ELSEVIER

Computer Communications 25 (2002) 464–477

computer communications

www.elsevier.com/locate/comcom

A tabu search approach for assigning cells to switches in cellular mobile networks

Samuel Pierre*,[1], Fabien Houéto

Department of Computer Engineering, Mobile Computing and Networking Research Laboratory (LARIM), École Polytechnique de Montréal, C.P. 6079, Succ. Centre-Ville, Montréal, Qué., Canada H3C 3A7

Received 11 January 2001; revised 24 April 2001; accepted 29 May 2001

Abstract

This paper proposes a tabu search approach for assigning cells to switches in wireless cellular networks. This problem is NP-hard and consequently cannot be practically solved by exact methods for real size networks. We first establish a new mathematical equivalence between the assignment problem and the well-known p-fixed hub location problem. From this equivalence as a basis for a heuristic method, initial solutions have been generated. Then, some flexible and powerful moves, a gain structure and its update procedures are defined and used to improve these initial solutions and generate near optimal final solutions. To evaluate the performance of this approach, we define two lower bounds for the global optimum, which are used as reference to assess the quality of the obtained solutions. Computational experiments show that our approach generally provides better results than those generated by other traditional heuristics found in the literature, especially for large-scale networks with more than 50 cells and 3 switches.

Keywords: Cellular networks; Mobile networks; Cell assignment; Tabu search; Heuristic methods

1. Introduction

In a cellular network, the area of coverage is often geographically divided in hexagonal cells. These cells are hierarchically set to reduce link costs, as illustrated in Fig. 1. A certain number of cells are chosen to install switches that communicate with one another and serve as relays for communications between any pair of cells. For various reasons and especially because of mobility, switches serving as relays to a given user could change if this latter moves from its current cell. The operation of detecting a user who has changed a cell and carrying out the required updates constitutes a handoff. When a handoff occurs between two cells linked to the same switch, it is called a simple handoff, because there are few necessary updates. On the other hand, a complex handoff describes a handoff between two cells related to different switches because, in this case, the update procedures consume more resources than in the case of a simple handoff.

Each cell uses an antenna for communications among users with pre-assigned frequencies. A switch relays communications among users of the same cell. For a user who moves, the signal sent is captured by the closest cells by taking into account a filtering threshold. For example, in Fig. 1, a user who moves from cell B to cell A causes a simple handoff. The network's database that keeps in memory the switch (managing each user) does not need an update. Switch 1 is used for this update and no other network entity intervenes. However, if a user moves from cell B to cell C, we are in the presence of a complex handoff. Actually, switches 1 and 2 have to exchange information on the user, and the database must also be updated. Furthermore, if Switch 1 is in charge of the billing, the handoff cannot simply replace Switch 1 with Switch 2. Communications between the two switches continue to be relayed through Switch 1 even after the handoff. Thus, we would have a user connection to Switch 2, then to Switch 1, and finally to the network. Therefore the cost of such a complex handoff is higher than that of a simple handoff.

If the handoff frequency between cell B and cell A (Fig. 1) is very high, while the handoff frequency between cell B and cell C is low, it is then reasonable to connect cells A and B to the same switch. Thus, the problem of cell assignment could be summarized as follows: for a set of cells and switches (whose positions are known), the problem consists of assigning cells to switches in a way that minimizes the cost function. The cost function integrates a link and a

* Corresponding author. Tel.: +1-514-340-4711, ext.: 4685; Fax: +1-514-340-3240.

E-mail address: samuel.pierre@polymtl.ca (S. Pierre).

[1] Supported by NSERC under Grant No. 140264-98.

0140-3664/02/$ - see front matter
PII: S0140-3664(01)00371-1

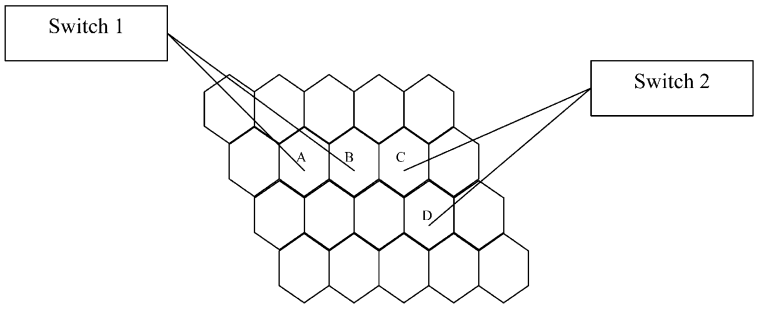

Fig. 1. Geographic division in a cellular network and handoff between switches.

handoff cost component. The assignment must take into account the switches' capacity constraints, which limit the number of calls a switch can control. In the real world, the assignment depends heavily on the geographical conditions and traffic loads. However, we do not consider in this case any geographical conditions and restrain ourselves to the traffic load constraints. Moreover, even if the problem is not a real-time problem, it cannot be solved exactly for many reasons. The first one is that the search for an exact solution is too much time-consuming. For a network with m switches and n cells, it should examine m^n solutions. With one hundred cells and two switches, if we take 1 ns (1000 MHz) per solution, the total running time is about 4×10^{13} years. The second reason is that, as the traffic grows, the operator will need to run again from time to time the assignment algorithm. Therefore, he could not afford prohibitive times. For these reasons, only heuristic approaches are used in the literature to solve this problem.

Merchant and Sengupta [17,18] studied this assignment problem. Their algorithm starts from an initial solution, which they attempt to improve through a series of greedy moves, while avoiding to be blocked in a local minimum. The moves used to escape a local minimum explore only a very limited set of options. These moves depend on the initial solution and do not necessarily lead to a good final solution.

This paper proposes a tabu search approach for assigning cells to switches in wireless cellular networks. Section 2 presents mathematical formulations of the assignment problem, the mathematical equivalence and a selected review of existing solving methods. Section 3 describes the proposed tabu search approach and presents some implementation details. Section 4 presents an analysis of results and compares them to other methods. Section 5 proposes two lower bounds for evaluating the quality of solutions provided by the proposed method.

2. Problem formulation and related work

The assignment problem consists in determining a cell assignment pattern to minimize the cost function, while respecting certain constraints, particularly those related to switches' capacity. In this section, we present a mathematical formulation of the problem, a new mathematical equivalence with the well-known p-fixed hubs location problem and a summary of related works.

2.1. Problem formulation

An assignment of cells to switches could be carried out according to a simple or a double homing of cells. A simple homing of cells refers to a situation where a cell can only be related to a single switch, while a double homing is when a cell could be related to at most two switches.

2.1.1. Simple homing

Let n be the number of cells to be assigned to m switches. The location of cells and switches is fixed and known. Let H_{ij} be the cost per unit of time for a simple handoff between cells i and j involving only one switch, and H'_{ij} the cost per time unit for a complex handoff between cells i and j ($i, j = 1, \ldots, n$ with $i \neq j$) involving two switches. H_{ij} and H'_{ij} are proportional to the handoff frequency between cells i and j. Let c_{ik} be the amortization cost of the link between cell i and switch k ($i = 1, \ldots, n$; $k = 1, \ldots, m$) and λ_i the number of calls per time unit destined to cell i. The capacity of a switch k is denoted M_k.

Let consider

$$x_{ik} = \begin{cases} 1 & \text{if cell } i \text{ is related to switch} \\ 0 & \text{if not} \end{cases}$$

The assignment of cells to switches is subject to a certain number of constraints. Actually, each cell must be assigned to only one switch, which is translated by the following formula:

$$\sum_{k=1}^{m} x_{ik} = 1 \quad \text{for } i = 1, \ldots, n. \tag{1}$$

Let z_{ijk} and y_{ij} be defined as:

$$z_{ijk} = x_{ik}x_{jk} \quad \text{for } i, j = 1, \ldots, n \text{ and } k = 1, \ldots, m, \text{ with } i \neq j.$$

$$y_{ij} = \sum_{k=1}^{m} z_{ijk} \quad \text{for } i, j = 1, \ldots, n, \quad \text{and} \quad i \neq j.$$

z_{ijk} is equal to 1 if cells i and j, with $i \neq j$, are both connected to the same switch k, otherwise z_{ijk} is equal to 0. y_{ij} takes the value 1 if cells i and j are both connected to the same switches and the value 0 if cells i and j are connected to different switches.

The cost per time unit f is expressed as follows:

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} H'_{ij}(1 - y_{ij}) + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} H_{ij}y_{ij} \tag{2}$$

The first term of the equation represents the link cost. The second term takes into account the complex handoffs cost and the third, the cost of simple handoffs. We should keep in mind that the cost function is quadratic in x_{ik}, because y_{ij} is a quadratic function of x_{ik}. Let's mention that an eventual weighting could be taken into account directly in the link and handoff costs definitions. If λ_i denotes the number of calls per unit of time destined to i, the limited capacity of switches imposes the following constraint:

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k \quad \text{for } k = 1, \ldots, m \tag{3}$$

according to which the total load of all cells, which are assigned to the switch k is less than the capacity M_k of the switch. Finally, the constraints of the problem are completed by:

$$x_{ik} = 0 \text{ or } 1 \quad \text{for } i = 1, \ldots, n \quad \text{and} \quad k = 1, \ldots, m. \tag{4}$$

$$z_{ijk} = x_{ij}x_{ik} \quad \text{and} \quad i, j = 1, \ldots, n \quad \text{and} \quad k = 1, \ldots, m. \tag{5}$$

$$y_{ij} = \sum_{k=1}^{m} z_{ijk} \quad \text{for } i, j = 1, \ldots, n \tag{6}$$

By adding, if necessary, fictive cells with non-null call volumes, we could bring constraint (3) to equality. Eqs. (1), (3) and (4) are then constraints of transport problems (these fictive cells will be discarded later). In fact, each cell i could be assimilated to a factory which produces a call volume λ_i. The switches are then considered as warehouses of capacity M_k where the cells production could be stored. Therefore, the problem is to minimize Eq. (2) under Eq. (1) and (3)–(6). When the problem is formulated in this way, it could not be solved with a standard method such as linear programming because constraint (5) is not linear. Merchant and Sengupta [17,18] replaced it by the following equivalent set of constraints:

$$z_{ijk} \leq x_{ik} \tag{7}$$

$$z_{ijk} \leq x_{jk} \tag{8}$$

$$z_{ijk} \geq x_{ik} + x_{jk} - 1 \tag{9}$$

$$z_{ijk} \geq 0 \tag{10}$$

Thus, the problem could be reformulated as follows; minimizing Eq. (2) under constraints (1), (3), (4) and (6)–(10). We can further simplify the problem by defining:

$$h_{ij} = H'_{ij} - H_{ij}.$$

h_{ij} refers to the reduced cost per time unit of a complex handoff between cells i and j. Relation (2) is then re-written as follows:

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} h_{ij}(1 - y_{ij}) + \underbrace{\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} H_{ij}}_{\text{constant}}$$

The assignment problem takes then the following form: Minimize:

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} h_{ij}(1 - y_{ij}) \tag{11}$$

subject to: Eqs. (1), (3), (4) and (7)–(10).

In this form, the assignment problem could be solved by usual programming methods.

2.1.2. Double homing

For a double homing, the call and handoff patterns do not remain the same during the day. We have, for example, two patterns for the daytime: one for the morning and another for the afternoon. The two problems, related to each part of the day, are not separated and cannot be solved independently. Optimization should not be performed separately for each pattern but simultaneously for both patterns, in a way that minimizes the total cost, especially the link cost.

Let λ_i and h_{ij} be respectively the number of calls per time unit assigned to cell i and the reduced cost per time unit of a complex handoff between cells i and j in pattern 1, the morning pattern. We will use λ'_i and h'_{ij} to denote the same notions in pattern 2. In the case of double homing, c_{ik} could be interpreted as the update cost required to assign cell i to switch k. M_k is still defined as the maximal capacity of switch k. The values M_k and c_{ik} remain the same in both patterns.

According to the principle of double homing, a cell could be connected to one or two switches, but in each pattern only one switch is active at a time. We must then find two assignment patterns for each part of the day, by indicating whether it is cost-effective to have a cell connected to two switches (with the cells shifting from one switch to the other) or to have the cell connected to only one switch. The difficulty in the case of double homing formulation lies in the fact that the link cost c_{ik} should not be accounted for twice, if cell i is connected to the same switch in both assignment patterns.

For pattern 1, let us keep the same definition as in the single homing case for variables x_{ik}, z_{ijk}, y_{ij}. These variables must, with variables λ_i and h_{ij}, satisfy constraints (1), (3), (4) and (6)–(10). For pattern 2, let us define equivalent variables x'_{ij}, z'_{ijk}, y'_{ij}, which with variables λ'_i and h'_{ij}, satisfy

the same constraints (1), (3), (4) and (6)–(10). The link cost between cell i and switch k is counted for if cell i is assigned to switch k in at least one of the two assignment patterns, i.e. if $x_{ik} = 1$ or $x'_{ik} = 1$. If w_{ik} is the new variable, which indicates whether we must count for the link cost between i and k, we have:

$w_{ik} = 1$ if $x_{ik} = 1$ or $x'_{ik} = 1$.

w_{ik} is defined as:

$$w_{ik} = x_{ik} \vee x'_{ik} \quad \text{for } i = 1,\ldots,n \text{ and } k = 1,\ldots,m \tag{12}$$

with ' $\vee$ ' referring to the logical operator 'or'.

The objective function is then:

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}w_{ik} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} h_{ij}(1 - y_{ij}) + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} h'_{ij}(1 - y'_{ij}) \tag{13}$$

under constraints (1), (3), (4) and (6)–(10), for variable x_{ik}, z_{ijk}, y_{ij}, λ_i and h_{ij} on one hand, and x'_{ik}, z'_{ijk}, y'_{ij}, λ'_i and h'_{ij} on the other. The additional constraints are:

$$w_{ik} = x_{ik} \vee x'_{ik} \quad \text{for } i = 1,\ldots,n \text{ and } k = 1,\ldots,m$$

The problem is now entirely defined, but it is non-linear. This is due to constraint (12), which could be replaced by the equivalent set of constraints:

$$w_{ik} \geq x_{ik} \tag{14}$$

$$w_{ik} \geq x'_{ik} \tag{15}$$

$$w_{ijk} \leq x_{ik} + x'_{ik} \tag{16}$$

$$w_{ik} \leq 1 \tag{17}$$

Hence, the following formulation: minimize Eq. (13) under constraints (1), (3), (4), (6)–(10) and (14)–(17), for variables x_{ik}, w_{ik}, z_{ijk}, y_{ij}, λ_i and h_{ij} on one hand, x'_{ik}, z'_{ijk}, y'_{ij}, λ'_i and h'_{ij} on the other.

At this point, let's mention that for both single and double homing, even if we consider c_{ik} only as a link cost, it's easy to interpret it also as a trunking cost. Actually in a cellular network, cells are linked to switches by radio links. Therefore, the trunking cost c_{ik} is a step function constant by given range intervals (for all the cells within a certain range of the switch, the trunking cost is the same (the cost of the end equipment) and doesn't depend on the distance). For the cells out of the range of the switch, the trunking cost c_{ik} is set to an arbitrary large value. More generally, a switch k can be made unavailable for a cell i by setting c_{ik} to an arbitrary large value.

2.2. Equivalence with the p-fixed hubs location problem

The assignment problem may be reduced to a certain number of well-known operational research problems, such as graph partitioning or p-fixed hubs location problem. Our analysis focuses on the single homing assignment problem and the p-fixed hubs location problem. For further details on the graph partitioning, see Refs. [17,18]. The formulation of the p-fixed hubs location problem, which corresponds the best to the assignment problem, is known as the p-fixed hubs location problem and was introduced by Skorin-Kapov et al. [23] and Sohn and Park [24].

Let's define H as a set of known and fixed positions of hubs with $|H| = m$ and c_{ik} as the fixed cost to connect hub k to cell i. The sets H of hubs and N of cells are disjoint. The problem is then stated as follows:

Minimize:

$$f_L = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{m}\sum_{p=1}^{m} v_{ij}A_{ijkp}x_{ijkp} + \sum_{i=1,i\notin H}^{n}\sum_{k=1}^{m} c_{ik}z_{ik} \tag{18a}$$

subject to:

$$z_{kk} = 1 \quad \text{for } k = 1,\ldots,m \tag{19}$$

$$\sum_{k=1}^{m} z_{ik} = 1 \quad \text{for } i = 1,\ldots,n \tag{20}$$

$$\sum_{p=1}^{m} x_{ijkp} = z_{ik} \quad \text{for } i = 1,\ldots,n-1 \quad \text{and} \quad j = i+1,\ldots,n,\ k = 1,\ldots,m \tag{21}$$

$$\sum_{k=1}^{m} x_{ijkp} = z_{jp} \quad \text{for } i = 1,\ldots,n-1 \quad \text{and} \quad j = i+1,\ldots,n,\ p = 1,\ldots,m \tag{22}$$

$$z_{ik} = 0 \text{ or } 1 \quad \text{for } i = 1,\ldots,n \quad \text{and} \quad k = 1,\ldots,m \tag{23}$$

$$x_{ijkp} \geq 0, \quad \text{for } i = 1,\ldots,n-1,\ j = i+1,\ldots,m \text{ and } p = 1,\ldots,m \tag{24}$$

where:

$v_{ij} = W_{ij} + W_{ji}$ with W_{ij} representing the flow from node i to node j;
$A_{ijkp} = a_{ik} + \alpha a_{kp} + a_{pj}$ with a_{ij}, the cost per flow unit between nodes i and j;
$\alpha \leq 1$ is a factor on the cost per flow unit between hubs (we suppose that $c_{ii} = 0$ for $i = 1,\ldots,n$).
x_{ijkp} is the portion of the flow between nodes i and j routed through the hubs located at nodes k and p (i is connected to k and j to p).
$z_{ik} = 1$ if node i is assigned to hub k and 0 if not.

The fact that the nodes of H are hubs results in constraint (19), while constraint (20) reflects that each node is assigned to one hub. Constraints (21) and (22) imply that the flow

coming to a node is totally forwarded (i.e. no node is source or sink for the flow).

The p-fixed hubs location problem, as introduced here, was relatively little studied. In the literature, it is generally split into two sub-problems: (i) localize the hubs, (ii) then assign nodes to hubs (Abdinnour-Helm [1], Cox [4], Klincewicz [15], Skorin-Kapov et al. [23], Sohn and Park [24]). In this paper, we are interested only in the assignment part.

In our previously introduced assignment problem, switches are equivalent to hubs and cells to nodes. We have $H \cap N = \varnothing$, where H is the set of switches and N the set of nodes. Moreover, $W_{ii} = 0$ for $i = 1, \ldots, n$, which means that there's no flow from cell i to itself. Let's suppose that $A_{ijkp} = A_{jipk}$ for $i, j = 1, \ldots, n$ and $k, p = 1, \ldots, m$ (which means that the handoff cost between cells i and j is the same as the handoff cost between cells j and i). Therefore, Eq. (18a) can be re-written:

$$f_L = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{m}\sum_{p=1}^{m} W_{ij}A_{ijkp}x_{ijkp} + \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}z_{ik} \qquad (18b)$$

In the context of the assignment problem, the cost per flow unit between nodes i and j, A_{ijkp} will be the cost per time unit of the handoff between i and j. Let's then define:

$$A_{ijkp} = \begin{cases} B_{ij} = a_{ik} + \alpha a_{kp} + a_{pj} \\ \quad \text{for a complex handoff between cells } i \text{ and } j \\ B_{ij} = a_{ik} + a_{pj} \\ \quad \text{for a simple handoff between cells } i \text{ and } j \end{cases}$$

In the case of the assignment problem, the flow (handoff) between cells i and j is wholly forwarded through switches k and p (consequently we have no fractional flow). Hence,

$$x_{ijkp} = x_{ik}x_{jp}, \quad z_{ik} = x_{ik}$$

and we can write:

$$\sum_{k=1}^{m}\sum_{p=1}^{m} A_{ijkp}x_{ik}x_{jp} = A_{ijk'p'}x_{ik'}x_{jp'}$$

$$= \begin{cases} B'_{ij}x_{ik'}x_{jp'} = b'_{ij} \\ \quad \text{if } k' \neq p' \text{ and } i \text{ is connected to } k',\ j \text{ connected to } p' \\ B_{ij}x_{ik'}x_{jp'} = b_{ij} \\ \quad \text{if } k' = p' \text{ and } i \text{ and } j \text{ are connected to the same switch } k' \end{cases}$$

Let's define $H_{ij} = W_{ij}b_{ij}$ and $H'_{ij} = W_{ij}b'_{ij}$, Eq. (18b) becomes:

$$f_L = \sum_{i=1}^{n}\sum_{j=1}^{n} H_{ij}y_{ij} + \sum_{i=1}^{n}\sum_{j=1}^{n} H'_{ij}(1 - y_{ij}) + \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} \qquad (18c)$$

where y_{ij} is defined as in Eq. (6).

Eq. (18c) is then equivalent to Eq. (2) and we have:

Constraint (19) can be eliminated because it is already taken into account by the fact that all the cells of H are hubs;
Constraint (20) corresponds to constraint (1) and renders the fact that every cell should be allocated to one hub;
Constraints (21) and (22) imply that $z_{ik} = x_{ik}$, and in fact come down to constraint (20);
Constraints (23) and (24) are obvious and reflect the fact that the assignment solution contains only 0 (when there's no connection) and 1 (when a connection exists).

Relation (3) is an additional constraint on the switches' capacity.

The assignment problem is then equivalent to a p-fixed hubs location problem. Let's however stress that in the p-fixed hubs location problem, the flow between two cells can be partially forwarded through different hubs. Consequently, x_{ijkp} is not necessarily an integer whereas in the assignment problem, we have only integer values.

2.3. Classical approaches

The assignment problem being NP-hard [3,7,23], exact algorithms often based on an exhaustive enumeration of all the possibilities can be applied only to small sized problems. By reducing the problem to a graph-partitioning problem, Kerninghan and Lin [12] have demonstrated that the number of possible combinations increases rapidly. For a problem with m switches and n cells, at the first sight, we have to examine m^n combinations.

Global methods have been developed to reduce the set of combinations to be explored. Thus, the Branch and Bound exploration technique limits the search area through a cost function associated with each possibility or solution. It starts from a point or solution considered as a root, on which it carries out one transformation at a time. The transformation to be performed is the one (theoretically), which among all possible transformations, is capable of minimizing the cost of additional efforts to reach the optimum. Unfortunately, calculating the cost of additional efforts to reach the optimum is almost like knowing this optimum in advance, which is as difficult as the initial problem. Therefore, a cost function that is an approximation of the real cost is often used. For more details on Branch and Bound, see Ref. [11].

O'Kelly [19] proposed an algorithm for solving both the location (where should the switches be located) and assignment problems. However, this algorithm emphasizes the location issue. In a first method, cells are assigned to the closest switch. A variant considers the two closest cells. Although it explores all switch location possibilities, the assignment aspect (cells to switches) of the algorithm is very weak and efficient only for a limited number of switches (the optimum is reached with certainty when we

have one or two switches). Klincewicz [15] presented a heuristic based on multiple criteria, which for the assignment part, takes into account the distance as well as the rate of transfer between cells with standardized weights w_1 and w_2. In Ref. [4], Cox et al., used a short term tabu search meta-heuristic, with embedded knapsack and network flow sub-problems to design a least-cost telecommunications networks to carry cell site traffic to wireless switches while meeting survivability, capacity, and technical compatibility constraints. This simultaneously requires (1) a selection of the hub (or switch) locations and types; (2) an optimal assignment of the cell to the assigned hubs while respecting capacity constraints on the links and routing-diversify constraints on the hubs to assure survivability; and (3) an optimal choice of the types of links to be used in interconnecting the nodes and hubs based on the capacities and costs associated with each link type. Each of these optimization problems must be solved while accounting for its impacts on the other two. The part of the problem similar to our assignment problem is the point (2). In this part, the link cost function used is similar to ours, however handoff costs are not considered. Nonetheless, this problem illustrates how our assignment problem fits in the global network design scheme.

One disadvantage of the proposed heuristics is the fact that they try to localize switches and carry out assignment at the same time. This often results in assignment algorithms, which do not take into account either capacity constraints or the fact that the positions of switches are fixed and known.

Fiduccia and Mattheyses [6] improved Kernighan and Lin [12] algorithm by proposing to move one cell at a time, which introduces a bigger flexibility and allows an easier choice of the permutation to be carried out. They introduced the notion of gain of cell c noted gain (c) and defined it as the improvement on the objective function by moving cell c from one block to another. They also provided a simple way to update the gain table.

Merchant and Sengupta [17,18] studied directly the assignment problem. Their algorithm starts from an initial solution, which they attempt to improve through a series of greedy moves, while avoiding to be blocked in a local minimum. The moves used to escape a local minimum explore only a very limited set of options. These moves depend on the initial solution and do not necessarily lead to a good final solution. The same problem was also studied by Saha et al. [2,21]. They proposed a few assignment heuristics and compared them with other existing ones including the one of Merchant and Sengupta.

The previously mentioned methods, although sufficiently efficient and rapidly leading to a solution, need to integrate mechanisms that are often complex to escape the trap of the local minimum. To reach the optimum, these methods are above all dependent on an initial good solution. To develop an optimal algorithm that offers high performance, exact approaches such as those based on linear programming take advantage of the convexity of the search space. However, in the field of integer programming, this convexity is unfortunately useless. In these circumstances, the exploration methods lead to optima that are often local. Consequently, 'cleverer' methods, which take advantage of the history of the search to converge to a 'good' solution, are often used.

3. Tabu search approach

A tabu search (TS) method is an adaptive technique used in combinatorial optimization to solve difficult problems. It is considered as a meta-heuristic method because it can be applied to different instances of problems to provide good solutions [8–10,20–23].

3.1. Basic principle

The tabu search method is an improvement to the general descending algorithm, because it attempts to avoid the trap of local minima. For this purpose, it is necessary to accept from time to time, solutions, which do not improve the objective function, with the hope to reach better solutions later. However, accepting solutions that are not necessary the best introduces a cycle risk, i.e. a return to the solutions that have already been considered, hence the idea of keeping a *tabu list T* of solutions that have already been considered. Thus, during the generation of the set V of neighbour candidates, we remove the solutions that are in the tabu list.

On one side, keeping a tabu list allows for avoiding the cycle risk, on the other side, the storage of the solutions that have already been found could require large amount of memory. Moreover, it proves useful to be able to return to a solution that has already been considered, in order to continue the search in another direction. A compromise is reached by keeping a tabu list, which avoids cycles with a length that is less or equal to k. However, a cycle with a length higher than k could always occur.

The algorithm stops when no improvement has intervened in a number *kmax* of iterations or if all the neighbouring candidate solutions are tabu, i.e. $V - T = \varnothing$. Fig. 2 outlines the tabu search method.

To minimize the objective function, the tabu search method starts from an initial solution and attempts to reach a global optimum by applying moves that allow the passage from a solution s_i to another solution s_{i+1} chosen in the neighbourhood $N(s_i)$ of s_i (the neighbourhood $N(s_i)$ of a solution s_i is defined as the set of solutions that are accessible by applying one move to s_i). At each iteration i, the neighbourhood where solutions are selected is redefined as the neighbourhood of the current solution s_i. The tabu conditions change and the admissible solutions are no longer the same.

The exploration of the search domain could be represented by a graph $G = (X, A)$ where X refers to the set of solutions and A the set of arcs $(x, m(x))$, where $m(x)$ is the solution obtained by applying the move m to x. Then, a

```
Initialization : Choose an initial solution s in X.
                 s* := s (s* is the best solution until now)
                 nbiter := 0 ( nbiter is the iteration counter)
                 bestiter := 0 ( bestiter is the iteration number for which, we have found
                 the latest improvement, i. e, the latest s*)
                 T := ∅ (T is the tabu list)
                 Continue := true
While        Continue do
                 if (nbiter - bestiter > kmax) or (V-T=∅)
                     then Continue := false
                 else nbiter := nbiter + 1
                      Generate V ⊆ N(s)
                      Find the best solution s' in V (f(s')=min f(s) with s ∈ V - T)
                      s := s'
                      Update T
                      if f(s') < f(s*) then s* := s'
                                            bestiter:= nbiter
                      End if
                 End if
End While
```

Fig. 2. General algorithm of the tabu search method.

given move will be equivalent to a set of arcs $\{(x, m(x)), x \in X\}$. The graph G is symmetrical because for each arc $(x, m(x))$, there exists an arc $(m(x), x)$ obtained by applying the reverse move m^{-1} to $m(x)$. The tabu search method starts from an initial solution x_0, node of the graph G, and will search in G a path $x_0, x_1, \ldots, x_k$, where $x_i = m(x_{i-1})$ with $i = 1, \ldots, k$. Each intermediary solution of the path is obtained by applying a move to the previous solution. The arcs (x_i, x_{i+1}) of the path are chosen by solving the optimization problem:

$$f(x_{i+1}) = \min f(x_i) \text{ with } x_i \in V - T$$

where V in this case is the whole neighbourhood $N(x_i)$ of x_i.

If the set V–T of solutions to be explored is too large, the cost induced by the algorithm could become prohibitive. Conversely, a too small set has the adverse effect on the quality of solutions found. The tabu search method could be distinguished from the general descending method, mainly by the use of a shorter-term memory structure, which allows for accepting less good solutions in order to exit local optima, while avoiding cycles. For more details on the tabu method, see Refs. [8–10].

3.2. Adaptation of the tabu search method

To solve the assignment problem, we have chosen a search domain free from capacity constraints on the switches, but respecting the constraints of unique assignment of cells to switches. The feasibility of the final solution is therefore not guaranteed, but as we explore a large number of possibilities, we increase the chances of obtaining good solutions. We associate with each solution two values: the first one is the intrinsic cost of the solution (or simply the cost), which is calculated from the objective function; the second is the evaluation of the solution, which takes into account the cost and the penalty for not respecting the capacity constraints. This penalty includes two parts: one is fixed and the other is variable and proportional to the degree of violation. At each step, the solution that has the best evaluation is chosen. Once an initial solution built from the problem data, the short-term memory component attempts to improve it, while avoiding cycles. The middle-term memory component seeks to intensify the search in specified neighbourhoods, while the long-term memory aims at diversifying the exploration area.

3.2.1. Short-term memory

The short-term memory moves iteratively from one solution to another while prohibiting a return to the k latest visited solutions. It starts with an initial solution, obtained simply by assigning each cell to the closest switch, according to an Euclidean distance metric. The objective of this memory component is to improve the current solution, either by diminishing its cost or by diminishing the penalties. For this purpose, we use a move $m(a, b)$ defined as follows: $m(a, b)$: re-assignment of cell a to switch b.

This type of move offers a great flexibility, since it does not take into account the capacity constraints. Moreover, it

constitutes a basic move allowing carrying out more complex moves, e.g. permuting two cells.

The neighbourhood $N(S)$ of a solution S is defined by all the solutions that are accessible from S by applying a move $m(a,b)$ to S. To rapidly evaluate the solutions in the neighbourhood $N(S)$, we define the gain $G_S(a,b)$ associated with the move $m(a,b)$ and the solution S by:

$$G_S(a,b) = \begin{cases} \sum_{i=1,i=\alpha}^{n} (h_{ai} + h_{ia})x_{ib_0} - \sum_{i=1,i=a}^{n} (h_{ai} + h_{ia})x_{ib} \\ \quad +c_{ab} - c_{ab_0} \quad \text{if } b \neq b_0 \\ M \qquad\qquad\qquad \text{if not} \end{cases} \tag{25}$$

where:

- h_{ij} refers to the handoff cost between cells i and j;
- b_0 is the switch of cell a in solution S, that is, before the application of move $m(a,b)$;
- x_{ik} takes value 1 if cell i is assigned to switch k, 0 otherwise;
- c_{ik} is the cost of linking cell i to switch k;
- M is an arbitrary large number.

At each iteration, the move having the weakest gain among all possible moves is selected. If $b \neq b_0$, $G_S(a,b)$ represents the gain on cost $f(S)$ of solution S, when the move $m(a,b)$ is carried out. For $b = b_0$, the move $m(a,b)$ does not change the solution and the gain $G_S(a,b)$ should be null. However, in this last case, we assign to the gain an arbitrary large value. Thus, when we reach a local minimum and no available move could improve the solution cost, the move that has the minimum gain would not be move $m(a,b_0)$ which cycles on the same solution. The cost of the new solution S' is simply obtained from the formula:

$$f(S') = f(S) + G_S(a,b) \tag{26}$$

At the beginning, we generate all gains $G_S(i,k)$ in a table $n \times m$, where n refers to the number of cells and m to the number of switches. After each move $m(a,b)$, we update the gain table. This update is rapidly carried out, because only the columns corresponding to switches b and b_0 and the lines corresponding to cell a and to cells assigned to switch b or b_0 change. For any cell p and any switch q, let denote by C_p the switch of p in the new solution S' where cell a is now assigned to switch b. The new gains $G_{S'}(p,q)$ are expressed as follows:

$$G_{S'}(p,q) = G_S(p,q) + (h_{ap} + h_{pa}) \quad \text{if } C_p = b,\ p \neq a,\ q \neq b_0,\ q \neq b \quad \text{or} \quad \text{if } C_p \neq b,\ C_p \neq b_0,\ q = b_0 \tag{27}$$

$$G_{S'}(p,q) = G_S(p,q) - (h_{ap} + h_{pa}) \quad \text{if } C_p = b_0,\ q \neq b_0,\ q \neq b \quad \text{or} \quad \text{if } C_p \neq b,\ C_p \neq b_0,\ q = b \tag{28}$$

$$G_{S'}(p,q) = G_S(p,q) - 2(h_{ap} + h_{pa}) \quad \text{if } C_p = b_0,\ q = b \tag{29}$$

$$G_{S'}(p,q) = G_S(p,q) + 2(h_{ap} + h_{pa}) \quad \text{if } C_p = b,\ p \neq a,\ q = b_0 \tag{30}$$

$$G_{S'}(p,q) = M \quad \text{if } C_p = q \tag{31}$$

$$G_{S'}(a,b_0) = -G_S(a,b) \tag{32}$$

$$G_{S'}(a,q) = G_S(a,q) - G_S(a,b) \quad \text{if } q \neq b_0,\ q \neq b \tag{33}$$

$$G_{S'}(p,q) = G_S(p,q) \text{ for all other cases.} \tag{34}$$

At each iteration, we define the neighbourhood of the current solution that is composed of solutions generated by all moves $m(a,b)$, then we choose the move that improves the solution the most, e.g. the move having the weakest gain. If there is no move that can improve the current cost, the solution that generates the minimum degradation of the cost is selected. Once a move $m(a,b)$ is performed, the method keeps in a tabu list the reverse move $m(a,b_0)$, where b_0 denotes the switch of cell a before applying move $m(a,b)$. We stop if *kmax* iterations have been done since the latest best solution or if there are no more available moves.

Intuitively, there are more chances to reach a better solution by increasing the delay *kmax*. However, we remark that, if we take increasingly large values for *kmax*, the search would be taken away from feasible solutions with a little chance to come back to feasible solutions. We use a 'callback' mechanism to bring the exploration back to feasible solutions. After a given number of consecutive non-feasible solutions, we penalize the gains of moves leading to non-feasible solutions. We incrementally add a penalty (composed of a constant part and a variable one) to the gains of moves, which add a cell to a switch that already has a negative residual capacity. The incremental feature of the penalty is implemented by introducing a multiplier in the variable part. At each non-feasible solution, this multiplier is incremented until a maximal value Maxsev. When the algorithm finds the first feasible solution, the callback mechanism is deactivated. By incrementally penalizing the moves, we avoid a brutal transition that could have neglected the best solutions.

The tabu list helps TS avoid cycles. However, it is sometimes profitable to return to a tabu solution and take the search in another direction. The tabu criterion of a move is thus cancelled, if the move leads to a solution whose value is lower than that of the best-known solution. The use of a gain table accelerates the choice of the next move. This implementation increases a lot the efficiency of our algorithm because the short-term memory component (and therefore the choice of a move $m(a,b)$ in the neighbourhood $N(S)$) is frequently used. To have better results, we add a middle-term component to the short-term memory.

3.2.2. Middle-term memory

The middle-term memory component tries to intensify the search in promising regions. It is introduced after the end of the short-term memory component and allows a return to solutions possibly omitted. It mainly consists of defining the regions of intensified search, and then choosing the types of move to be applied.

Defining a promising region for an intensified search is not an easy task. We chose to keep a list of the latest best solution, as well as the values associated with gains allowing to later restore the search context. Every time we find a solution s' that improves the latest best solution found, s' is substituted to the best oldest solution of the list. This approach is justified by the fact that, in an assignment context, we can suppose that best solutions are not far from each other.

The basic intensification idea is to vary the moves towards solutions, which were neglected by the short-term memory moves. In our case, two types of intensification moves have been defined:

- $i_1(a, c)$: permute cells a and c according to the weakest gains;
- $i_2(a, b)$: remove cell a from switch band reassign it to another cell in order to re-establish the capacity constraints.

Case 1: Move $i_1(a, c)$

The objective of this move is to improve the evaluation of a solution by decreasing the associated cost. However, to avoid the move $m(a, b)$ that has already been defined in the short-term memory component, we decided to review the gain table and choose two cells a and c whose permutation seems to generate the largest gain. In fact, the two cells are selected without taking into account the fact that, after removing cell a, the gain table changes and cell c is no longer necessarily the best to be removed. However, since this move is generally applied to feasible solutions, the objective is to disturb as slightly as possible the topology of the solution, while attempting to improve it. The estimate gain is rapidly calculated and, even though it does not take into account the capacity constraints, it proves sufficient to guide the search.

For the sake of efficiency and generality, we applied the permutation with the already defined mechanisms for the moves $m(a, b)$. The permutation is divided into two consecutive sub-moves of cells. Thus, two reverse moves will be recorded in the tabu list LT1 associated with moves i_1. A permutation is tabu if at least one of its sub-moves is. A multiplying factor of 2 is used to always keep the two tabu sub-moves. The aspiration criterion is defined in the same manner as in the case of the short-term memory: it consists of accepting a tabu permutation, if this permutation leads to a solution better than those already obtained. As move $i_1(a, c)$ is generally applied to already feasible solutions, it does not take into account the capacity constraints and is exclusively supported by gain estimate for the choice of the permutation. For non-feasible solutions, move $i_2(a, b)$ will be applied.

Case 2: Move $i_2(a, b)$

The objective sought by this type of move is to restore capacity constraints, and thereby decrease both the penalties and solution evaluation. Move $i_2(a, b)$ consists of:

- determining switch c' which has the minimal residual capacity;
- finding cell a assigned to c' which has the minimal call volume;
- assigning cell c' to a switch b which has a sufficient residual capacity and gives the minimal gain.

This move is based on both capacity constraints and gains. It has a tabu list $LT2$ of the same size as $LT1$ list and does not implement any aspiration criterion.

The middle-term memory component, just like the short-term memory component, stops if the best solution has not been improved during the latest *ikmax* iterations, or if there are no more available moves. Then, follow diversification mechanisms.

3.2.3. Long-term memory

To diversify the search, we use a long-term memory structure in order to guide the search towards regions that have not been explored. This is often done by generating new initial solutions. In this case, a table $n \times m$ (where n is the number of cells and m the number of switches) counts, for each arc (a, b), the number of times this arc appears in the visited solutions. A new initial solution is generated by choosing, for each cell a, the least visited arc (a, b). Solutions visited during the intensification phase are not taken into account because they result from different types of moves than those applied in short and long-term memory components. From the new initial solution, we start a new search with short and middle-term memory mechanisms.

4. Computational experience and results

We submitted our approach to a series of tests in order to determine its efficiency and sensitivity to different parameters. Thus, we will present a few results, which we compare to those provided by other known heuristics.

4.1. Test generation

To select a global reliable behaviour, we executed our program on a large number of test cases inspired from Ref. [16]. A MATLAB® program has generated the tests by supposing that the cells are arranged on a hexagonal grid of almost equal length and width. The antennas are located at the center of cells and distributed evenly on the grid. However, when two or several antennas are too close to each other, the antenna arrangement is rejected and a new

Table 1
Test cases used to execute the plan of experiments

Number of cells	Number of switches	Number of test cases
15	2	50
30	3	50
50	4	50
100	5	50
150	6	50
200	7	50

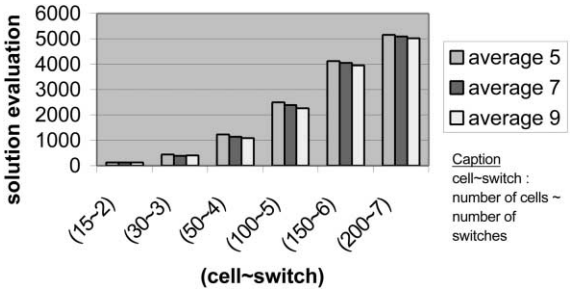

Fig. 3. Effect of the size of the tabu list on obtained solutions.

arrangement is chosen. The cost of linking a cell to a switch is proportional to the distance separating both. We took a proportionality coefficient equal to the unit. The call rate γ_i of a cell i follows a gamma law of average and variance equal to the unit. The call duration inside the cells are distributed according to an exponential law of parameter equal to 1. This is justified by the fact that we are considering a simple model. For more realistic call duration distributions and models, see Ref. [5]. If a cell j has k neighbors, the $[0, 1]$ interval is divided in $k + 1$ sub-intervals by choosing k random numbers distributed evenly between 0 and 1. At the end of the service period in cell j, the call could be either transferred to the ith neighbour ($i = 1, \ldots, k$) with a handoff probability r_{ij} equal to the length of the ith interval, or ended with a probability equal to the length of the $k + 1$th interval. To find the call volumes and the rates of coherent handoffs, the cells are considered as $M/M/1$ queues forming a Jackson's network [14]. The incoming rates α_i in cells are obtained by solving the following system:

$$\alpha_i - \sum_{j=1}^{n} \alpha_j r_{ji} = \gamma_i \quad \text{avec } i = 1, \ldots, n$$

If the incoming rate α_i is greater than the service rate, the distribution is rejected and chosen again. The handoff rate h_{ij} is defined by:

$$h_{ij} = \lambda_i r_{ij}$$

All the switches have the same capacity M calculated as follows:

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

where K is uniformly chosen between 10 and 50, which insures a global excess of 10–50% of the switches' capacity compared with the cells' volume of call.

4.2. Plan of experiments

Our plan of experiments aims at testing the parameters of each memory component of the proposed approach. As we have a very large number of possible combinations, we changed only one parameter at a time. For each parameter, we choose three possible discrete values. The algorithm is executed for each of the values of the parameter over a set of 300 test cases with a number of cells varying between 15 and 200 and a number of switches varying between 2 and 7. Table 1 summarizes the test cases used.

Essentially, the short-term memory displays two processes: a tabu list for reducing the cycle risks and a callback mechanism that orients the exploration towards feasible solutions. Tests are carried out while all other middle and long-term memory components are deactivated.

According to Glover [8], the size of the tabu list must be around 7, independently of the size and structure of the problem. We wanted however to study the impact of different choices of size for the tabu list on the quality of obtained solutions. For this purpose, we executed our program with tabu lists of sizes 5, 7, and 9. Then, we made an average for all test cases having the same number of cells and switches. Fig. 3 represents the average of the evaluation of obtained solutions in function of the number of cells and switches, and this, for each of the chosen values on the tabu list. We note that, on average, a tabu list of size 9 gives best results, i.e. low cost solutions. In fact, the callback mechanism has the tendency to quickly bring the exploration back to feasible solution areas. Therefore, with a short tabu list, when the search leaves the non-feasible regions, it is more likely to come back to already explored regions if it is not far enough. Consequently, we explore fewer solutions and have fewer chances to have good results.

During the exploration of the search area, the short-term memory component uses a callback mechanism to take the search back to feasible solution regions. The callback mechanism is activated after a certain delay measured in number of non-feasible consecutive solutions. This delay should not either be too long because the search will be moved far away from promising solutions or too short because the search would constantly be brought back to areas of feasible solutions and thereby could omit best solutions. We studied the case where the callback mechanism is triggered after 2, 5, and 8 consecutive non-feasible solutions. We chose an upper limit of 8 because otherwise, the search is brought too far from feasible regions. Fig. 4 gives the average of the evaluation of solutions that are obtained in function of the number of cells and switches, and this for each value chosen for the delay of triggering the callback mechanism. On average, a delay of triggering after two consecutive non-feasible solutions provides best solutions.

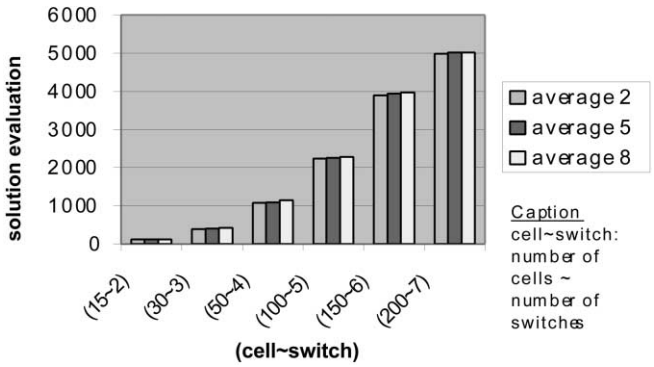

Fig. 4. Effect of the activation delay of the callback mechanism on the solutions.

Indeed, the callback mechanism (incrementally applied) must be triggered as soon as possible in order to orient the search towards feasible solutions.

The intensity of the callback mechanism is incrementally modulated with the help of a multiplier that is incremented at each non-feasible solution until a maximal value Maxsev. Fig. 5 represents the average evaluation of obtained solutions in function of the number of cells and switches, and this for Maxsev values of 10, 15 and 18 respectively.

We do not note a sensible difference between the results obtained for the three values of the chosen Maxsev. This means that, in general, the callback mechanism is not applied until the smallest among the chosen values for the maximal intensity, because we find no more than 10 consecutive non-feasible solutions during the execution of the algorithm. Thus, for our simulation, we took a value of 15, because this value is the average among the three used for our test. The execution time is not particularly affected.

The middle-term memory component keeps the latest best solutions and intensifies the search on their neighbourhood. It uses two types of moves: the first move exchanges two cells; the second consists of re-assigning cells in order to re-establish the capacity constraints. The first move is the default one; the second one is applied after a certain number of consecutive non-feasible solutions.

As far as diversification is concerned, the most important parameter remains the number nbstart of starts. Tests were carried out without the activation of the intensification mechanism. In order to capture the true influence of the parameter nbstart on the quality of obtained solutions, the values chosen for the variable nbstart are 1, 2, 3, 5 and 10 respectively. The case of a single restart serves as a reference. The influence of execution time is taken into account by considering the relation between improvement of solutions and increases in execution times. Fig. 6 shows the results. We notice that either the improvement resulting from more restarts decreases linearly or there is a sharp decline if more than 3 restarts are used. Consequently, three re-starts give in general the best solutions.

More globally, the contribution of the intensification and diversification mechanisms to the improvement of solutions increases with the problem size. The percentage of feasible solutions obtained by using the short-memory component generally decreases with the number of switches. For a large number of switches, the short-term memory component hardly generates a feasible solution. For a fixed number of switches and a variable number of cells, the percentage of feasible solutions remains of the same order, whatever mechanisms are applied. With the activation of the intensification and/or diversification mechanism, we obtain more feasible solutions. However, the middle-term memory is proven to be more efficient than the long-term memory in

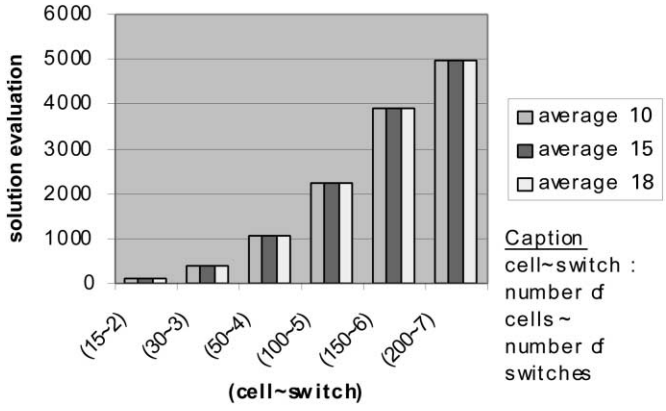

Fig. 5. Effect of the variation of the maximal intensity of the callback mechanism on the solutions.

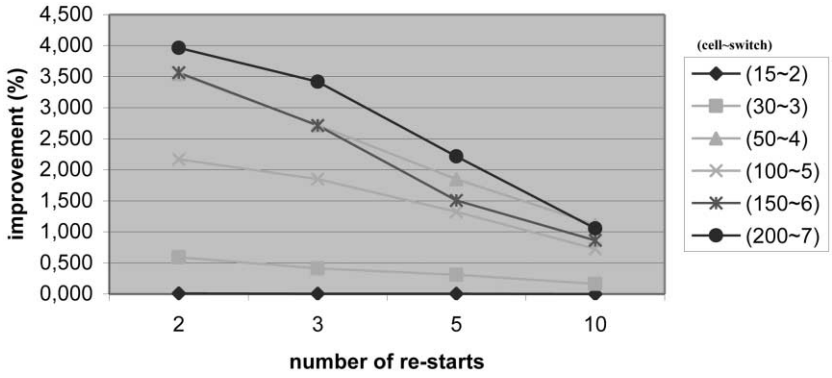

Fig. 6. Effect of the number of restarts.

the generation of good solutions. This is because the intensification mechanism implements a move that has as an explicit objective of re-establishing feasibility, while the diversification mechanism attempts to globally improve the solution without specifically seeking to re-establish the feasibility.

4.3. Comparison with other heuristics

Merchant and Sengupta [18] have designed a heuristic, which we call *H*, for solving the cell assignment problem. They provide little comparative data that are applicable to our case. However, for a few cases, we have compared execution times and percentages of feasible solutions. Our tests were about cases with variable number of cells and four switches. To reproduce the same conditions as Merchant and Sengupta, we run our program on a 'SUN SPARCSTATION 2 (4/75)'. Table 2 presents a comparison of results.

The two heuristics always find feasible solutions. For a large number of cells, our TS algorithm is a bit faster than heuristic *H*. Conversely, for problems of smaller size, TS is a bit slower. However, these results inform only on the feasibility of obtained results without demonstrating whether these solutions are among the best. In Ref. [18], Merchant and Sengupta have presented a detailed implementation example of their method. On the data of this example, like heuristic *H*, our TS algorithm leads to the optimum. Simulated annealing (SA) is a probabilistic algorithmic approach to solve optimization problems. Kirkpatrick et al. [13] adapt it to solve combinatorial optimization problems. Conversely to a local search algorithm, SA allows for a given optimization problem to accept solutions which deteriorate the cost, even if later these solutions will be abandoned if they generate no improvements. SA uses randomness to decide whether to reject or accept a solution, which deteriorates the cost. In each of the six considered series of tests introduced in Table 1, we have chosen two test cases for each number of cells and switches. SA implementation has been executed 500 times on each of the cases and we retained the best evaluation. Tables 3 and 4 present comparisons for a few test series. Generally, RT algorithm performs better than SA algorithm. In particular, for large size problems, RT finds solutions that are twice better than those of SA.

Table 2
Comparison of TS heuristic with *H* heuristic

Cells	# Problems		% Feasible solutions		CPU Times (s)	
	H	TS	*H*	TS	*H*	TS
15	74	50	100	100	0.05	0.08
20	74	50	100	100	0.09	0.13
30	74	50	100	100	0.16	0.24
40	29	50	100	100	0.30	0.37
50	9	50	100	100	0.43	0.53
60	9	50	100	100	0.64	0.65
70	9	50	100	100	0.88	0.82
80	9	50	100	100	1.22	1.02
90	9	50	100	100	1.53	1.29
100	9	50	100	100	1.96	1.53

5. Lower bounds on the global optimum

A TS-based solution does not necessarily correspond to a global optimum. For lack of knowing the global optimum, we will define a lower bound allowing to evaluate the quality of solution provided by this approach.

Let n be the number of cells to be assigned to m switches and x_{ik} a decision variable such as:

$$x_{ik} = \begin{cases} 1 & \text{if cell } i \text{ is related to switch } k, \\ 0 & \text{if not.} \end{cases}$$

Let λ_i be the volume of calls per hour intended to cell i, M_k the capacity per hour of switch k, and h_{ij} the handoff cost per hour from cell i to cell j. The assignment problem essentially consists of:

Minimizing

$$f = \sum_{i=1}^{n}\sum_{k=1}^{m} c_{ik}x_{ik} + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} h_{ij}\left(1 - \sum_{k=1}^{m} x_{ik}x_{jk}\right)$$

subject to the following constraints already formulated in

Table 3
Comparison with SA Method (case 1)

# Cells	# Switch	TS		% Improvement TS in relation to SA
15	3	105	103	2
15	3	139	124	11
20	3	189	211	− 12
20	3	161	158	2
30	3	359	295	17
30	3	369	364	2
40	3	553	444	20
40	3	611	420	31
50	3	724	597	18
50	3	748	588	21
60	3	832	713	14
60	3	1073	866	19

Section 2:

$x_{ik} = 0$ or 1 for $i = 1, \ldots, n$ and $k = 1, \ldots, m$

$$\sum_{k=1}^{m} x_{ik} = 1 \quad \text{for } i = 1, \ldots, n$$

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k \quad \text{for } k = 1, \ldots, m$$

An intuitive lower bound for the problem is:

$$LB1 = \sum_{i=1}^{n} \min_{k} (c_{ik})$$

which is the link cost of the solution obtained by assigning each cell to the nearest switch. This lower bound does not take into account handoff cost. In fact, we suppose that capacity constraint (3) is totally relaxed and that all cells could be assigned to a single switch. Thus, we have a lower bound whatever the values of M_k and λ_i.

Another bound could be found if we do not consider the trivial case where all cells are assigned to a same switch.

Table 4
Comparison with SA Method (case 2)

# Cells	# Switch	SA	TS	% Improvement TS in relation to SA
15	2	124	130	− 5
15	2	123	118	4
30	3	405	382	5
30	3	448	324	28
50	4	951	689	28
50	4	851	580	32
100	5	1999	1374	31
100	5	2595	1234	52
150	6	4271	2013	53
150	6	3240	2010	38
200	7	5550	2948	47
200	7	7801	2768	64

Table 5
Relative distance between our solutions and the lower bound (series no. 4)

# Switches	Mean distance (%)
2	3
3	6
4	11
5	15
6	20
7	21

Then we could state, without a great loss of generality, that a single switch cannot take in charge all the cells. Thus, we have at least a bi-partition of the cell set. In the case of a bi-partition (p, q) of a set of n cells, the total number of handoff costs to take into account is $t = 2pq$. The minimal number of handoffs to be considered for a bi-partition is obtained by solving the problem:

$\min 2pq$

subject to:

$p + q = n, \quad p \geq 1, \; q \geq 1.$

This problem admits bi-partitions $(1, n-1)$ and $(n-1, 1)$ as solutions. Among all possible partitions, these solutions are those, which generate the least handoffs (because a multi-partition is obtained by dividing several times again a bi-partition). Therefore, a lower bound for the number of handoffs is $2(n-1)$. Let's note that if handoff $i \rightarrow j$ is taken into account, it is necessary to do the same with handoff $j \rightarrow i$. Let h_T be the triangular upper part of matrix $(H + H^T)$ where H refers to cost matrix and H^T to its transposed. If we want to take into account the handoff cost in the lower bound, we have to consider at least $n-1$ handoff of h_T. Thereby, a new lower bound is:

$$LB2 = \sum_{i=1}^{n} \min_{k} (c_{ik}) + \sum_{p=1}^{n-1} \sum_{q=p+1}^{n} 1_{N\_}\{h_T(p,q)\} h_T(p,q)$$

where $N\_$ refers to the set of the first $n-1$ minima of the triangular matrix h_T, $1_{N\_}$ refers to the indicative function of set $N\_$, with $1_{N\_}\{x\} = 1$ if $x \in N\_$ and 0 if not. Finally, $h_T(p, q)$ refers to the element of line p and column q of the triangular matrix h_T. To finish, let's note that $LB1 \leq LB2$ and therefore $LB2$ is better lower bound than $LB1$.

However, due to the fact that each cell could have only 6 neighbours (if we consider hexagonal cells), the $(n-1)$ first minima of h_T are often null (if the number of cells is more than a few tens). Then $LB1$ and $LB2$ are often equal. Tables 5 and 6 show the relative distance between our solutions and the lower bounds for test series nos 4 and 6.

The TS method gives solutions 'close' to the lowest bound (and thereby to the global optimum). Let's remind that our lowest bound does not include handoff costs and therefore, no solution could equal the lowest bound.

Table 6
Relative distance between our solutions and the lower bound (series no. 6)

Cells	Mean distance (%)
15	11
30	9
50	10
100	9
150	9
200	8

6. Concluding remarks

In this paper, we proposed a new mathematical equivalence between the assignment problem and the well-known p-fixed hubs location problem. We then used a tabu search approach to efficiently solve the problem. In our implementation, a gain structure and its update procedures were defined to improve the tabu-search algorithm performances. We also implemented some advance long and middle-term mechanisms to improve the quality of the obtained solutions. To evaluate the performance of this approach, we defined two lower bounds for the global optimum, which are used as references to judge the quality of the obtained solutions. Generally, the results are sufficiently close to the global optimum. Our implementation has been also tested in relation to different parameters of tabu search. Finally, the results obtained have been compared to the results provided by other heuristics, which confirmed the efficiency and robustness of our approach, particularly for large-scale networks with more than 50 cells and 3 switches. Our algorithm has also been tested with data supplied by a local network operator and the obtained results were also satisfying. However the best guarantee of the efficiency of the algorithm is the large range of the data used in our tests.

If the traffic distribution changes, the network topology should be computed all again. This could lead to a totally different assignment, incompatible with the already installed equipment. A workaround could be to modify the problem constraints to force a solution close to the existing topology.

A future work could also be an adaptation of the algorithm so that it could generate incremental modifications to adapt the existing network to the traffic evolution. A version of the algorithm taking in consideration the double homing problem could also be designed.

References

[1] S. Abdinnour-Helm, A hybrid heuristic for the uncapacitated hub location problem, European Journal of Operational Research 106 (1998) 489–499.
[2] Bhattacharya, P.S., Saha, D., Mukherjee, A., Heuristics for assignment of cells to switches in a PCSN: a comparative study, Proceedings 1999 IEEE Int. Conf. On Personal Wireless Comm., Feb 1999, pp. 331–334.
[3] Beaubrun, R., Pierre, S., Conan J. An efficient Method for Optimizing the Assignment of Cells to MSCs in PCS Networks, Proceedings 11th Int. Conf. on Wireless Comm., Wireless 99, vol. 1, July 1999, Calgary (AB), pp. 259–265.
[4] L.A. Cox Jr, J.R. Sanchez, Designing least-cost survivable wireless backhaul networks, Journal of Heuristics 6 (4) (2000) 525–540.
[5] Y. Fang, I. Chlamtac, Y. Lin, P.C.S. Modeling, Networks under general call holding time and cell residence time distributions, IEEE/ACM Transactions on Networking 5 (6) (1997) 893–905.
[6] Fiduccia, C.M., Mattheyses, R.M., A linear-time heuristic for improving network partition, Proceedings 19th Design Automat. Conf., 1982, pp. 175-181.
[7] M.R. Garey, D.S. Johnson, Computers and Intractability, Freeman, San Francisco, CA, 1979.
[8] F. Glover, Tabu Search — Part I, ORSA Journal on Computing 1 (3) (1989) 190–206.
[9] F. Glover, Tabu Search: A Tutorial, INTERFACES 20 (4) (1990) 74–94.
[10] F. Glover, E. Taillard, D. de Werra, A user's guide to tabu search, Annals of Operations Research 41 (3) (1993) 3–28.
[11] E. Horowitz, S. Sahni, Fundamentals of computer algorithms, Computer Science Press, Inc, 1990.
[12] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, The Bell System Technical Journal 49 (1970) 291–307.
[13] S. Kirkpatrick, C.D. Gelatt Jr, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[14] L. Kleinrock, Queuing Systems, I., Theory, Wiley, New York, 1975.
[15] J.G. Klincewicz, Heuristics for the p-hub location problem, European Journal of Operational Research 53 (1991) 25–37.
[16] B. Krishnamurthy, Constructing test cases for partitioning heuristics, IEEE Transactions on Computers C-36 (1987) 1112–1114.
[17] A. Merchant, B. Sengupta, Multiway graph partitioning with applications to PCS networks, IEEE Infocom'94 2 (1994) 593–600.
[18] A. Merchant, B. Sengupta, Assignment of cells to switches in PCS networks, IEEE/ACM Transactions on Networking 3 (5) (1995) 521–526.
[19] M.E. O'Kelly, A quadratic integer program for the location of interacting hub facilities, Journal of Operational Research 32 (1987) 393–404.
[20] S. Pierre, A. Elgibaoui, A tabu-search approach for designing computer-network topologies with unreliable components, IEEE Transactions on Reliability 46 (3) (1997) 350–359.
[21] D. Saha, A. Mukherjee, P.S. Bhattacharya, A simple heuristic for assignment of cells to switches in a PCS network, Wireless Personal Communications 12 (3) (2000) 209–224.
[22] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, ORSA Journal on Computing 2 (1) (1989) 33–45.
[23] D. Skorin-Kapov, J. Skorin-Kapov, On tabu search for the location of interacting hub facilities, European Journal of Operational Research 73 (1994) 502–509.
[24] J. Sohn, S. Park, Efficient solution procedure and reduced size formulations for p-hub location problems, European Journal of Operational Research 108 (1998) 118–126.