOM'94*, vol. 2, Toronto, ON, Canada, (June 1994), pp. 593–600.
- [5] F. Houéto, S. Pierre, A tabu search approach for assigning cells to switches in cellular mobile networks, *Comput. Commun.* 25 (3) (2002) 464–477.
- [6] C. Hedible, S. Pierre, Algorithme génétique pour l'affectation de cellules à des commutateurs dans les réseaux de communications personnelles, in: *Proceedings of the IEEE Canadian Conf. Elect. Comput. Eng.*, Nova Scotia, Halifax, Canada, (May 2000), pp. 1077–1081.
- [7] A. Quintero, S. Pierre, Evolutionary approach to optimize the assignment of cells to switches in personal communications networks, *Comput. Commun.* 26 (2003) 927–938.
- [8] A. Quintero, S. Pierre, Assigning cells to switches in cellular mobile networks: a comparative study, *Comput. Commun.* 26 (2003) 950–960.

- [9] W. Hung, X. Song, On optimal cell assignment in PCS networks, in: Proceedings of the IEEE Int. Perform., Comput., Commun. Conf., Phoenix, AZ, (April 2002), pp. 225–232.
- [10] P. Bhattacharjee, D. Saha, A. Mukherjee, Heuristics for assignment of cells to switches in a PCSN: a comparative study, in: Proceedings of the IEEE Int. Conf. Personal Wireless Commun., Jaipur, India, (February 1999), pp. 331–334.
- [11] Y. Shrivastava, S. Dasgupta, S.M. Reddy, Guaranteed convergence in a class of Hopfield networks, *IEEE Trans. Neural Networks* 3 (6) (1992) 951–961.
- [12] S. Salcedo-Sanz, C. Bousoño-Calzón, A.R. Figueiras-Vidal, A mixed neural-genetic algorithm for the broadcast scheduling problem, *IEEE Trans. Wireless Commun.* 2 (2) (2003) 277–283.
- [13] S. Salcedo-Sanz, R. Santiago-Mozos, C. Bousoño-Calzón, A hybrid Hopfield network-simulated annealing approach for frequency assignment in Satellite Communications systems, *IEEE Trans. Syst. Man Cybern. B* 34 (2) (2004) 1108–1116.
- [14] S. Salcedo-Sanz, X. Yao, A hybrid Hopfield network genetic algorithm approach for the terminal assignment problem, *IEEE Trans. Syst. Man Cybern.* 34 (6) (2004) 2343–2353.
- [15] X. Yao, F. Wang, K. Padmanabhan, S. Salcedo-Sanz, Hybrid evolutionary approaches to terminal assignment in communication networks, *Recent Adv. Memetic Algorithms* (2005) 129–159.
- [16] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [17] S. Khuri, T. Chiu, Heuristic algorithms for the terminal assignment problem, in: Proceedings of the ACM Symposium on Applied Computing, ACM press, 1997, pp. 247–251.
- [18] J. Richardson, M. Palmer, G. Liepins, M. Hilliard, Some guidelines for genetic algorithms with penalty functions, in: Proceedings of the 3rd Int. Conf. Genetic Algorithms and App., Morgan Kaufmann, California, 1989 .
- [19] F. Glover, Tabu search—part I, *ORSA J. Comput.* 1 (3) (1989) 190–206.
- [20] F. Glover, Tabu search—part II, *ORSA J. Comput.* 2 (1) (1990) 4–32.
- [21] A. Kershbaum, *Telecommunications Network Design Algorithms*, McGraw-Hill, 1993.
- [22] P. Cortés, J. Larrañeta, L. Onieva, J.M. García, M.S. Caraballo, Genetic algorithm for planning cable telecommunication networks, *Appl. Soft Comput.* 1 (1) (2001) 21–33.