

# Some Algorithms for Synchronizing Clocks of Base Transceiver Stations in a Cellular Network

Jean-Louis Dornstetter\*, Daniel Krob†,  
Michel Morvan† and Laurent Viennot‡

November 10, 1998

## Abstract

This paper deals with synchronization problems arising in the context of cellular networks. It presents and compares several algorithms that can be used for solving these problems.

## 1 Introduction

The Base Transceiver Stations (BTS) are the components of a cellular network that ensure the connection between the Mobile Stations (MS) and the communication network. Every BTS has an internal clock, but all these different clocks are not synchronized. In other words, the BTS do not have access to any absolute reference of time.

We are here interested in proposing distributed algorithmical solutions for synchronizing these clocks. The main difficulty of our problem comes from the fact that we do not give us any possibility of measuring the dephasing between a given clock and some time reference. The only informations come from random measures of local dephasings between two clocks : one can indeed measure such dephasings (with noise) when a Mobile Station makes an handover (i.e. lies in the intersection of two cells covered by the two corresponding BTS). Moreover only these two BTS are informed of this measure. We also suppose that no information can be exchanged by mean of messages between the different BTS. Note finally that there is another difficulty for designing algorithms in such a context : an algorithm must indeed necessarily be executed in a totally non

---

\* Nortel Matra Cellular France – 38, rue Paul Cézanne – BP 50 – MS MI 54 – 78042  
Guillancourt Cedex – France – e-mail: [dornstetter@nortel.can](mailto:dornstetter@nortel.can)

† LIAFA(CNRS) – Université Paris 7 – 2, place Jussieu – 75251 Paris cedex 05 – France –  
e-mail: [dk](mailto:dk), [morvan](mailto:morvan), [lavie@liafa.jussieu.fr](mailto:lavie@liafa.jussieu.fr)

‡ corresponding author: Laurent Viennot

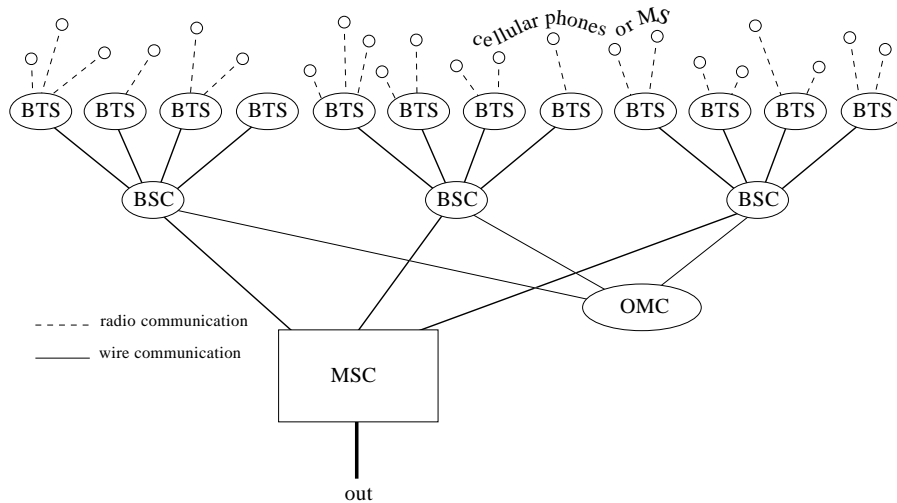
synchronized way on the different BTS. Such a problem never occurs in classical situations of parallel or distributed computations.

One of the purpose of this paper is therefore to try to present some ideas that can however be applied in such a constrained context. We must indeed put the stress on the fact that there is clearly no unique solution to our problem. Let us also give the structure of our paper. We present in a first section the detailed specifications of our synchronization problem. The second section deals then with some algorithmical solutions.

## 2 The synchronization problem

### 2.1 Cellular networks

In a first approximation (that will be sufficient for the purpose of this paper), one can model a cellular communication network as follows : the Mobile Stations (MS) are connected to the network through radio communications with the so-called Base Transceiver Stations (BTS), each BTS being in charge of all the communications of a certain geographic area called a *cell*; the different BTS are then organized in a hierarchic way (see Figure 1).



**Figure 1:** A cellular GSM network.

During a communication, the MS is governed by the BTS that manages the cell in which the MS lies. The MS sends periodically bursts to its BTS at a frequency which is given by the internal clock of the BTS (each burst is sent during a eighth of the period of the clock). The MS synchronizes its own clock with the clock of the BTS. Each burst is composed of two parts : the first one

contains standard system informations and the second one is devoted to the encoding of the voice signal.

Each BTS has its own clock. Our problem consists in finding a method for synchronizing all these clocks. This problem is rather important for several reasons. First of all, one must recall that a MS that moves is managed by several BTS during its motion. When a MS switches from one BTS to the other, it must emit shorter bursts in order to avoid perturbing the communications of its new BTS. Moreover the communication between a MS and its BTS perturbs the communications with any nearby unsynchronised BTS: whenever a burst of the communication overlaps properly in time another burst of a MS communicating with the nearby BTS, the interferences cannot be properly treated; this is not the case when the bursts totally overlap or do not overlap at all. Synchronization thus allows the network to manage more MSs.

The present GSM networks work in spite of this problem. A hardware solution would of course be possible. In this paper we present simple software ways of solving the problem. During a handover, *i.e.* when a MS switches from one cell to another, the two corresponding BTS are informed of the dephasing of their clocks. We propose solutions to get synchronization using this only information. This information is unpredictable and measured with some noise. An interesting point is that BTSs interfering often will also often get dephasing measures of their clocks since the interferences come from the presence of many MSs in the overlapping area of the two corresponding cells.

Each BTS can change its phase by slightly speeding or slowing its clock (it should not break the communications in course)<sup>1</sup>. We are thus looking for a distributed algorithm for controlling this actions in order to get synchronization. Each BTS must decide whether it should slow or speed its clock according to the dephasing measures it receives.

## 2.2 Modeling the Problem

We model this synchronization problem using a graph constructed in the following way. The BTSs are the vertices and we put an edge between two BTSs when their clock dephasing is measured frequently enough (we simply say that they interfere). This graph is unknown but distant BTSs are generally less likely to interfere. Moreover it evolves in time with the variations of the communication traffic.

We can enhance our model by taking a discrete time. Let  $p_{i,j}(t)$  be the probability that BTS  $i$  gets a measure of its clock dephasing with BTS  $j$  during the unit of time  $t$ . This can be defined as soon as the unit of time is sufficiently long for grabbing the regularities of the communications traffic (the unit of time can typically be a second long since a BTS can receive dephasing measures every few seconds during high traffic periods). We thus get a probability matrix  $P(t)$  modeling the dephasing measures between the BTSs during the time. The

---

<sup>1</sup>At maximum rate, a BTS may change its clock phase by a half period in approximately two hours.

matrix is symmetric since a dephasing measure is made when a MS is in charge of both BTSs that receive that measure.

The geographic positions of the BTSs are fixed and the flows of MSs are quite regular. The BTSs interfering with a fixed BTS quite often are almost always the same, we may thus suppose that the probability matrix is constant if we consider the problem of synchronizing the network during a long period of time (several months would be acceptable in practice)<sup>2</sup>.

Fixing a probability threshold, we get an undirected graph modeling the dephasing measures: two BTSs are linked if the probability that they receive a dephasing measure is greater than the threshold. We can then see the synchronization problem as a connected components computation. Indeed, it is possible to synchronize two BTSs if there exists a path connecting them (otherwise, there is no way in our model to measure their clock dephasing). This imply that we can use some ideas from graph connected components methods: our problem consists more or less in computing the connected components of some graph in our model.

However, the most classical approach consists in seeing this problem as an optimization problem: since we can for instance look for an algorithm that minimizes the sum of the squares of the dephasings. We first study this approach and then consider the connected component view for elaborating synchronization algorithms. A good solution should certainly mix these two approaches as we will see.

## 3 Synchronization Algorithms

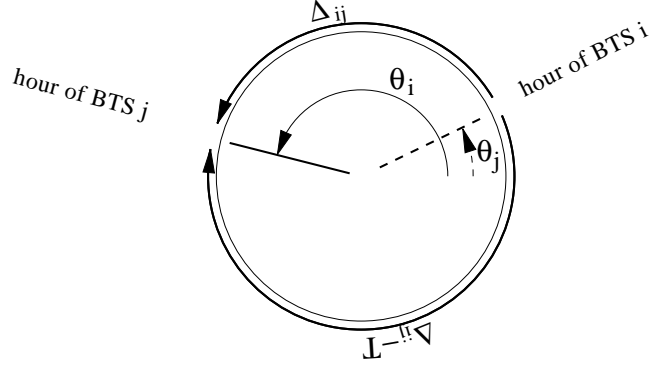
### 3.1 Gradient Algorithm

We first define a global cost function  $F$  whose absolute minima correspond to the exact synchronization of the BTSs. Let  $\theta_i$  denote the dephasing of BTS  $i$  with respect to some absolute reference time (note that  $\theta_i$  cannot be measured). Now let  $\Delta_{ij}$  be the dephasing between BTSs  $i$  and  $j$ . All the clocks have same period  $T$ . As we will see, this modular aspect of the variables makes the problem difficult. For now, just notice that there are in fact two possible dephasings: a BTS can adjust its time to another BTS by speeding its clock or by slowing it (see Figure 2), notice that one way is shorter than the other. In some situations, a BTS must adjust its clock by the longest way as in Figure 3.

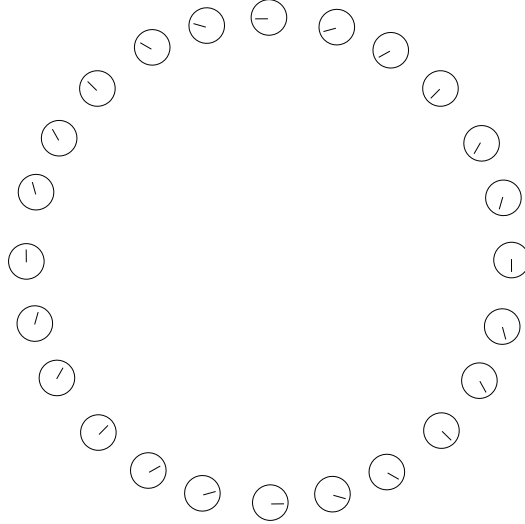
If the BTSs are approximately synchronized, the modularity problem disappears: suppose we have a very loose synchronization in the sense that there exists a reference time such that all the  $\theta_i$  are in the range  $-T/4, T/4$ . In that case, an algorithm as simple as the Gradient Algorithm allows to achieve synchronization. We define a global cost function  $F$  and a local cost function  $f_i$  for

---

<sup>2</sup>This may suppose some adaptations like taking a longer unit of time during the night, but it still gives an framework solution for elaborating algorithms that should be afterwards tuned for taking this in account.



**Figure 2:** The two possible values for the dephasing between two BTSs (modulo  $T$ ). BTS  $i$  can adjust its time according to the clock of BTS  $j$  by slowing of  $|\theta_j - \theta_i - T|$  or by speeding of  $|\theta_j - \theta_i|$ .



**Figure 3:** Each BTS interferes with the two nearby other BTSs. A deadlock situation: to synchronize the BTSs, one of them has to adjust its time to one of the nearby BTSs by following the longest arc (*i.e.* dephase of almost  $T$ ).

each BTS  $i$  as follows:

$$\begin{cases} f_i(\theta_1, \dots, \theta_n) &= \sum_{1 \leq j \leq n} \Delta_{ij}^2, \\ F(\theta_1, \dots, \theta_n) &= \sum_{1 \leq i, j \leq n} \Delta_{ij}^2 = \sum_{0 \leq i < n} f_i(\theta_1, \dots, \theta_n). \end{cases}$$

$F$  defines a surface in  $R^{n+1}$  and we are interested in its absolute minima. The simplest algorithm for finding a minimum consists in following the gradient  $\nabla F$  defined by:

$$\nabla F = \left( \frac{\partial F}{\partial \theta_1}, \dots, \frac{\partial F}{\partial \theta_{n-1}} \right).$$

In our case, we have:

$$\begin{aligned} \frac{\partial F}{\partial \theta_i} &= 2 \frac{\partial}{\partial \theta_i} \left[ \sum_{1 \leq j \leq n} (\theta_j - \theta_i)^2 \right] \\ &= 4 \sum_{1 \leq j \leq n} (\theta_j - \theta_i) \\ &= 4 \sum_{1 \leq j \leq n} \Delta_{ij} \end{aligned}$$

The Gradient Algorithm consists in iterating the following operation:

$$(\theta_1, \dots, \theta_n) \leftarrow (\theta_1, \dots, \theta_n) - \varepsilon \nabla F(\theta_1, \dots, \theta_n),$$

where  $\varepsilon$  is a tunable constant.

For a fixed BTS  $i$ , we can only obtain dephasing measures with a subset  $S_i$  of BTSs that we may call the neighbours of BTS  $i$  (those BTSs  $j$  for which the probability  $p_{i,j}$  is greater than some threshold). We will thus rather consider the following local cost function:

$$f_i^{loc} = \sum_{j \in S_i} \Delta_{ij}^2,$$

we thus define:

$$F^{loc} = \sum_{1 \leq i \leq n} f_i^{loc}.$$

The probability threshold should be large enough so that the considered dephasings are measured frequently enough and small enough so that the network to be synchronized remains connected with respect to the associated graph.

If  $i$  and  $j$  are two non-neighbour BTSs, we can still extrapolate the value of  $\Delta_{ij}$  with:

$$\widetilde{\Delta}_{ij} = \Delta_{ii_1} + \Delta_{i_1 i_2} + \dots + \Delta_{i_k j} \quad \text{mod } T,$$

where  $i_0 = i, i_1, \dots, i_{k+1} = j$  is a path from  $i$  to  $j$ . We then get:

$$\widetilde{\Delta}_{ij}^2 \leq n \times \max_{0 \leq l \leq k} \Delta_{i_l i_{l+1}}^2 \leq n F^{loc},$$

and thus

$$0 \leq F \leq n^3 F^{loc}.$$

**Algorithm 1** Gradient Algorithm for BTS  $i$

**Repeat**

*Every unit of time:*  
    ┌ Compute  $m = \frac{1}{|S_i|} \sum_{j \in S_i} \Delta_{ij}$ .  
    └ Dephase the clock of BTS  $i$  of  $\varepsilon m$  along the current unit of time.

If we achieve an absolute minimum according to the cost function  $F^{loc}$ , it also minimizes the cost function  $F$ .

The Gradient Algorithm then becomes:

$\varepsilon$  should be small enough so that it is possible to dephase a clock of  $\varepsilon m$  at the maximum dephasing speed  $v_m$ . In practise,  $\varepsilon m$  can be replaced by any continuous function of  $m$  increasing from  $-v_m \times \text{unit of time}$  to  $v_m \times \text{unit of time}$  and evaluating to 0 when  $m = 0$ .

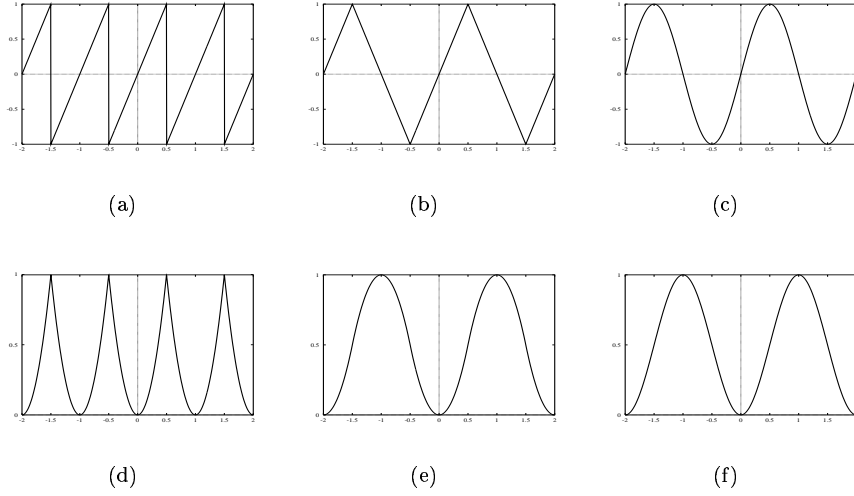
Each BTS runs this algorithm independently: there is no synchronization of each BTS unit of time, we may even choose different lengths for different BTSs units of time.

This algorithm surely minimizes the function  $F^{loc}$  since it is a simple quadratic function in the special case where the BTSs are approximately synchronized. Indeed, if there exists (virtually) a reference time such that all the  $\theta_i$  are in the range  $-T/4, T/4$ , this situation will remain true during the execution of the gradient algorithm. The BTS with the smallest time will always adjust forward and the BTS with the greatest time will always adjust backward. When these two BTSs get the same time, synchronization is achieved. This thus gives a partial solution to the problem, the difficult part consists in achieving the approximate synchronization: the Gradient Algorithm then allows to achieve and to maintain a complete synchronization. **This paper is mainly dedicated to proposing algorithms for achieving this approximate synchronization.**

We now come back to the general case. Each BTS gets some dephasing measures. We can still compute from these measures the shortest dephasing (it is the only value equaling this measure or its opposite modulo  $T$  in the range  $-T/2, T/2$ ). But this definition is far from insuring continuity in the general case, because a dephasing could increase up to  $T/2$  and then jump to  $-T/2$  (with a continuous evolution in reality). The Gradient Algorithm have little chance to minimize a non derivable function, it will rather oscillate never ending (considering the noise in the dephasing measures). To overcome this problem, we can consider the dephasing measures modulo  $2T$  and consider the cost function

$$F = \sum_{i,j} g(\Delta_{ij})^2, \text{ where } g(x) = 0 \text{ iff } x = 0$$

where  $g$  is some function insuring at least the derivability of  $F$ . Figure 4 gives some interesting  $g$  functions. This trick is possible because the dephasings are measured modulo  $8T$  in reality. Considering them as values modulo  $2T$  is thus no problem.



**Figure 4:** (a) discontinuous gradient of the cost function (d). (b) continuous gradient of the derivable cost function (e). (c) Sinusoidal gradient of the  $\mathcal{C}^\infty$  cost function (f). The x-coordinate spreads from  $-2T$  to  $2T$ . In (a), (b) and (c), the y-coordinate spreads from  $-T$  to  $T$ . In (d), (e) and (f), the y-coordinate spreads from  $0$  to  $T^2/2$ .

Unfortunately, starting from a random situation, the Gradient Algorithm is likely to be blocked in a deadlock, i.e. a local minimum of the cost function  $F^{loc}$  as in Figure 3.

Algorithm 1 supposes that at least a measure is made every unit of time (this is no problem if the unit of time is several minutes).

To make Algorithm 1 more reliable considering that there is some noise in the dephasing measures, we can use the mean of several recent measures of dephasings between BTS  $i$  and BTS  $j$  for  $\Delta_{ij}$ .

Recall that the Gradient Algorithm has now the form:

**Algorithm 2** Gradient Algorithm for BTS  $i$

**Repeat**

- Every unit of time:
  - Compute  $m = \frac{1}{|S_i|} \sum_{j \in S_i} g(\Delta_{ij})$ .
  - Dephase the clock of BTS  $i$  of  $\varepsilon m$  along the current unit of time.

This is a good solution when the BTSs are approximately synchronized. We still lack a solution for the initial phase: starting from a random situation, reach enough synchronization to launch the Gradient Algorithm.

### 3.2 Simulated Annealing

Simulated Annealing [5, 6] is a general method for finding absolute minima of a function rather than local minima. It can be seen as an enhancement of the Gradient Algorithm. The times  $\Theta = (\theta_1, \dots, \theta_n)$  of the BTSs are interpreted as the coordinates of a particle in a  $n$ -dimensional phase space. The particle is submitted to a potential given by the cost function and to a thermic energy of temperature  $\tau^t$ .  $t$  represents the time. The temperature is cooled with an appropriate function:  $\tau^0 = \infty$  and  $\tau^\infty = 0$ . The Metropolis Algorithm [6] allows to simulate the laws of Physics which imply that the particle will end up in an absolute minima of the potential if the temperature is cooled sufficiently slowly.

#### Algorithm 3 Metropolis Algorithm

##### Repeat

Every unit of time  $t$ :

- Choose at random  $\Xi = (\theta'_1, \dots, \theta'_n)$ .
- Make the particle pass from  $\Theta$  to  $\Xi$  ( $\Theta \leftarrow \Xi$ ) with probability  $e^{\frac{F(\Xi) - F(\Theta)}{k\tau^t}}$ .

The problem is that we cannot compute the cost function. First, we can use  $F^{loc}$  instead of  $F$  as previously. BTS  $i$  may decide to change its clock time from  $\theta_i$  to  $\theta_i \pm \varepsilon$  and it can compute

$$F^{loc}(\theta_1, \dots, \theta_i \pm \varepsilon, \dots, \theta_n) - F^{loc}(\theta_1, \dots, \theta_n) \simeq \pm \varepsilon \times 4 \sum_{j \in S_i} g(\Delta_{ij}).$$

We may now propose a Simulated Annealing Algorithm:

#### Algorithm 4 Simulated Annealing Algorithm for BTS $i$

##### Repeat every unit of time:

- Compute  $m = \frac{1}{|S_i|} \sum_{j \in S_i} g(\Delta_{ij})$ .
- Set with equal probability  $\alpha \leftarrow 1$  or  $\alpha \leftarrow -1$ .
- Let  $q$  be a random number between 0 and 1.
- If**  $q < e^{\frac{\alpha \varepsilon}{\tau}}$  **Then**
  - Dephase the clock of BTS  $i$  of  $\varepsilon m$  along the current unit of time.
  - $\tau \leftarrow \text{Cool}(\tau)$

The main problem in Simulated Annealing is the tuning of the cooling function Cool. Usually  $\tau^t = \tau^0 B^t$  is a good choice where  $B$  is a constant close to 1 (typically  $B = 0.99$  and thus  $\text{Cool}(\tau) = 0.99\tau$ ). There is still a problem here: the BTSs are not synchronized in their computations, each BTS must have its own temperature and begin the cooling process at a different moment. Moreover, if a BTS is booted, it should get the temperature of a neighbour BTS in order to avoid perturbing the algorithm.

### 3.3 Connected Components Techniques

In the previous algorithms, the choice of the unit of time determines a threshold over which nearby interfering BTSs are considered as neighbours. This defines a graph whose vertices are the BTSs. Two BTSs may be able to synchronize each other if and only if there exists a path from one to the other in this graph. A synchronization algorithm can be seen as a connected component algorithm: consider the connected component number of a BTS is its time. Two BTSs are in the same connected component if they are synchronized. A synchronization algorithm thus computes the connected components of this graph in some way. Notice that two BTSs have very little chance to be synchronized if no algorithm does something for that. This approach allows us to use the numerous connected components techniques for solving our problem. We will first focus on spanning tree techniques and then propose an adaptation of the classical PRAM algorithm for computing connected components.

#### Spanning tree techniques

Suppose we know a spanning tree of the graph: each BTS  $i$  except the root have a father BTS  $B_i$ . We can then use the following algorithm for synchronizing the BTSs.

**Algorithm 5** Spanning Tree Algorithm for BTS  $i$

*Repeat every unit of time:*

- ┌ Compute  $m = g(\Delta_{iB_i})$ .
- └ Dephase the clock of BTS  $i$  of  $\varepsilon m$  along the current unit of time.

This algorithm surely synchronizes all the BTSs to the root one but it is not fault tolerant. If a BTS  $i$  crashes, or if the edge  $iB_i$  disappears from the graph (a big new building grows between the two BTSs for example), all the associated subtrees will fail to synchronize to the rest of the BTSs. Another problem with this algorithm is the stability of long chains in the tree. The noise perturbing the measures may increase the instability along the chain.

This last problem can be overcome by mixing the gradient algorithm (which has good stability properties) and the previous one by inserting weights in the mean of the neighbours dephasings ( $\alpha$  is a small constant, 0.1 for example).

**Algorithm 6** Stable Spanning Tree Algorithm for BTS  $i$

*Repeat every unit of time:*

- ┌ Compute  $m = g(\Delta_{iB_i}) + \alpha \frac{1}{|S_i|-1} \sum_{j \in S_i, j \neq B_i} g(\Delta_{ij})$ .
- └  $m \leftarrow m / (1 + \alpha)$
- └ Dephase the clock of BTS  $i$  of  $\varepsilon m$  along the current unit of time.

The fault tolerance problem comes along with the problem of computing a spanning tree. A fixed spanning tree is not desirable since it should be computed

dynamically from time to time to be fault tolerant.

We can compute spanning forests by choosing a father for each BTS with a simple numerical criterium of one of the following kinds:

- give a large random number  $n(i)$  to each BTS  $i$  and set  $B_i \leftarrow k$  where  $n(k) = \min_{j \in S_i} n(j)$  (this number may be communicated to the neighbour BTSs by the control network at little cost),
- for each BTS  $i$ , measure the frequency  $f_{ij}$  with which the dephasing  $\Delta_{ij}$  is measured, and set  $B_i \leftarrow k$  where  $f_{ik} = \min_{j \in S_i} f_{ij}$ .

$1B_1, \dots, nB_n$  defines an oriented graph. The numerical criterium is decreasing along any chain of 3 BTSs (this assertion relies on the symmetry of the graph). This implies that there cannot be any cycle of length greater or equal than 3. The resulting graph is thus a spanning forest where the root of each tree is a cycle of length 2 (i.e. two BTSs that will surely synchronize to each other).

As this technique gives a spanning forest rather than a spanning tree, it may not solve the problem as there may be a deadlock between the tree components of the forest. But since it reduces the deadlock problem from the BTSs to bigger components, this may be enough to make deadlocks unprobable.

### Spanning directed graph

We can generalize the previous idea. We may synchronize the BTSs by considering only the directed edges of a spanning directed graph with no directed cycle of length greater or equal than 3 and only one maximal strongly connected component. Indeed, this means that all the strongly connected component do contain at most two BTSs. These strongly connected components of one or two BTSs will synchronize in a top bottom fashion to the maximal strongly connected component (recall that every directed graph induces a directed acyclic graph over its strongly connected components).

The problem is to compute such a spanning directed graph. We can imagine a scheme for that when the BTSs know their geographic position and the positions of their neighbours. Any BTS may then synchronize to all its north west neighbours... Changing the direction from time to time will make the scheme reliable to BTS faults.

### Parallel Connected Component Algorithm

We now adapt the algorithm proposed for the PRAM model by [7, 8]. Their algorithm computes growing connected sub-components represented by stars, *i.e.* a tree where all the non root nodes are children of the root. Stars are computed from classical spanning trees contracted with an operation called “pointer jumping” where a node replaces its father by its grandfather. This operation seems hard to implement in our framework since the dephasing measure with the grandfather may have to be computed from the dephasing measures between the node and its father and between the father and the grandfather. This would

generate many message passing between the BTSs, and this resource is very limited. The heart of the PRAM algorithm consists in “hooking” a subcomponent that is represented by a star to another subcomponent when an edge is detected between two nodes of the two subcomponents. Several hooking operations may attempt to hook a given subcomponent to different subcomponents, but only one operation succeeds.

We now introduce an algorithm based on this last idea. Any edge between two BTSs tries to synchronize one to the other. If it cannot succeed, the reason is that some other edge is trying to synchronize the subcomponent in the other direction. In that case the edge is inactivated (i.e. ignored) for a time  $U$  (long enough so that the BTS may synchronize itself according to the other edge). To keep the subcomponent notion, a BTS always synchronizes itself to its neighbours in the same subcomponent (i.e. those with which it has already been synchronized for a while). To get a stable algorithm, we can simulate all this by using the Gradient Algorithm and inserting weights in the mean computation. A BTS may have three types of neighbours, each type having a different weight:

- the inactive neighbours (which correspond to an inactivated edge) have a small weight ( $w_i = 0.1$  for example),
- the active neighbours (which correspond to an active edge) have a greater weight ( $w_a = 1$  for example),
- the subcomponent neighbours have an even greater weight ( $w_c = 2$  for example).

The associated sets of neighbours are respectively denoted by  $I_i$ ,  $A_i$  and  $C_i$  for each BTS  $i$ . See Algorithm 7.

Consider the edges connecting a subcomponent (i.e. a set of synchronized BTSs). If they tend to synchronize the subcomponent in contradicting directions (both forward and backward), the subcomponent will not dephase significantly but some edges will be deactivated one after another until no contradiction remains, the subcomponent will then dephase significantly (either forward or backward) and synchronize to another subcomponent. The inactivation time of an inactivated edge must be long enough to ensure that.

In theory, the dephasing speed of a subcomponent may be very slow. Consider for example a chain of  $k$  BTSs where one extremity is pulling all the others. The dephasing speed of the other extremity will be approximately  $\left(\frac{w_a}{w_c+w_a}\right)^k$  slower.  $U$  should thus increase exponentially with the diameter of the subcomponent of the BTS to ensure that one single edge may synchronize the subcomponent to another component.

In practise,  $U$  should be long enough to break the deadlock situations of the gradient like behaviour of the algorithm. Deadlocks occur along cycles. Due to their geographic positions, actual interference graphs of BTSs have a mesh structure which is often modeled with an hexagonal mesh. This implies that a deadlock situation occurs on concentric cycles and that the smallest cycle has

### Algorithm 7 Connected Components Algorithm

Initialize  $I_i$ ,  $A_i$  and  $C_i$  with  $\emptyset$ ,  $S_i$  and  $\emptyset$  respectively.

Repeat every unit of time:

```
    Compute  $m = \frac{w_c}{|C_i|} \sum_{j \in C_i} g(\Delta_{ij}) + \frac{w_a}{|A_i|} \sum_{j \in A_i} g(\Delta_{ij}) + \frac{w_i}{|I_i|} \sum_{j \in I_i} g(\Delta_{ij})$ .  
     $m \leftarrow m / (w_c + w_a + w_i)$   
    Dephase the clock of BTS  $i$  of  $\varepsilon m$  along the current unit of time.  
     $\Delta_{ij}^{old}$  denotes the estimation of the dephasing with BTS  $j$  at the previous  
    unit of time.  
    For each  $j \in A_i$  do  
        If  $\Delta_{ij} \geq \Delta_{ij}^{old}$  Then  
            Remove  $j$  from  $A_i$  and put it in  $I_i$ .  
        Else  
             $\Delta_{ij}^{old} \leftarrow \Delta_{ij}$   
        If  $|\Delta_{ij}| \leq \beta$  Then put  $j$  in  $C_i$ .  
    For each  $j \in I_i$  do  
        If  $j$  has been put in  $I_i$  for a time greater than  $U$ , remove it from  $I_i$   
        and put it in  $A_i$  with  $\Delta_{ij}^{old} \leftarrow \infty$ .
```

a bounded small length. Hence it is sufficient in practice to tune  $U$  as if the diameter of the subcomponent was less than 6.

Conversely to the PRAM algorithm, we cannot compute a spanning tree from our algorithm. In the PRAM algorithm, the collection of the edges that succeed in “hooking” two subcomponents form an undirected spanning tree. In our case, we do not know which edge succeeds in synchronizing two subcomponents as there may be several such edges.

## Conclusion

We have introduced some new algorithmic ideas for the problem of BTS synchronization. The algorithms introduced in this paper have been simulated. The Connected Component Algorithm appears to be more reliable than the Simulated Annealing Algorithm: it has always succeeded in synchronizing the network of BTSs (somehow randomized graphs of bounded degree where used in the simulations) and the simulated algorithm sometimes fails (it appears to be difficult to tune since the graph between the BTSs is not known in advance).

Interesting future work resides in the probabilistic study of the forests generated by the methods described in the spanning tree techniques section. Information about the number of trees, the depth of the trees could be useful.

## References

- [1] E.H.L. Aarts and P.J.M. Van Laarhoven. *Simulated Annealing : Theory and Applications*. New York Wiley, 1987.
- [2] R. Azencott. *Simulated annealing : parallelization techniques*. New York Wiley, 1992.
- [3] O. Catoni. Applications of sharp large deviation estimates to optimal cooling schedules. *Ann. Inst. Henri Poincaré, Probabilités et statistiques*, 27(4), 1991.
- [4] A. Denise. *Méthodes de génération aléatoire d'objets combinatoires de grande taille et problèmes d'énumération*. PhD thesis, Université Bordeaux I, janvier 1994.
- [5] D. Geman and S. Geman. Stochastic relaxation, Gibbs distribution, Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6:721–741, 1984.
- [6] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–686, 1983.
- [7] C. Kruskal, L. Rudolph, and M. Snir. The power of parallel prefix. In *Internat. Conf. Parallel Processing*, pages 180–184, 1985.
- [8] J.H. Reif. *Synthesis of parallel algorithms*. Morgan Kaufmann Publishers, 1992.
- [9] J. Siarry and G. Dreyfus. *La méthode du recuit simulé*. Paris IDSET, 1989.