# Resource Allocation on Sparse Graphs

MATHIEU COLLAS

MSc by Research in Pattern Analysis and Neural Networks

ASTON UNIVERSITY

September 2005

ASTON UNIVERSITY

# Resource Allocation on Sparse Graphs

MATHIEU COLLAS

MSc by Research in Pattern Analysis and Neural Networks, 2005

**Thesis Summary**

Considering a network of workers represented by a sparse graph where each worker (i.e. node) has a specific load of small independent jobs, our aim is to move the jobs around such that all tasks will be carried out while the communication is minimized. We examine the performance of a new algorithm based on message passing methods in comparison with a standard quadratic programming based algorithm.

**Keywords:** Resource allocation, Flow minimization, Sparse graph, Graph optimization, Message passing

# Acknowledgements

I would like to thank all the people who share this year with me and help me through this:

- My supervisor, David Saad, for his help and all the time he spent with me to solve my problems
- My second supervisor, Gabriele Migliorini
- All the staff of the NCRG who welcome us among them

This year would not have been the same if some people would have not been here, I speak about:

- My family for the support they gave me all year long
- My girlfriend *pour son amour et ce qu'elle est*
- All the 'frenchies':
  - Mathieu Chatelain
  - Etienne Mallard
  - Boris Dubuisson
  - Jean Nicolas Turlier
- Amy Rostron for her incredible accent
- Stéphane Bounkong for helping me to tame MPI
- All the PHDs, who were always here to speak and help us

# Contents

# List of Figures

# Chapter 1

# Resource Allocation on Sparse Graphs

## 1.1 What is the problem

The problem we are focusing on is "Resource allocation on sparse graphs", so, first of all, we have to define what a sparse graph is.

Informally, a graph with relatively few links with respect to the number of nodes is sparse, and a graph with many links is dense (scaling in $O(|V|^2)$ with the number of nodes). But more precisely, it is usually defined that a sparse graph is a graph where the total number of links is scaling in $O(|V|)$ with the number of nodes. For a Graph $G$ with Vertices $V$ and Edges $E$, $G = (V, E)$, we have:

- $G$ is a sparse graph if $|E| = O(|V|)$
- $G$ is a dense graph if $|E| = O(|V|^2)$

The resource allocation problem is a well known problem [1, 2], the aim is to distribute a decomposable task onto different working nodes with different work capacities.

In our case, the nodes of the graph are the workers, each of them has a particular work capacity and initial task, and we want to distribute the jobs (the whole task is already decomposed into small independent jobs) among the nodes such that the flow is minimized and all tasks are carried out.

So we start in a state where all node have pre-assigned jobs. Some of the nodes are saturated (i.e. their tasks exceed their capacity), some of them have some free working capacity.

For example, figure 1.1 represents a graph in the initial state. In figure 1.2 we see the same graph solved with the flow indicated on the edges, and figure 1.3 shows us the same graph solved with the capacity of each node computed. We see that there are no saturated nodes anymore and that the flow needed to solve the graph is minimized.

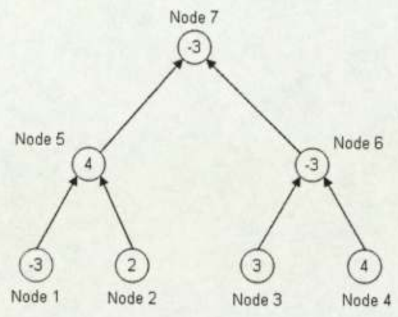

Figure 1.1: Example of a graph in the initial state.

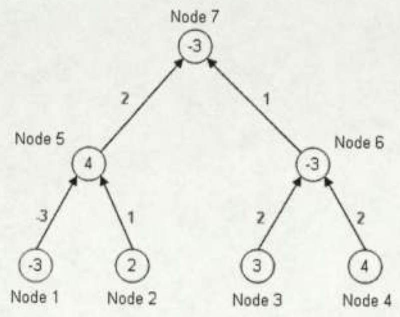

Figure 1.2: The same graph solved with original capacities and flow indicated.

## 1.2 Practical problems

As our world is mainly based on networks (power, water, roads, Internet, aso...), it is easy to find a lot of practical applications:

- Provisioning store network by minimizing the items moves between storage points and stores while meeting the demand.
- Peer to peer network: sharing files among computers while minimizing network traffic.
- Electrical network: provisioning end users by minimizing electrical flow between power stations while meeting the demand.
- Water network: provisioning end users by minimizing water flow between water storage points.
- And so on...

in each of these examples, the element that is costly and that we want to minimize is the flow.

## 1.3 Existing methods

The resource allocation problem is a well known problem in the area of distributed computing [1, 2] to which significant effort has been dedicated with the computer sci-

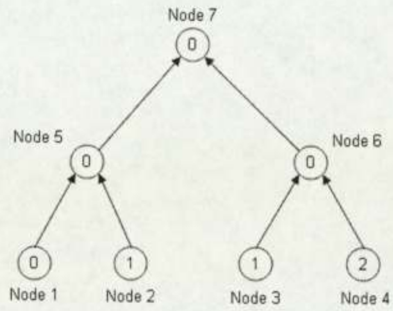

Figure 1.3: The same graph solved.

ence community.

The problem is quite general and can be applied, as we have seen, to a lot of practical problems. I didn't find any generic method for solving the "resource allocation problem". I have identified approximation methods to solve problems like Vehicle Routing Problem (VRP) or Traveling Salesman Problem (TSP), but they are different from the current problem.

The problem we are addressing here is more generic, it is why no existing method can be cited here.

## 1.4 Why are we looking for a new method ?

All methods used to solve specific application of the "resource allocation problem" are global optimization methods.

The main problem of global optimization methods is that the computational cost to solve the problem scales in a cubical way with the system size. So it is clear that it will be impossible to solve large resource allocation problems with global optimization methods.

On the other hand, all these methods are directly derived from the cost function they want to minimize. In our case, the cost function is directly based on the flow, which is very simple. But for more complex cases, it may be interesting to be able to have more complex cost functions. With complex cost functions, it will be harder, or even impossible, to derive a global optimization method to solve the problem.

These two drawbacks of global optimization methods motivate us to find a new solution to resource allocation problems.

## 1.5 How will we work ?

In our study, we are interested to compare the performances of two different algorithms (which will be derived in chapter 2).

As we want to compare a very generic algorithm, the message passing algorithm, against an optimized algorithm, the quadratic programming based algorithm, we will use a very simple cost function in order to see the differences of performances of the two algorithms on a very simple case.

So, for our study, we will use a very basic cost function : the sum of the square of the flow moving through the graph.

By studying the two algorithms on such a simple case, we will be able to focus more easily on their efficiency and we will be able to see which one is better than the other and why.

One drawback may be that by using a too simple cost function we will miss some limitations in the algorithms, but as we will see later, only the quadratic programming based algorithm will have limitations, the message passing algorithm is a lot more flexible, and may be applied on almost every type of cost function.

So in the next chapter, we will derive the two algorithms from the same cost function, we will then be able to make all the experiments we want in order to discuss on the performances of the two algorithms.

# Chapter 2

# Two Different Approaches

This chapter is based on the work of Michael Wong and David Saad in different personal communications [3, 4] and a preprinted article [5].

## 2.1 Definition of the cost function

### 2.1.1 An example

*Work in progress*

### 2.1.2 Steady state of message passing

*Work in progress*

### 2.1.3 Definition of the cost function

We can now precisely define our cost function thanks to the preceding example.

Consider the message from node $j$ to node $i$ after receiving a message from node $k \in N_j^{\setminus\{i\}}$, where $N_j$ is the set of neighboring nodes of $j$, as shown in figure 2.3. The messages are $(\partial E_k^{\setminus j}/\partial y_{jk}, \partial^2 E_k^{\setminus j}/\partial y_{jk}^2) \equiv (a_{jk}, b_{jk})$, where $b_{jk}$ is positive.

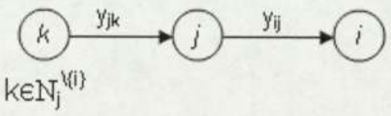

Figure 2.1: The neighboring of node $j$.

Suppose the current from node $j$ to node $i$ increases by $\varepsilon_{ij}$. We are interested to find the current change $\varepsilon_{jk}$ from node $k$ to node $j$ by minimizing the energy:

$$E_j^{\setminus i} = \sum_{k\in N_j^{\setminus\{i\}}} \left[ a_{jk}\varepsilon_{jk} + \frac{1}{2} b_{jk}\varepsilon_{jk}^2 + (y_{jk}+\varepsilon_{jk})^2 \right]$$

subject to:

$$\sum_{k\in N_j^{\setminus\{i\}}} (y_{jk}+\varepsilon_{jk}) + \Lambda_j \geq y_{ij} + \varepsilon_{ij},$$

where $\Lambda_j$ is the capacity of node $j$.

## 2.2 Quadratic programming based solution

### 2.2.1 The chemical potential in a tree

Consider the message from node $j$ to node $i$ after receiving a message from node $k \in N_j^{\setminus\{i\}}$, where $N_j$ is the set of neighboring nodes of $j$, as shown in figure 2.3. The messages are $(\partial E_k^{\setminus j}/\partial y_{jk}, \partial^2 E_k^{\setminus j}/\partial y_{jk}^2) \equiv (a_{jk}, b_{jk})$, where $b_{jk}$ is positive.

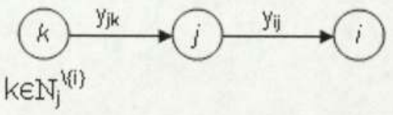

Figure 2.2: The neighboring of node $j$.

Consider the effects of a change in the current from node $j$ to node $i$. We are interested to find the current change $\varepsilon_{jk}$ from node $k$ to node $j$ by minimizing the energy:

$$E_j^{\setminus i} = \sum_{k\in N_j^{\setminus\{i\}}} \left[ a_{jk}\varepsilon_{jk} + \frac{1}{2} b_{jk}\varepsilon_{jk}^2 + (y_{jk}+\varepsilon_{jk})^2 \right] \tag{2.1}$$

subject to:

$$\sum_{k\in N_j^{\setminus\{i\}}} (y_{jk}+\varepsilon_{jk}) + \Lambda_j \geq y_{ij}, \tag{2.2}$$

where $\Lambda_j$ is the capacity of node $j$.

Introducing the Lagrange multiplier, the function to be minimized is:

$$\begin{aligned} L &= \sum_{k\in N_j^{\setminus\{i\}}} \left[a_{jk}\varepsilon_{jk} + \frac{1}{2}b_{jk}\varepsilon_{jk}^2 + (y_{jk}+\varepsilon_{jk})^2\right] \\ &+ \mu_{ij}\left[\sum_{k\in N_j^{\setminus\{i\}}} (y_{jk}+\varepsilon_{jk}) + \Lambda_j - y_{ij}\right], \end{aligned} \tag{2.3}$$

where at least one of the two factors in the last term vanishes, and the Lagrange multiplier $\mu_{ij}$ is non-positive. The optimal solution is described in the following two cases:

- **Case 1**: $\sum_k (b_{jk}y_{jk} - a_{jk})(1+b_{jk})^{-1} + \Lambda_j - y_{ij} < 0$
  The change in the current is:

$$\varepsilon_{jk} = -\frac{y_{jk}+a_{jk}+\mu_{ij}}{1+b_{jk}}, \tag{2.4}$$

  where

$$\mu_{ij} = \frac{\sum_k (b_{jk}y_{jk} - a_{jk})(1+b_{jk})^{-1} + \Lambda_j - y_{ij}}{\sum_k (1+b_{jk})^{-1}}. \tag{2.5}$$

  The message from $j$ to $i$ is:

$$a_{ij} = -\mu_{ij}, \tag{2.6}$$

$$b_{ij} = \frac{1}{\sum_k (1+b_{jk})^{-1}}. \tag{2.7}$$

  At the steady state, $\varepsilon_{jk}=0$. So using equation 2.4,

$$\mu_{ij} = -y_{jk} - a_{jk} \tag{2.8}$$

  for all $k \in N_j^{\setminus\{i\}}$. Note that $\mu_{ij}$ is independent of the node $i$ to which the message is fed. This enable us to write $\mu_{ij} = \mu_j$. Similarly, using equation 2.6 $a_{jk} = -\mu_{jk} = -\mu_k$, equation 2.8 then becomes:

$$\mu_j - \mu_k = -y_{jk}. \tag{2.9}$$

So we can interpret $\mu_j$ as the chemical potential at node $j$, and current flows from high to low potential. Furthermore, the equality in equation 2.2 implies that:

$$\mu_j = \frac{1}{C}(\sum_{k\in N_j} \mu_k + \Lambda_j), \tag{2.10}$$

where $C$ is the total connectivity of a node.

- **Case 2**: $\sum_k (b_{jk}y_{jk} - a_{jk})(1+b_{jk})^{-1} + \Lambda_j - y_{ij} \geq 0$
  The change in the current is:

  $$\varepsilon_{kj} = -\frac{y_{jk} + a_{jk}}{1+b_{jk}}. \tag{2.11}$$

  The message from $j$ to $i$ is $a_{ij} = b_{ij} = 0$. At the steady state, $\varepsilon_{jk} = 0$. Using equation 2.11,

  $$0 = -y_{jk} - a_{jk}. \tag{2.12}$$

  Hence we can interpret that the chemical potential at node $j$ is $\mu_j = 0$. Then we arrive at equation 2.9 again.

- **Summary**: Summarizing cases 1 and 2, we have:

  $$\mu_j = \min\left(0, \frac{1}{C}(\sum_{k\in N_j} \mu_k + \Lambda_j)\right). \tag{2.13}$$

### 2.2.2 The chemical potential in the presence of terminal nodes

We just have derived the algorithm in generic cases, we now have to see how it will work in presence of terminal nodes.

Consider the message from node $j$ to node $i$ after receiving messages from nodes $k \in N_j^{\backslash\{i\}}$, where $N_j$ is the set of neighboring nodes of $j$, with message $(\partial E_k^{\backslash j}/\partial y_{jk}, \partial^2 E_k^{\backslash j}/\partial y_{jk}^2) \equiv (a_{jk}, b_{jk})$, and terminal nodes $l \in T_j^{\backslash\{i\}}$. We are interested to find the current change $\varepsilon_{jk}$ from node $k$ to node $j$ by minimizing the energy:

$$E_j^{\backslash i} = \sum_{k\in N_j^{\backslash\{i\}}} \left[a_{jk}\varepsilon_{jk} + \frac{1}{2}b_{jk}\varepsilon_{jk}^2 + (y_{jk}+\varepsilon_{jk})^2\right] + \sum_{l\in T_j^{\backslash\{i\}}} (y_{jl}+\varepsilon_{jl})^2 \tag{2.14}$$

subject to:

$$\sum_{k\in N_j^{\backslash\{i\}}} (y_{jk}+\varepsilon_{jk}) + \Lambda_j \geq y_{ij} \tag{2.15}$$

and

$$\Lambda_l \geq y_{jl} + \varepsilon_{jl} \tag{2.16}$$

for $l \in T_j^{\setminus\{i\}}$.

Introducing Lagrange multipliers, the function to be minimized is:

$$\begin{aligned} L &= \sum_{k \in N_j^{\setminus\{i\}}} \left[ a_{jk}\varepsilon_{jk} + \frac{1}{2} b_{jk}\varepsilon_{jk}^2 + (y_{jk} + \varepsilon_{jk})^2 \right] \\ &+ \mu_{ij} \left[ \sum_{k \in N_j^{\setminus\{i\}}} (y_{jk} + \varepsilon_{jk}) + \Lambda_j - y_{ij} \right] \\ &+ \sum_{l \in T_j^{\setminus\{i\}}} \lambda_l \left[ \Lambda_l - y_{jl} - \varepsilon_{jl} \right], \end{aligned} \tag{2.17}$$

where at least one of the two factors in the Lagrange multiplier terms vanishes, and the Lagrange multipliers $\mu_{ij}$ and $\lambda_l$ are non-positive. The optimal solution is given by:

$$\mu_j - \mu_k = -y_{jk}, \tag{2.18}$$

where

$$\mu_l = \min(0, \mu_j + \Lambda_l) \tag{2.19}$$

for terminal nodes, and

$$\mu_j = \min \left( 0, \frac{1}{C} ( \sum_{k \in N_j \cup T_j} \mu_k + \Lambda_j ) \right) \tag{2.20}$$

for non terminal nodes.

Note that equation 2.19 is a special case of equation 2.20, since $C = 1$ for the terminal nodes.

### 2.2.3 Loopy networks

The above formulation is applicable to treelike networks. The question remains open for networks containing loops. However, we can find an even simpler proof for loopy networks as follows. Using the convention that $y_{ij} = -y_{ji}$, our task is to minimize the energy:

$$E = \sum_{(ij)} \frac{1}{2} y_{ij}^2, \tag{2.21}$$

where the summation is taken over all links (ij). The minimization is subject to:

$$\sum_{j \in N_i} y_{ij} + \Lambda_i \geq 0 \tag{2.22}$$

for all $i$.

Introducing Lagrange multipliers, the function to be minimized is:

$$L = \sum_{(ij)} \frac{1}{2} y_{ij}^2 + \sum_i \mu_i \left( \sum_{j \in N_i} y_{ij} + \Lambda_i \right). \tag{2.23}$$

Optimizing with respect to $y_{ij}$ or equivalent $-y_{ji}$, we obtain:

$$y_{ij} = \mu_j - \mu_i. \tag{2.24}$$

And from the constraint 2.22, we get:

$$\mu_i = \min \left( 0, \frac{1}{C} (\sum_{j \in N_i} \mu_j + \Lambda_i) \right). \tag{2.25}$$

### 2.2.4 Updating algorithm

Starting from the energy function:

$$E_j^{\setminus i} = \sum_{k \in N_j^{\setminus \{i\}}} \left[ a_{jk} \varepsilon_{jk} + \frac{1}{2} b_{jk} \varepsilon_{jk}^2 + (y_{jk} + \varepsilon_{jk})^2 \right], \tag{2.26}$$

we have found using Lagrange multipliers that, in presence of terminal nodes or not, and for any network (loopy or not), our update equation is simply:

$$\mu_j = \min \left( 0, \frac{1}{C} (\sum_{k \in N_j} \mu_k + \Lambda_j) \right), \tag{2.27}$$

where $\Lambda_j$ is the capacity of the node $j$ and $C$ its connectivity.

This huge simplification is only possible because of the simplicity of the cost function and the usage of the Lagrange multipliers. It is not possible to find such a simple update equation from any cost function.

## 2.3 Message passing solution

We will now see how the message passing algorithm is derived, firstly in generic case, and then in presence of terminal nodes.

### 2.3.1 Message updating algorithm

Consider the message from node $j$ to node $i$ after receiving a message from node $k \in N_j^{\setminus \{i\}}$, where $N_j$ is the set of neighboring nodes of $j$, as shown in figure 2.3. The messages are $(\partial E_k^{\setminus j} / \partial y_{jk}, \partial^2 E_k^{\setminus j} / \partial y_{jk}^2) \equiv (a_{jk}, b_{jk})$, where $b_{jk}$ is positive.

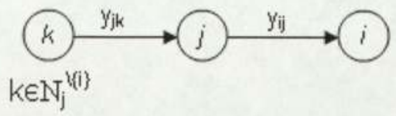

Figure 2.3: The neighboring of node $j$.

Suppose the current from node $j$ to node $i$ increases by $\varepsilon_{ij}$. We are interested to find the current change $\varepsilon_{jk}$ from node $k$ to node $j$ by minimizing the energy:

$$E_j^{\setminus i} = \sum_{k \in N_j^{\setminus\{i\}}} \left[ a_{jk}\varepsilon_{jk} + \frac{1}{2} b_{jk}\varepsilon_{jk}^2 + (y_{jk} + \varepsilon_{jk})^2 \right]$$

subject to:

$$\sum_{k \in N_j^{\setminus\{i\}}} (y_{jk} + \varepsilon_{jk}) + \Lambda_j \geq y_{ij} + \varepsilon_{ij},$$

where $\Lambda_j$ is the capacity of node $j$.

Introducing the Lagrange multiplier, the function to be minimized is:

$$\begin{aligned} L &= \sum_{k \in N_j^{\setminus\{i\}}} \left[ a_{jk}\varepsilon_{jk} + \frac{1}{2} b_{jk}\varepsilon_{jk}^2 + (y_{jk} + \varepsilon_{jk})^2 \right] \\ &- \mu_{ij} \left[ \sum_{k \in N_j^{\setminus\{i\}}} (y_{jk} + \varepsilon_{jk}) + \Lambda_j - y_{ij} - \varepsilon_{ij} \right], \end{aligned}$$

where at least one of the two factors in the last term vanishes, and the Lagrange multiplier $\mu_{ij}$ is non-negative. The optimal solution is described in the following two cases:

- **Case 1**: $y_{ij} - \Lambda_j + \sum_k (a_{jk} - b_{jk}y_{jk})(2 + b_{jk})^{-1} > 0$
  The currents $y_{jk}$ are updated to:

  $$y_{jk} + \varepsilon_{jk} = \frac{\mu_{ij} - a_{jk} + b_{jk}y_{jk}}{2 + b_{jk}},$$

  where

  $$\mu_{ij} = \frac{y_{ij} - \Lambda_j + \sum_k (a_{jk} - b_{jk}y_{jk})(2 + b_{jk})^{-1}}{\sum_k (2 + b_{jk})^{-1}}.$$

  The message from node $j$ to $i$ is updated to:

  $$\begin{aligned} a_{ij} &= \mu_{ij}, \\ b_{ij} &= \frac{1}{\sum_k (2 + b_{jk})^{-1}}. \end{aligned}$$

- **Case 2**: $y_{ij} - \Lambda_j + \sum_k (a_{jk} - b_{jk}y_{jk})(2+b_{jk})^{-1} \leq 0$
  The currents $y_{jk}$ are updated to:
  $$y_{jk} + \varepsilon_{jk} = \frac{-a_{jk} + b_{jk}y_{jk}}{2+b_{jk}}.$$
  The message from node $j$ to $i$ is updated to $a_{ij} = b_{ij} = 0$.

## 2.3.2 In presence of terminal nodes

We just have derived the algorithm in generic cases, we now have to see how it will work in presence of terminal nodes.

Consider the message from node $j$ to node $i$ after receiving a message from node $k \in N_j^{\backslash\{i\}}$, where $N_j$ is the set of neighboring nodes of $j$, with messages $(\partial E_k^{\backslash j}/\partial y_{jk}, \partial^2 E_k^{\backslash j}/\partial y_{jk}^2) \equiv (a_{jk}, b_{jk})$, where $b_{jk}$ is positive, and terminal nodes $l \in T_j^{\backslash\{i\}}$. We are interested to find the current change $\varepsilon_{jk}$ from node $k$ to $j$ by minimizing the energy:

$$E_j^{\backslash i} = \sum_{k \in N_j^{\backslash\{i\}}} \left[ a_{jk}\varepsilon_{jk} + \frac{1}{2} b_{jk}\varepsilon_{jk}^2 + (y_{jk} + \varepsilon_{jk})^2 \right] + \sum_{l \in T_j^{\backslash\{i\}}} (y_{jl} + \varepsilon_{jl})^2$$

subject to:

$$\sum_{k \in N_j^{\backslash\{i\}}} (y_{jk} + \varepsilon_{jk}) + \Lambda_j \geq y_{ij} + \varepsilon_{ij}$$

and

$$\Lambda_l \geq y_{jl} + \varepsilon_{jl}$$

for $l \in T_j^{\backslash\{i\}}$.

Introducing Lagrange multipliers, the function to be minimized is:

$$\begin{aligned} L &= \sum_{k \in N_j^{\backslash\{i\}}} \left[ a_{jk}\varepsilon_{jk} + \frac{1}{2} b_{jk}\varepsilon_{jk}^2 + (y_{jk} + \varepsilon_{jk})^2 \right] \\ &- \mu_{ij} \left[ \sum_{k \in N_j^{\backslash\{i\}}} (y_{jk} + \varepsilon_{jk}) + \Lambda_j - y_{ij} - \varepsilon_{ij} \right] \\ &- \sum_{l \in T_j^{\backslash\{i\}}} \lambda_l \left[ \Lambda_l - y_{jl} - \varepsilon_{jl} \right], \end{aligned}$$

where at least one of the two factors in the Lagrange multiplier terms vanishes, and the Lagrange multipliers $\mu_{ij}$ and $\lambda_l$ are non-negative. The optimal solution is described in the following two cases:

- **Case 1**: $y_{ij} - \Lambda_j + \sum_{k \in N_j^{\backslash\{i\}}} (a_{jk} - b_{jk}y_{jk})(2+b_{jk})^{-1} - \sum_{l \in T_j^{\backslash\{i\}}} \Lambda_l \Theta(\mu_{ij} - 2\Lambda_l) > 0$
  The currents $y_{jk}$ from the non-terminal nodes are updated to:
  $$y_{jk} + \varepsilon_{jk} = \frac{\mu_{ij} - a_{jk} + b_{jk}y_{jk}}{2+b_{jk}},$$

where

$$\mu_{ij} = \frac{y_{ij} - \Lambda_j + \sum_{k\in N_j^{\setminus\{i\}}}(a_{jk} - b_{jk}y_{jk})(2+b_{jk})^{-1} - \sum_{l\in T_j^{\setminus\{i\}}}\Lambda_l\Theta(\mu_{ij} - 2\Lambda_l)}{\sum_{k\in N_j^{\setminus\{i\}}}(2+b_{jk})^{-1} + \sum_{l\in T_j^{\setminus\{i\}}}\Theta(2\Lambda_l - \mu_{ij})2^{-1}}.$$

The currents $y_{jl}$ from the terminal nodes are updated to:

$$y_{jl} + \varepsilon_{jl} = \min(\frac{1}{2}\mu_{ij}, \Lambda_l).$$

The message from $j$ to $i$ is updated to :

$$\begin{aligned} a_{ij} &= \mu_{ij}, \\ b_{ij} &= \frac{1}{\sum_{k\in N_j^{\setminus\{i\}}}(2+b_{jk})^{-1} + \sum_{l\in T_j^{\setminus\{i\}}}\Theta(2\Lambda_l - \mu_{ij})2^{-1}}. \end{aligned}$$

- **Case 2**: $y_{ij} - \Lambda_j + \sum_{k\in N_j^{\setminus\{i\}}}(a_{jk} - b_{jk}y_{jk})(2+b_{jk})^{-1} - \sum_{l\in T_j^{\setminus\{i\}}}\Lambda_l\Theta(\mu_{ij} - 2\Lambda_l) \leq 0$
  The currents $y_{jk}$ from the non-terminal nodes are updated to:

  $$y_{jk} + \varepsilon_{jk} = \frac{-a_{jk} + b_{jk}y_{jk}}{2 + b_{jk}}.$$

  The currents $y_{jl}$ from the terminal nodes are updated to:

  $$y_{jl} + \varepsilon_{jl} = \min(0, \Lambda_l).$$

  The message from $j$ to $i$ is updated to $a_{ij} = b_{ij} = 0$.

## 2.4 Synopsis

We have derived the quadratic programming based and the message passing algorithms from the same cost function.

The quadratic programming based algorithm is derived from an optimization specific to our cost function, while the message passing algorithm stay generic.

So we will be able to see how the generic solution based on the message passing algorithm performs against a optimized solution: the quadratic programming based algorithm.

# Chapter 3

# Test Protocol

As our aim is to see how the message passing algorithm performs in comparison with the quadratic programming based algorithm, we have to define a test protocol to study the performance of the two algorithms.

## 3.1 The free parameters

Firstly, we have to generate a data set (i.e. a graph set) on which our algorithms will run. Before generating our data set, we have to define which parameters of the graph we want to vary. These parameters are important and should be chosen carefully because we will study how the two algorithms scale against each free parameter.

The free parameters we will concentrate on are:

- **Number of nodes in the graph**: it will tell us how the algorithms scale with the system size. This parameter is very important because the message passing algorithm is a local algorithm, and must scale linearly with the system size.
- **Capacities of the nodes**: it will tell us how the algorithms scale with the increase of average constraint in the system. If we have a high overall capacity, the problem will be easy to solve (only a few jobs will have to move), but if we take an overall capacity near to zero, the problem becomes a lot harder to solve (a lot of jobs have to move, sometimes through most of the graph).
- **Connectivity of the nodes**: it will tell us again how the algorithms scale with another constraint in the system. With high connectivity, a saturated node has a high probability of being connected to an unsaturated node, so the graph will be easy to solve. Having low connectivity, the algorithm will have to move jobs far away to find an unsaturated node.

By exploring how the two algorithms scale against each of the free parameters, we will be able to understand the properties of each algorithm.

Secondly, we have to choose the method to use to generate the graphs. As using only one method would not be representative (we have to make sure the algorithms are generic, so we have to check how our results depend on the graph characteristics), we will use two different methods for generating the graphs.

- **Poissonian graphs**: links between nodes are generated at random with some probability giving rise of a Poisson distribution for the connectivity of individual nodes. Figure 3.1 shows a typical Poisson distribution.

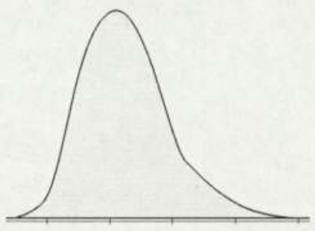

Figure 3.1: Example of a Poisson distribution.

- **Linear graphs**: links between nodes are generated using a linear combination between two connectivity values, with some ratio between them. Figure 3.2 demonstrate a linear distribution which combine two connectivity values.

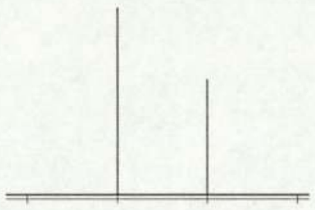

Figure 3.2: Example of a linear distribution.

With the three free parameters and the two different graph generation methods, we will have a good set of graphs for testing the algorithms.

## 3.2 Generation of the graphs

Once all the free parameters have been defined, I have developed a graph generator. As it was created specifically for this project, this graph generator corresponds exactly as what we want and what we need.

I have defined all the ranges which will be used for each free parameter to generate our graph set:

- **Number of nodes in the graph**: the system size will vary from 1,000 nodes to 10,000 nodes by step of 300.

- **Capacities of the nodes**: the capacities of the nodes will be generated from a gaussian with a fixed variance of 1 and a mean varying from 0.01 to 0.2 by step of 0.01 and then from 0.2 to 1 by step of 0.1.
- **Connectivity of the nodes**:
  - **Poissonian graphs**: the connectivity of the nodes will be generated to give rise of a Poisson distribution from mean 3 to 6 by step of 1.
  - **Linear graphs**: the connectivity of the nodes will be generated using connectivity values from 3 to 6 with coefficients which will simulate steps of 0.5.

All these ranges have been defined to give us a good panel of different kind of graphs, so we will be able to understand the properties of the algorithms thanks to different cases.

We will generate 100 graphs for each set of parameters (i.e. defined values for all the free parameters), so we will be able to have strong results. It will also give us information about the reliability of a specific result; if the result is far from the mean of the 100 results, it will not be very reliable. So we will be able to define error bars on our results which will be very useful to see the stability of each algorithm.

Firstly, all the graphs (100 graphs for each set of free parameters) will be generated and stored, this will be our graph set. Secondly our two algorithms - quadratic programming based and message passing algorithms - will be ran on it. It make us sure that the results of our experiments will be comparable, because the two algorithms will be run on the same graph set.

## 3.3 Running the algorithms

As said, we will run the two algorithms on the same graph set. All the results of the experiments will be kept, and we will only focus on the median of all the results for each set of parameters:

- **Mean**: simply the sum of the results divided by the number of results. It would give us a good vision of the results of the experiments, but unfortunately, it varies a lot when a specific result is far from the mean.
- **Median**: the value of the middle of the sorted results. It will give us a more stable vision of the results of the experiments because it is stronger than the mean when a specific result is far from the median. It is why we will only focus on the median for our understanding of the algorithms.

We will also able to define error bars for our results, defined from the standard deviation - square root of the sum of all the squared differences between a result and the mean value of all the results - it will let us see the stability of each algorithm.

# Chapter 4

# Simulation Results

## 4.1 Presentation

In this chapter, I will present the simulation results and try to interpret them to understand how each algorithm is correlated with each parameter of the graphs.

I will often talk about 'iteration'. An iteration is an update of all nodes of the graph. For example, if it takes 3 iterations to the message passing algorithm to solve a graph, it means that each node of the graph have been updated 3 times by the message passing algorithm.

For our study, more iterations an algorithm needs to solve a graph, less efficient it is.

## 4.2 About the errors

Earlier, I said we were able to know the errors of the simulations based on the standard deviation. During my experiments, all the results were always very stable and the errors not high at all:

- Never more than 4% for the quadratic programming based algorithm
- Never more than 7% for the message passing algorithm

The biggest values of the errors are for the very hard to solve graphs, for the graphs which are easier to solve, the error bars are very close to the median results.

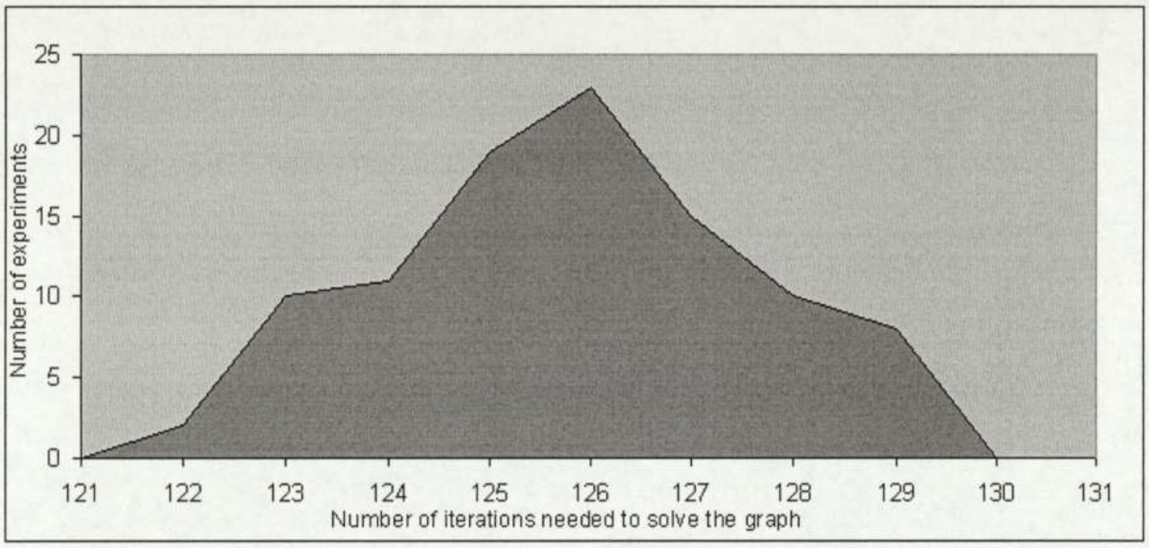

Figure 4.1: Distribution of the results of the 100 experiments to solve a Poissonian graph of mean 4, capacity mean 0.05 and 4,000 nodes with the quadratic programming based algorithm.

It is a good point because it says that our test protocol and simulation results are precise.

We can see in figures 4.1 and 4.2 the distributions of the results of different experiments. In all these figures, we can see that the distributions are clean, which mean that our simulation results are precise and of confidence.

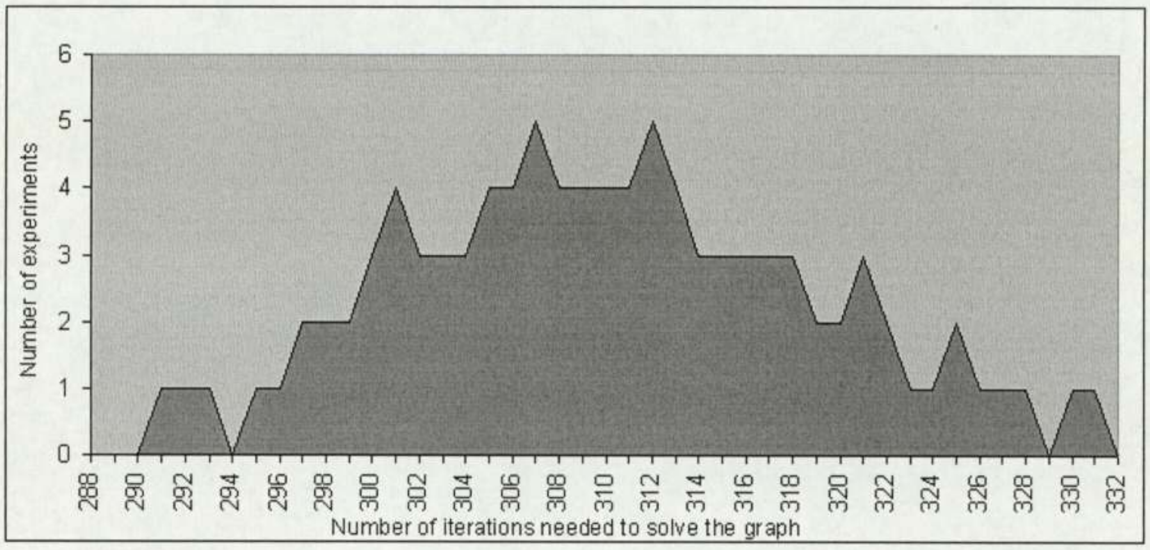

Figure 4.2: Distribution of the results of the 100 experiments to solve a Poissonian graph of mean 3, capacity mean 0.03 and 5,200 nodes with the message passing algorithm.

## 4.3 Properties of the quadratic programming based algorithm

We will focus on the results concerning the quadratic programming based algorithm.

### 4.3.1 Computational complexity against system size

We will see how the quadratic programming based algorithm performs when the system size (i.e. the number of nodes in the graph) varies.

In figure 4.3, I have run the quadratic programming based algorithm on two sets of graphs (Poissonian and linear graphs) where all the parameters are the same except the number of nodes which varies.
When a graph has 1,000 nodes, one iteration means 1,000 updates, and when a graph has 10,000 nodes, one iteration means 10,000 nodes. As the number of iterations doesn't really change when the system size grows, the correlation between the Computational cost needed to solve a graph and the graph itself is based on the number of updates needed to make one iteration.

So we can say from figure 4.3 that the quadratic programming based algorithm scales linearly with the system size. We can also see that this correlation is the same for Poissonian and linear graphs and for low and high capacity means.

The irregular shape of the results for the graphs with low capacity means (at the top of the figure 4.3) may be explained by a low correlation to the system size. Indeed, it seems that the shape is irregular more because of the low capacity mean (which make the graphs harder to solve) than because of the system size: for high capacity mean,

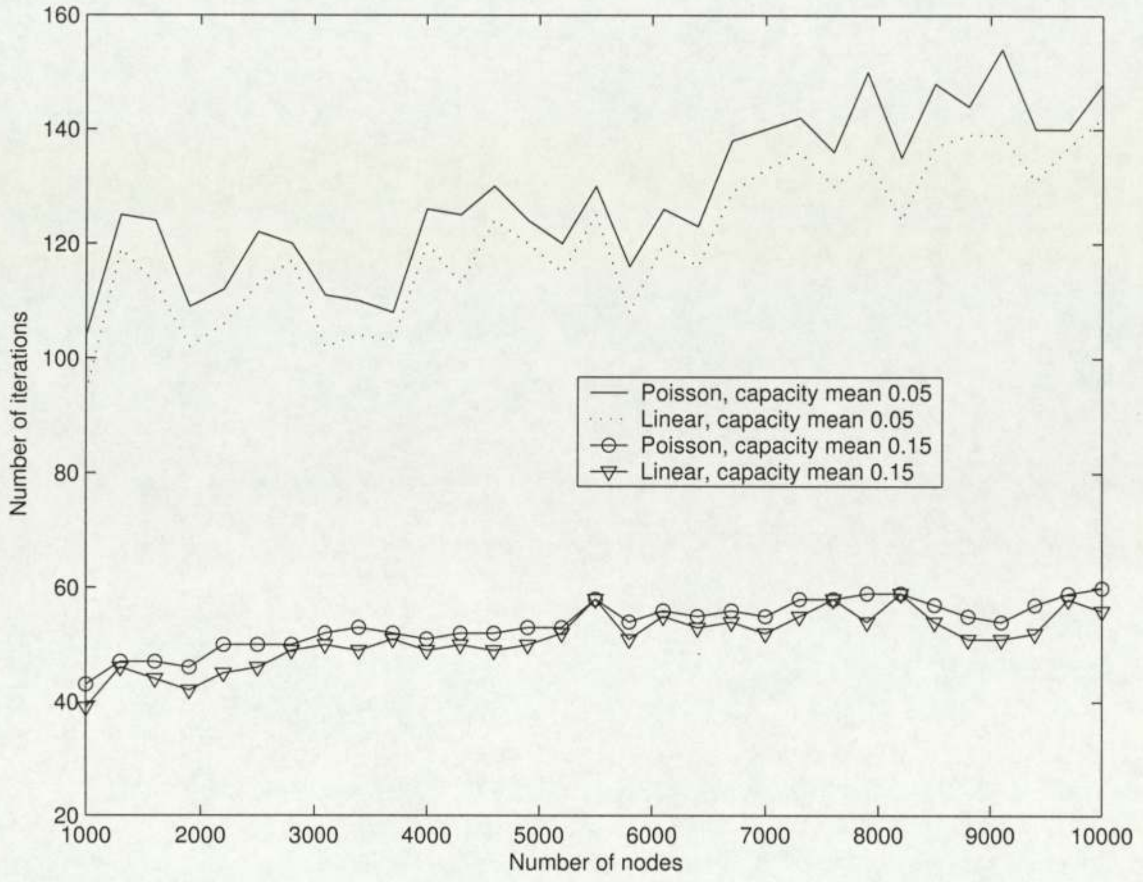

Figure 4.3: Number of iterations needed by the quadratic programming based algorithm to solve Poissonian graphs of mean 4, and linear graphs of mean 4.25 as a function of the number of nodes.

the shape is very clean.

So we can conclude that the quadratic programming based algorithm is linearly correlated with the system size, and we can predict that it will be more strongly correlated to the capacity mean.

### 4.3.2 Computational complexity against connectivity

We will see how the quadratic programming based algorithm performs when the connectivity between the nodes varies.

In figure 4.4, I have run the quadratic programming based algorithm on graphs where only the connectivity means vary, for Poissonian and linear graphs.

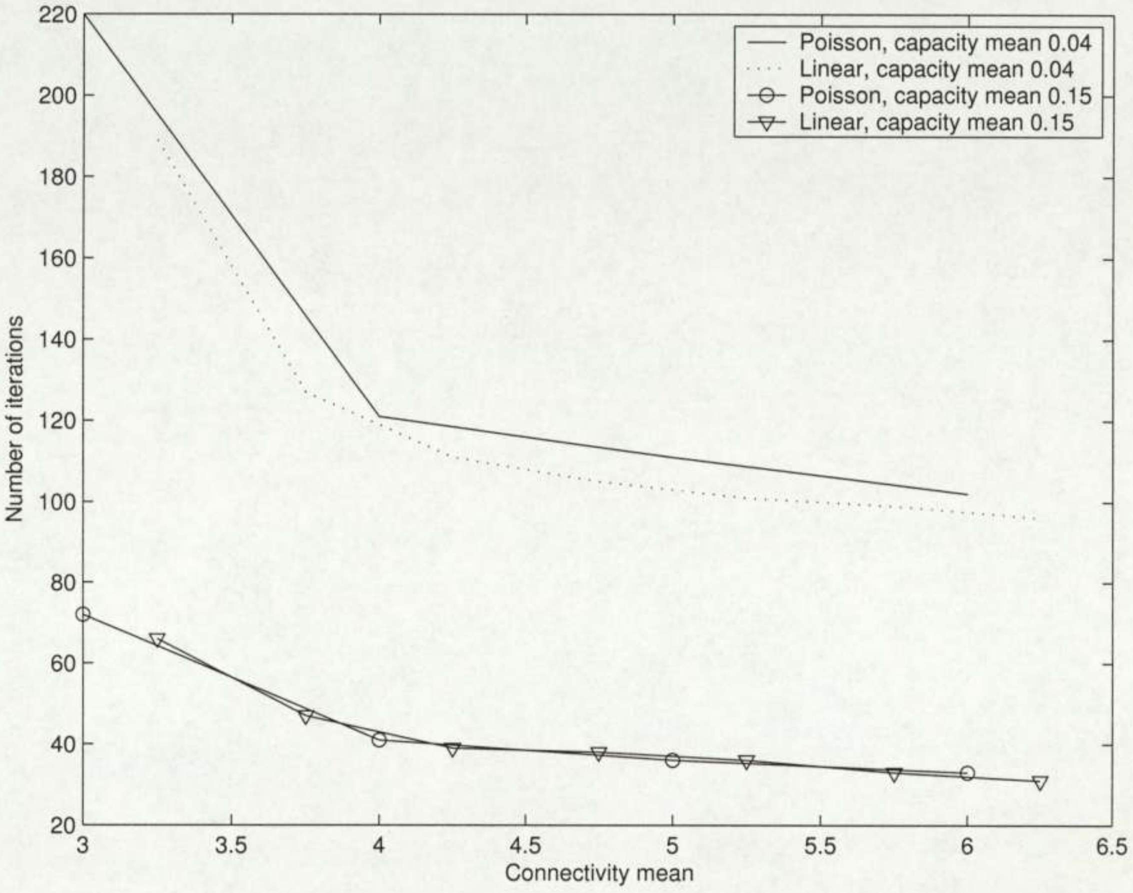

Figure 4.4: Number of iterations needed by the quadratic programming based algorithm to solve graphs of 4,900 nodes as a function of the connectivity mean.

As we can see, the results are very sparse for Poissonian graphs, it is because the connectivity means must be integers. It is where it is useful to have the the linear graphs because we can have non-integer connectivity means (for example, 75% of nodes with connectivity 3 and 25% of nodes with connectivity 4 will give of a connectivity mean of 3.25).

The connectivity means do not go below 3 because a connectivity of 2 for a node is not interesting, the graph would not be a graph anymore, it would be a sort of snake.

Seeing the results in figure 4.4 we can say that the Computational cost increase exponentially when the connectivity mean decreases. It is because when a saturated node has few links to other nodes, the probability of finding an unsaturated node directly connected to it is low (and the graph is hard to solve). When the connectivity increases, this probability increases too, and the graph becomes easier to solve.

The fact that the slope is stronger for low capacity means (at the top of figure 4.4) corresponds to this explanation, when the capacity mean is low, it is even harder to find an unsaturated node (the portion of unsaturated nodes goes down as the capacity mean decreases) so the graph is even harder to solve.

### 4.3.3 Computational complexity against capacity

We will see how the quadratic programming based algorithm performs when the capacity mean of the nodes varies.

In figures 4.5 and 4.6, I have run the quadratic programming based algorithm on graphs where only the capacity means vary, for Poissonian and linear graphs.

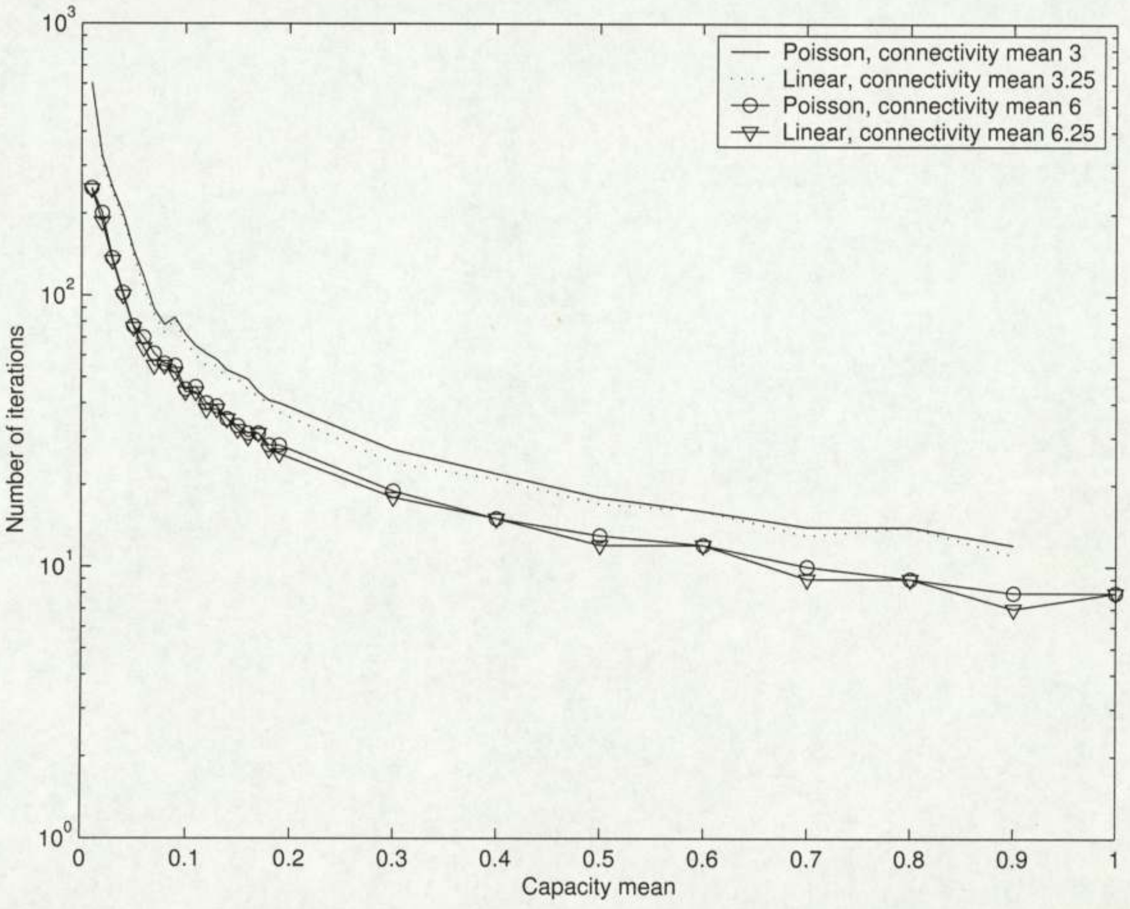

Figure 4.5: Number of iterations needed by the quadratic programming based algorithm to solve graphs of 5,200 nodes as a function of the capacity mean.

These two figures are linear/log plots, the x-axis is in linear format while the y-axis is in log format. This let us see that the number of iterations needed to solve a graph scales exponentially when the capacity mean decreases. With a zoom on the capacity mean between 0.01 and 0.2 (figure 4.6), we can see that it is also the case for low capacities.

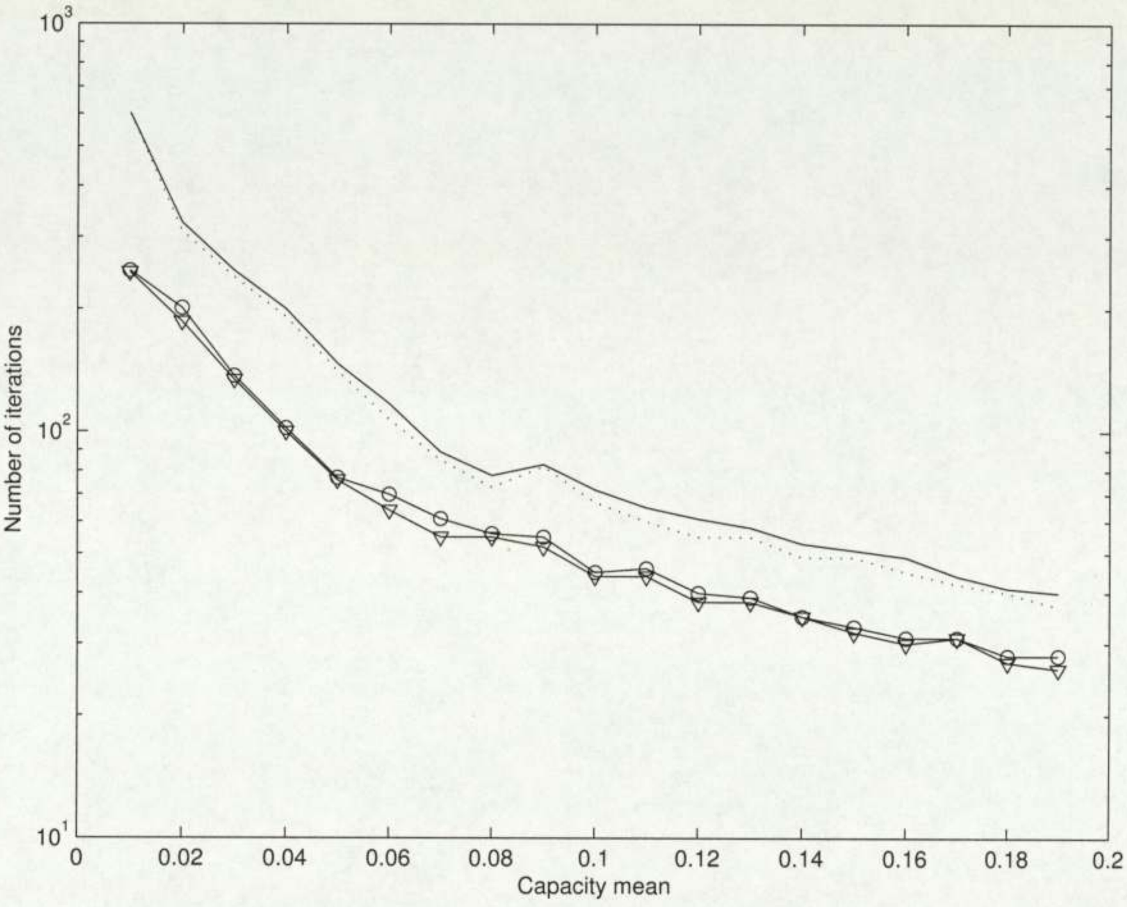

Figure 4.6: Number of iterations needed by the quadratic programming based algorithm to solve graphs of 5,200 nodes as a function of the capacity mean, zooming on capacity mean between 0.01 and 0.2.

It is because when the capacity mean decreases, the portion of saturated nodes in the graph increases. So there are more nodes to unsaturate; it is even worst, because as the capacity mean decreases, the probability of having an unsaturated node connected to a saturated node decreases. So when the capacity mean decreases, the algorithm has to do a lot more iterations to solve the graph.

We can also see from figure 4.5 that there are two different regimes. When the capacity mean is higher than 0.2, the correlation decreases and the graphs become easier to solve. Below a capacity mean of 0.2, the number of iterations needed to solve the graphs increases exponentially when the capacity mean decreases.

The capacity mean is the stronger parameter, its variation will influence a lot the number of iterations needed to solve the graphs.

## 4.4 Properties of the message passing algorithm

We will focus on the results concerning the message passing algorithm.

We see a similar pattern of behavior than for the quadratic programming based algorithm, so the explanations in this part will be very near to the preceding explanations.

### 4.4.1 Computational complexity against system size

We will see how the message passing algorithm performs when the system size varies.

In figure 4.7, I have run the message passing algorithm on two sets of graphs (Poissonian and linear graphs) where all the parameters are the same except the number of nodes which varies.

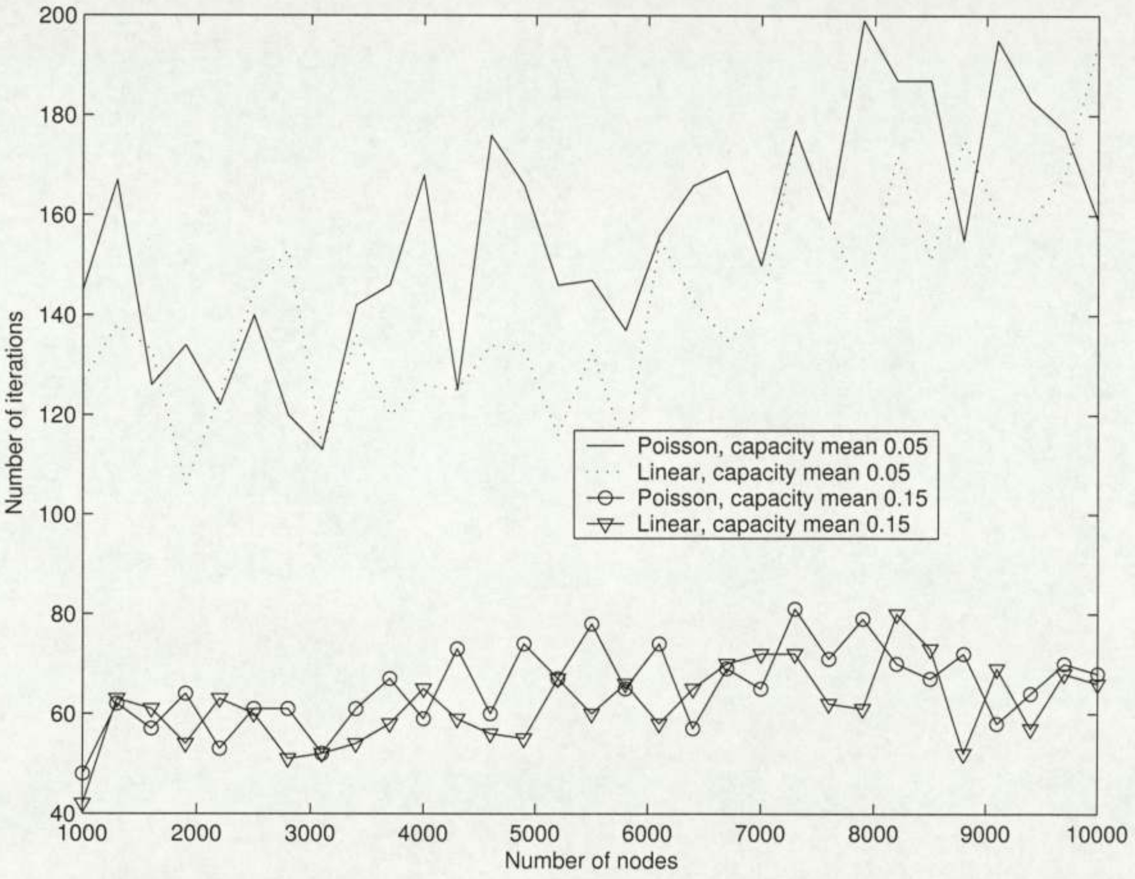

Figure 4.7: Number of iterations needed by the message passing algorithm to solve Poissonian graphs of mean 4, and linear graphs of mean 4.25 as a function of the number of nodes.

We can see that the results are very similar to the results for the quadratic programming based algorithm. The message passing algorithm scales also linearly with the system size; it is also the same for Poissonian and linear graphs and for low and high capacity means.

As for the quadratic programming based algorithm, the irregular shape of the results for the graphs with low capacity means (at the top of figure 4.7) can be explained by a low correlation to the system size. Indeed, it seems that the shape is irregular more because of the low capacity mean (which make the graph harder to solve) than because of the system size: the shape is very clean for high capacity mean.

### 4.4.2 Computational complexity against connectivity

We will see how the message passing algorithm performs when the connectivity between the nodes varies.

In figure 4.8, I have run the message passing algorithm on graphs where only the connectivity means vary, for Poissonian and linear graphs.

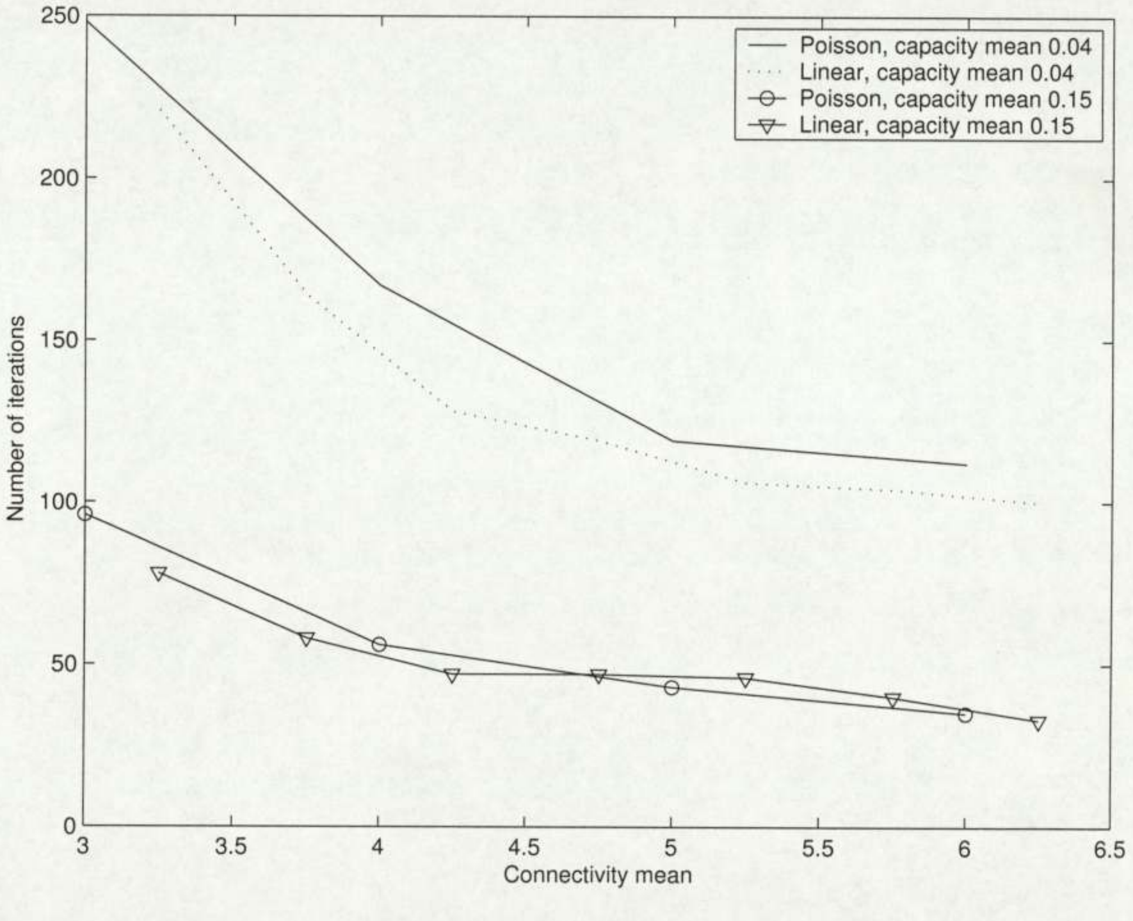

Figure 4.8: Number of iterations needed by the message passing algorithm to solve graphs of 4,900 nodes as a function of the connectivity mean.

Again, it is the same as for the quadratic programming based algorithm, the figure 4.8 tells us that the message passing algorithm scales exponentially when the connectivity mean decreases. The explanation is the same as well: it is because when a saturated node a few links to other nodes, the probability of being connected to an unsaturated node is low (and the graph is hard to solve). When the connectivity increases, this probability increases too and the graph becomes easier to solve.

### 4.4.3 Computational complexity against capacity

We will see how the message passing algorithm performs when the capacity mean of the nodes varies.

In figures 4.9 and 4.10, I have run the message passing algorithm on graphs where only the capacity means vary, for Poissonian and linear graphs.

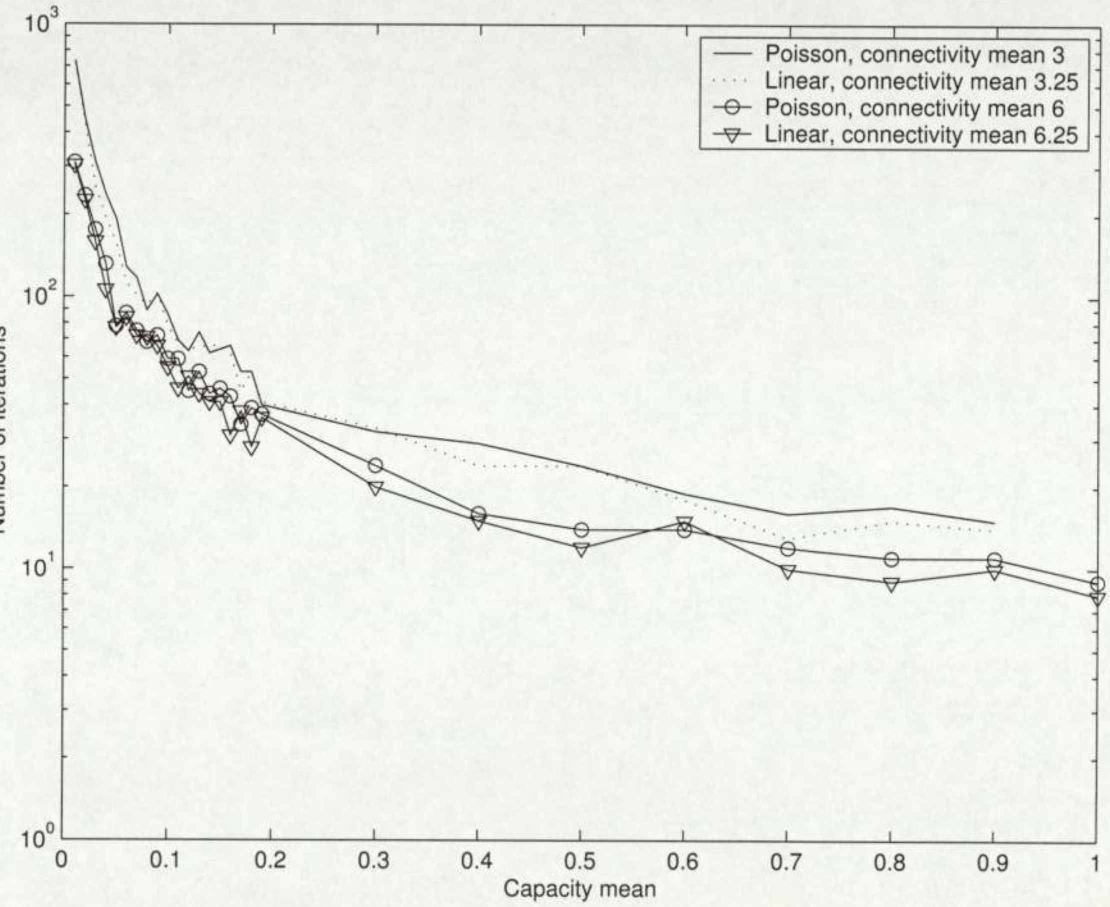

Figure 4.9: Number of iterations needed by the message passing algorithm to solve graphs of 5,200 nodes as a function of the capacity mean.

These two figures are linear/log plots, the x-axis is in linear format while the y-axis is in log format. This let us see that the number of iterations needed to solve a graph scales exponentially when the capacity mean decreases. With a zoom on the capacity mean between 0.01 and 0.2 (figure 4.10), we can see that it is also the case for low capacities.

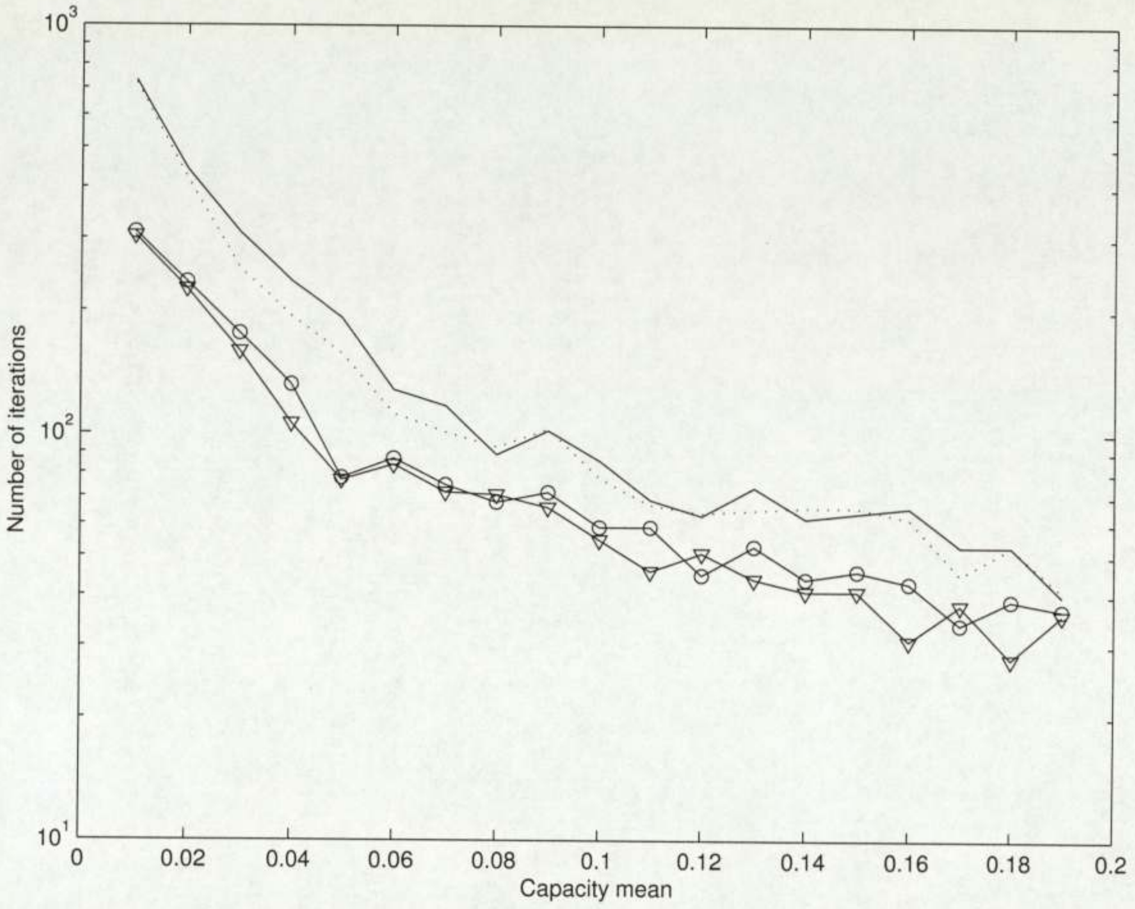

Figure 4.10: Number of iterations needed by the message passing algorithm to solve graphs of 5,200 nodes as a function of the capacity mean, zooming on capacity mean between 0.01 and 0.2.

And again, the behavior is the same than for the quadratic programming based algorithm, when the capacity mean decreases, the portion of saturated nodes in the graph increases. So there are more nodes to unsaturate; it is even worst, because as the capacity mean decreases, the probability of having an unsaturated node connected to a saturated node decreases. So when the capacity mean decreases, the algorithm has to do a lot more iterations to solve the graph.

We can also see from figure 4.9 that there are two different regimes. When the capacity mean is higher than 0.2, the correlation decreases and the graphs become easier to solve. Below a capacity mean of 0.2, the number of iterations needed to solve the graphs increases exponentially when the capacity mean decreases.

The capacity mean is the stronger parameter, its variation will influence a lot the number of iterations needed to solve the graphs.

## 4.5 Comparison of the two algorithms

We know how each algorithm scales against each parameter; we will now compare the scaling of the two algorithms together.

### 4.5.1 Computational complexity against system size

We will compare the scaling of each algorithm against the system size.

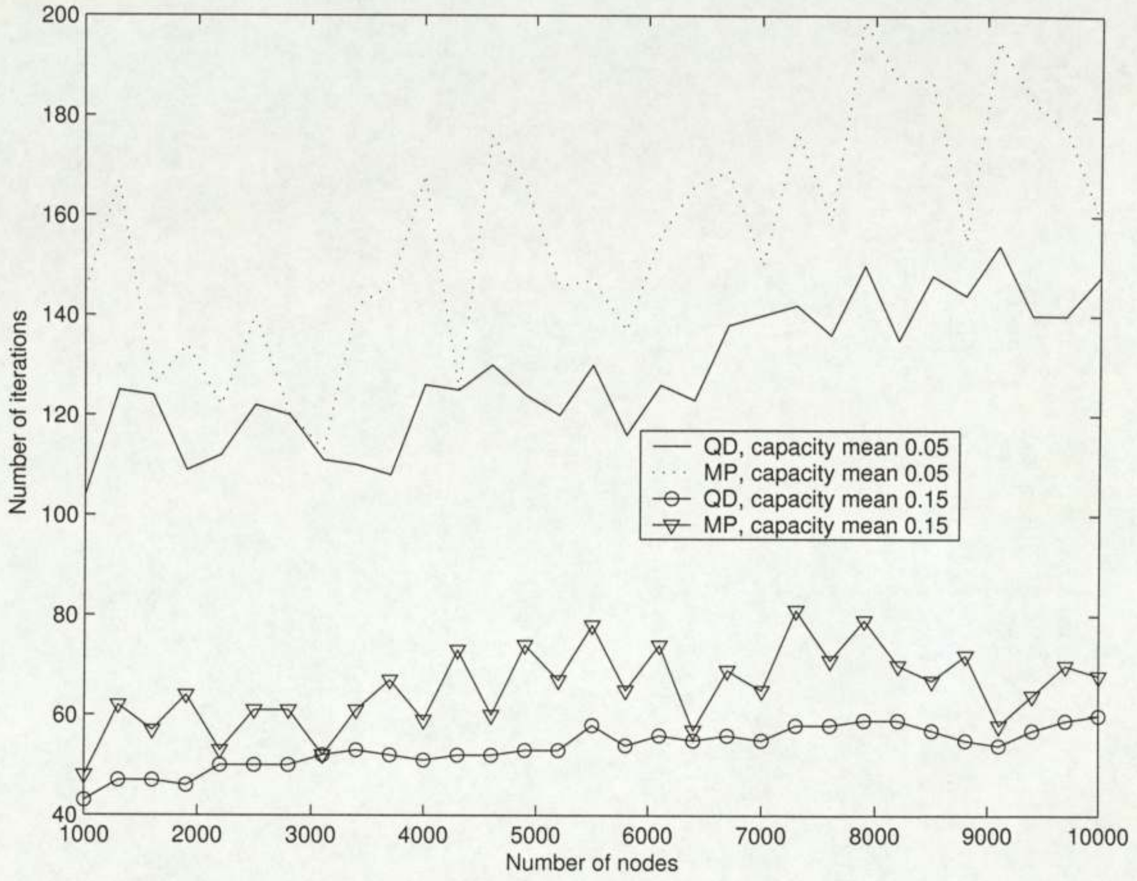

Figure 4.11: Number of iterations needed by the two algorithms (QD and MP) to solve Poissonian graphs of mean 4 as a function of the number of nodes.

As we can see in figure 4.11, the two algorithms scale the same way against the system size. The results are very similar except that the message passing algorithm is less regular than the quadratic programming based algorithm. This difference is due to the different origins of the two algorithms, the quadratic programming based algorithm is directly derived from the cost function while the message passing algorithm is more generic and so a little less optimized.

## 4.5.2 Computational complexity against connectivity

We will compare the scaling of each algorithm against the connectivity mean.

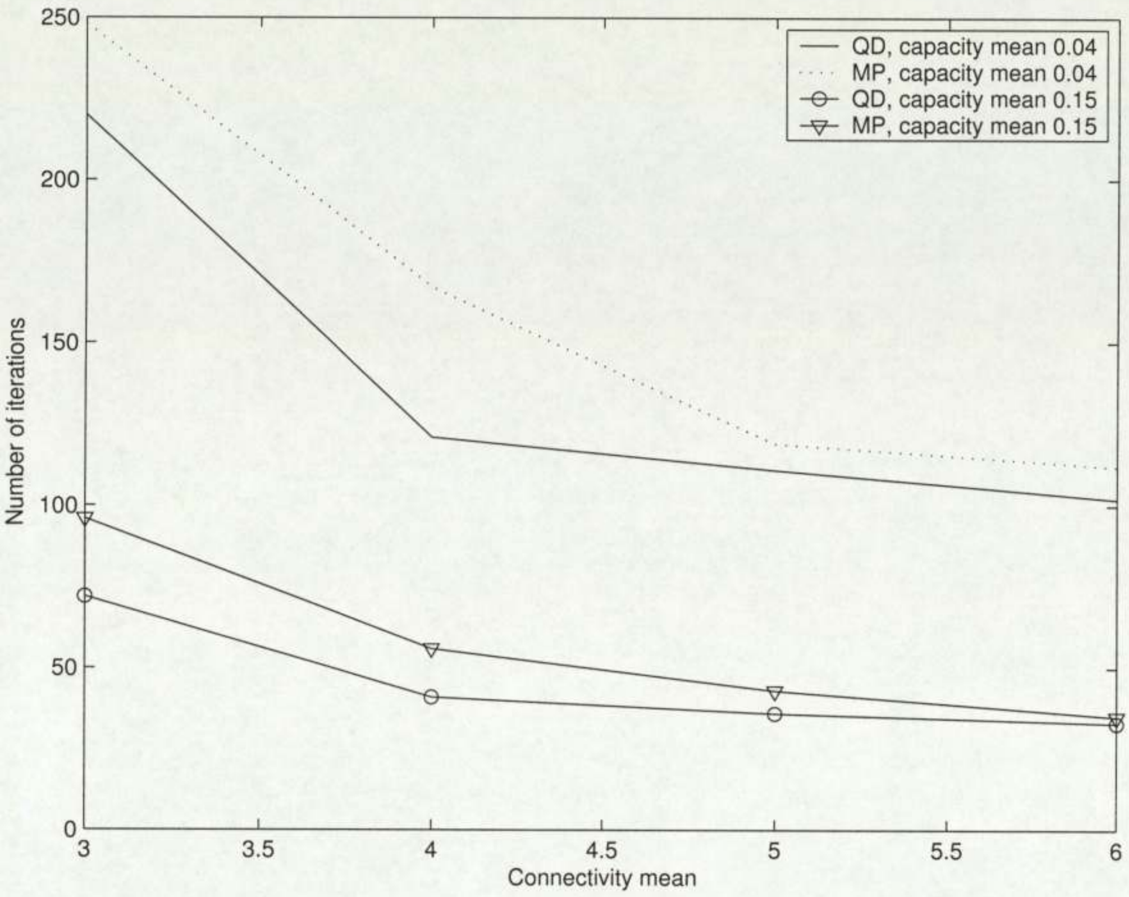

Figure 4.12: Number of iterations needed by the two algorithms (QD and MP) to solve Poissonian graphs of 4,900 nodes as a function of the connectivity mean.

Again, the two algorithms scale exactly the same way against the connectivity mean, and the message passing algorithm is a little less good than the quadratic programming based algorithm.

## 4.5.3 Computational complexity against capacity

We will compare the scaling of each algorithm against the capacity mean.

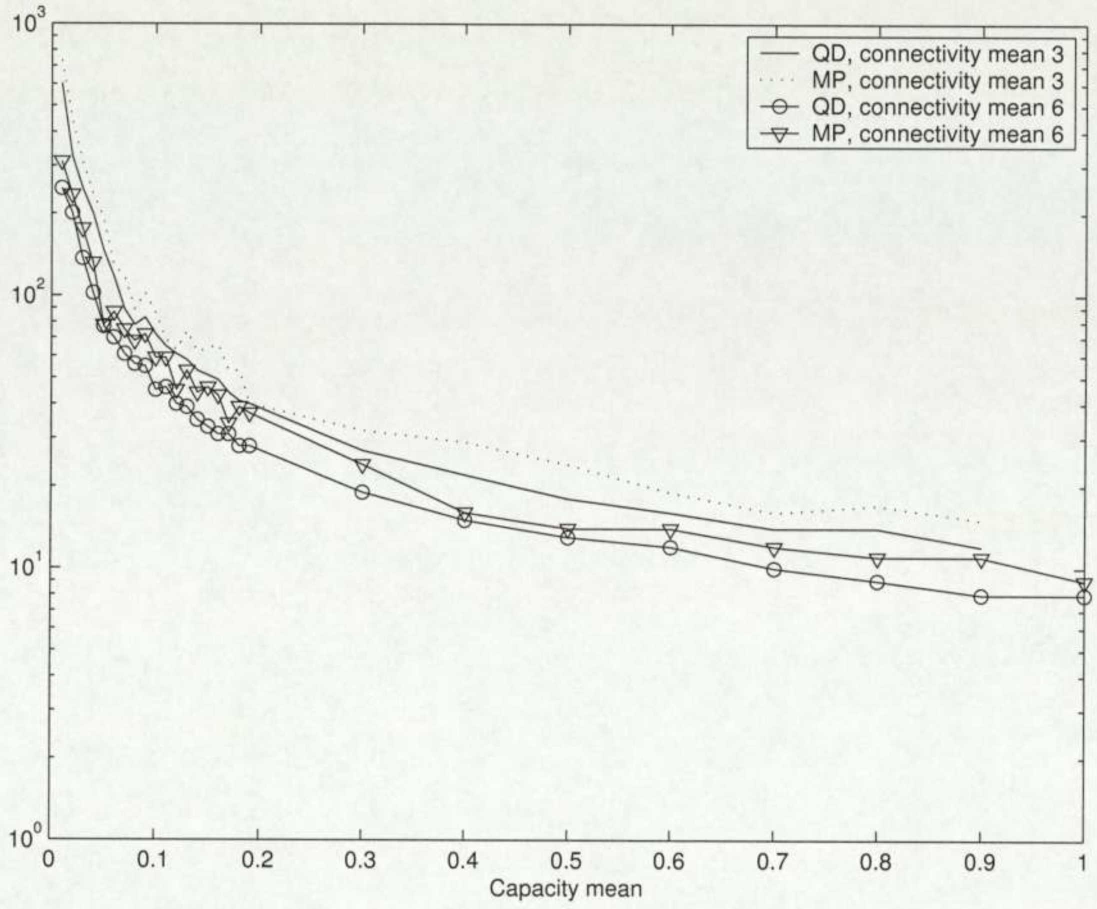

Figure 4.13: Number of iterations needed by the two algorithms (QD and MP) to solve Poissonian graphs of 5,200 nodes as a function of the capacity mean.

One more time, the two algorithms scale the same way against the capacity mean, while the message passing algorithm needs a little more iterations than the quadratic programming based algorithm to solve the graphs.

### 4.5.4 Evolution of the cost

We will see how the cost evolves for each algorithm.

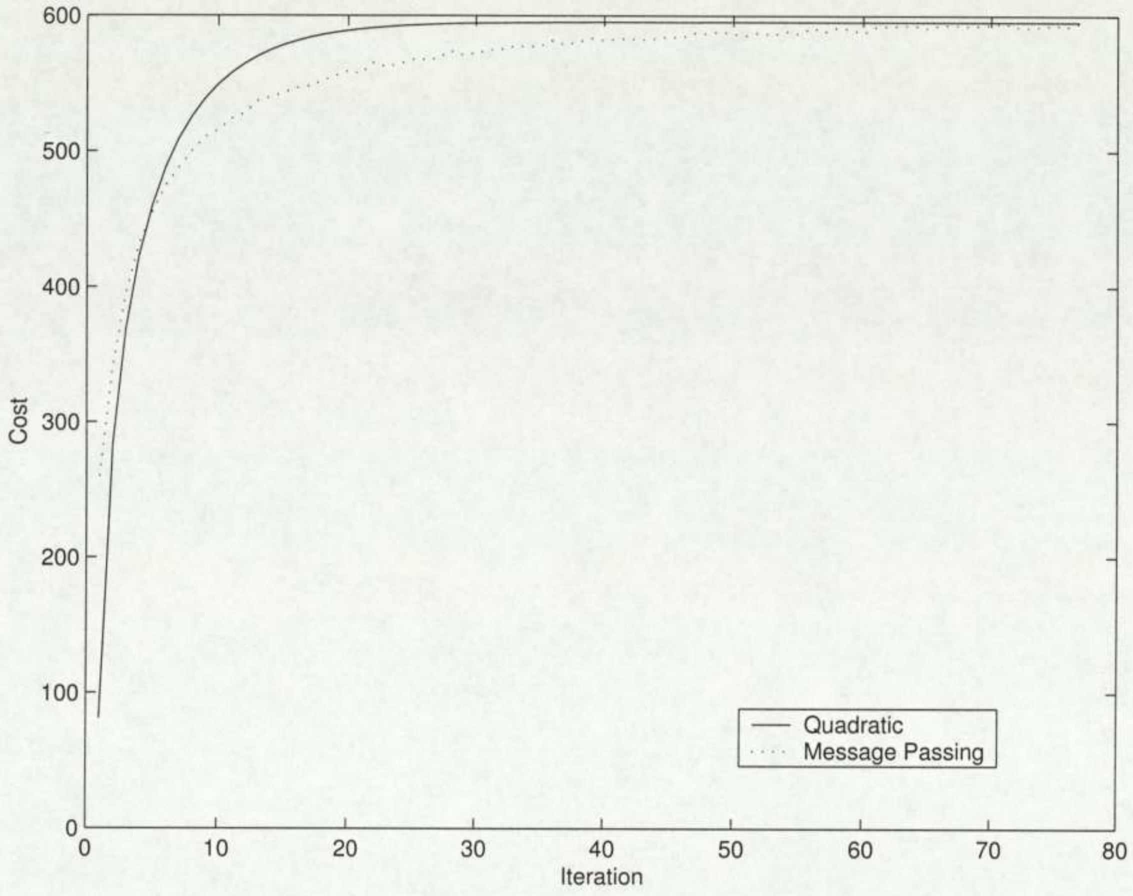

Figure 4.14: Evolution of the cost during the resolution of a Poissonian graph with mean 5, 9,700 nodes and capacity mean 0.1.

We can see in figure 4.14 how the cost evolves during the resolution of a graph for the two algorithms. It is very interesting because it shows us how each algorithm make its optimization. We can see that the message passing algorithm moves more jobs in the beginning, the cost function increases quicker than for the quadratic programming based algorithm. But for the end, it is clear that the quadratic programming based algorithm goes to the steady state quicker than the message passing algorithm. The message passing algorithm seems to have more difficulties to reach the steady state.

It explains why the message passing needs more iterations than the quadratic programming based algorithm to solve the same graphs (figures 4.11, 4.12 and 4.13).

# Chapter 5

# Future Work

In the future, it will be interesting to explore further how the message passing algorithm performs on more complex cases.

## 5.1 Unsatisfiable cases

For all my experiments, I have generated graphs where the overall capacity where positive. It means that the whole computational capacity of the nodes was high enough to solve the task applied on the graph. On this kind of graph, we are sure that a solution to the problem exists.

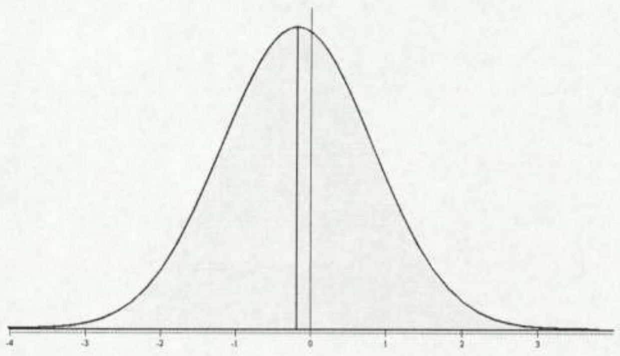

Figure 5.1: Distribution of the node capacities used to obtain an unsatisfiable case.

Figure 5.1 is an example of the capacity distribution which can be used to generate a graph where the overall capacity is negative. By using this kind of capacity distribution, we generate graphs where the overall capacity is negative, it means that there is no solution to the problem.

A solution suggested by David Saad to get good results on unsatisfiable cases (figure 5.2) is to inverse our optimization process. Currently, we have some saturated nodes that we want to unsaturate thanks to the unsaturated nodes linked to it. For unsatisfiable cases, we may consider that we have some unsaturated nodes that we want to saturate thanks to the saturated nodes linked to it.

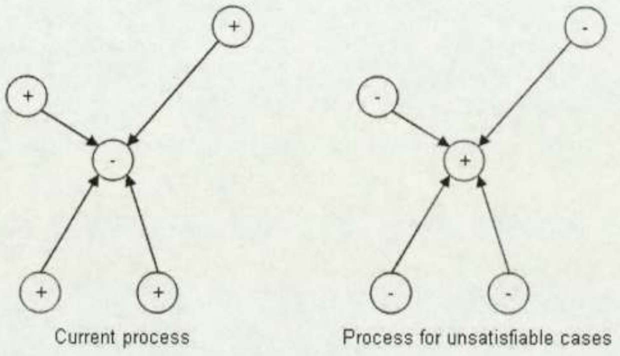

Figure 5.2: Solution suggested by David Saad to get good results on unsatisfiable cases.

With this process, we are sure we will be able to have good results on unsatisfiable cases exactly the same way that the cases studied earlier.

## 5.2 More complex cost function

### 5.2.1 Coefficients on the edges

For all the experiments, the cost function I used was very simple and directly based on the flow.

It will be interesting to see how the message passing algorithm performs with more complex cost functions, as other restrictions may exist in the network.

For instance, we may imagine a cost function which takes into account a cost on edges, so it will be possible to attribute costs to edges and see how the message passing algorithm performs under these conditions.

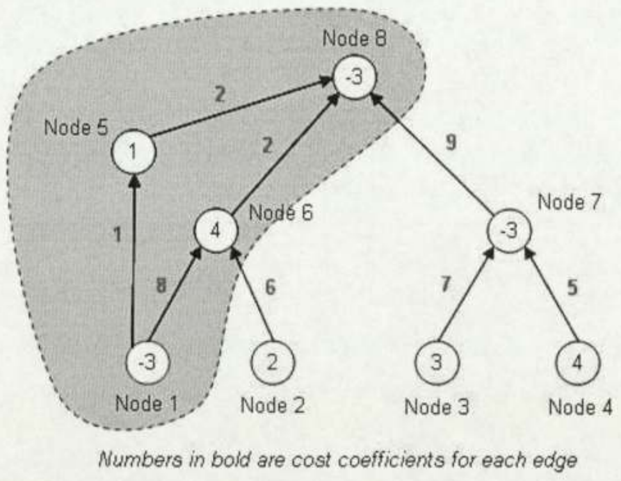

Figure 5.3: Example of graph with cost coefficients on edges.

For example, figure 5.3, contains cost coefficients on its edges; it is less costly to use nodes 5 and 8 to pass jobs from node 1 to node 6 instead of using the direct route.

### 5.2.2 Coefficients on the nodes

Another interesting cost function is one cost function taking into account a cost coefficient on each node.

Using such cost function, it will be possible to specify more and less important nodes (with different coefficients) in the graph in term of reliability, applicability, etc. In real life applications, defining vital nodes is very important for the reliability of the system, because due to this information, the algorithm will be able to consider these vital nodes in the optimization process.

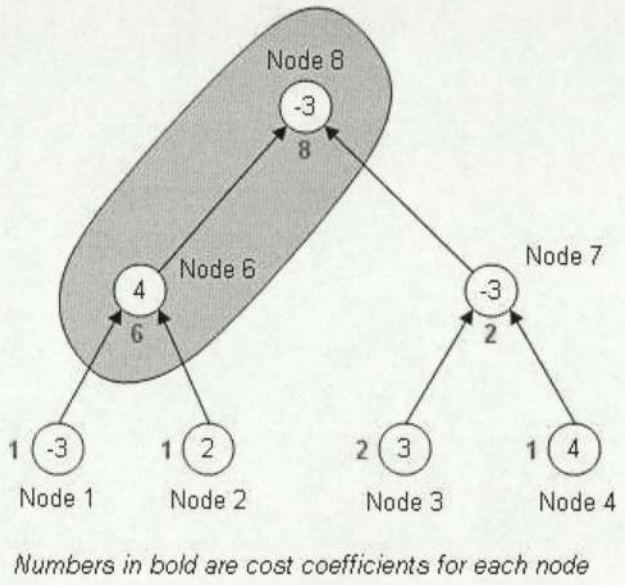

Figure 5.4: Example of graph with cost coefficients on nodes.

In figure 5.4, we can see the two nodes of high importance (nodes 6 and 8) defining the backbone of the system.

Thanks to a knowledge of more important nodes, in cases where the whole problem can't be solve, the algorithm will be able to unsaturate in priority the vital nodes. This kind of optimization is very important for Internet application for example, where in some cases, the network is saturated, but it is very important to keep the backbone unsaturated not to have the whole network stuck.

## 5.3 More complex graphs

For the experiments, I always have used simple random graphs.

It may be interesting to see how the message passing algorithm performs on complex structured graphs. For example, we can take a graph constructed with two standard graphs linked together with a bridge of few vertices (i.e. 1 or 2).

This kind of graph may be difficult to solve, and it may be more simple to try to solve each of the standard graph separately and then take the bridge into account.

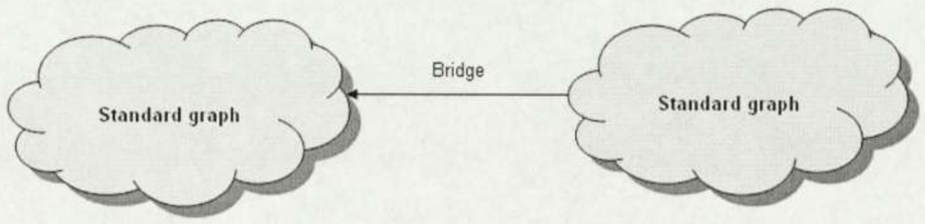

Figure 5.5: Example of a complex graph composed of two standard graphs and a bridge between them.

For instance, figure 5.5 may be very difficult to solve, even if the two standard graphs are simple. This is due to the fact that a lot of jobs may have to pass through the bridge, and the message passing algorithm may require a lot of iterations to pass all the messages and satisfy all the demands by passing messages through the bridge.

# Chapter 6

# Conclusion

## 6.1 Summary of the work done

We have tried to find an efficient algorithm to solve the "Resource allocation on sparse graphs" problem. We have first derived an optimized algorithm specific to a defined cost function - the quadratic programming based algorithm -, and a generic algorithm based on the same cost function - the message passing algorithm -.

We then have generated a graph set of two different types of graphs with different properties to have a large panel of graphs to run our algorithms on.

Finally, we have applied these two algorithms on our graph set to see how each algorithm performs.

And, seeing the results, we have concluded that the generic algorithm is very efficient, it scales the same way that the optimized algorithm for each complexification of our graph set.

These results tell us that this new generic algorithm may be very efficient for complex cases based on more complex cost functions where a specific optimization can't be found. It can be applied to more realistic cases of nonlinear cost functions and additional constraints on the capacities of nodes and links, which constitutes a rich area for further investigations with many potential applications.

## 6.2 Afterword

This project has been interesting for many reasons: from the definition of the problem to the application of the two algorithms. It has not only be a way to understand and apply the methods learnt in the first term of the MSC, but it has also be a way to discover the real work environment: working on the cluster with MPI library, creating a test protocol, managing the results, aso.

I also really want to thank David again for his help and is kindness with me over these nine months.

# Appendix A

# Listings

## A.1 Formal algorithms

Here are very formal algorithms just to show how simple each algorithm is.

### A.1.1 Graph generator - Poissonian graphs

The algorithm used to generate the Poissonian graphs.

We have defined a connectivity mean $ConnecMean$ and a number of nodes $NbNodes$.

```
$TotalLinks = ConnecMean * NbNodes$
While ($TotalLinks$ not reached)
    • Select two nodes $N1$ and $N2$ randomly
    • Create a link between $N1$ and $N2$
EndWhile
```

### A.1.2 Graph generator - Linear graphs

The algorithm used to generate the linear graphs.

We have defined a connectivity mean $ConnecMean$ and its ratio $ConnecRatio$. For example, if we define $ConnecMean = 3$ and $ConnecRatio = 75\%$, it will generate a graph with 75% of the nodes with connectivity 3 and 25% with connectivity 4.

The algorithm will define, for each node $Nx$, the needed number of links : $AimNbLinks(Nx)$ and the current number of links $CurNbLinks(Nx)$ with respect to the parameters we have defined.

```
While (All the links do not respect $CurNbLinks(Nx) = AimNbLinks(Nx)$)
    • Select two nodes $N1$ and $N2$ randomly where
      $CurNbLinks(N1) < AimNbLinks(N1)$ and
      $CurNbLinks(N2) < AimNbLinks(N2)$
    • Create a link between $N1$ and $N2$
EndWhile
```

## A.1.3 Quadratic programming based algorithm

The quadratic programming based algorithm used to solve the graph set.

```
While (Steady state not reached)
    While {All the node have not been updated}
        • Select randomly a node $N$ among those not yet updated
        • Update $N$ with the quadratic programming based algorithm (see section
          2.2.4)
    EndWhile
EndWhile
```

## A.1.4 Message Passing algorithm

The message passing algorithm used to solve the graph set.

```
While (Steady state not reached)
    While {All the node have not been updated}
        • Select randomly a node $N$ among those not yet updated
        • Update $N$ with the message passing algorithm (see section 2.3.2)
    EndWhile
EndWhile
```

# Bibliography

[1] Peterson L. and Davie B.S. *Computers Networks : A systems approach.* Academic Press, San Diego CA, 2000.

[2] Servi L. Ho Y.C. and Suri R. *Large Scale Systems.* 1, 1980.

[3] Michael Wong and David Saad. Load balancing on a tree. 2004.

[4] Michael Wong and David Saad. The chemical potential in a tree. 2004.

[5] David Saad Michael Wong and Zhuo Gao. Message passing for task redistribution on sparse graphs. 2005.