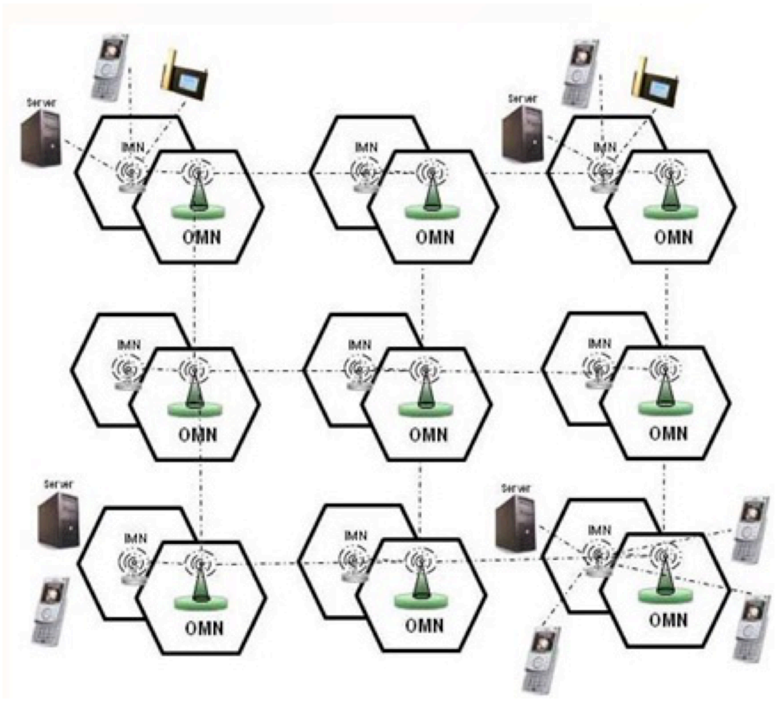
Multi-agent project Report

# Multiagent Frequency Allocation



ZunzunWANG

YishuoLYU

# Summary

## 1 - Introduction

When radio communication links are assigned the same or closely related frequencies, there is a potential for interference. Consider a radio communication network, defined by a set of radio links. The radio link frequency assignment problem is to assign, from limited spectral resources, a frequency to each of these links in such a way that all the links may operate together without noticeable interference. Moreover, the assignment has to comply with certain regulations and physical constraints of the transmitters. Among all such assignments, one will naturally prefer those which make good use of the available spectrum, trying to save the spectral resources for a later extension of the network. Furthermore, When several bands are available, and for various reasons (for example radio wave propagation, ease of deployment etc.), the lower bands are usually most favoured and one should try to assign frequencies preferably from the lower bands.

## 2 - Problem Presentation

The Radio Link frequency Assignment Problem consists in assigning frequencies to a set of radio links defined between pairs of sites in order to avoid interferences. Each radio link is represented by a variable whose domain is the set of all frequencies that are available for this link. The essential constraints involve two variables $F_1$ et $F_2$:

$$|F_1 - F_2| > k_{12}$$

The two variables represent two radio links which are "close" one to the other. The constant $k_{12}$ depends on the position of the two links and also on the physical environment. It is obtained using a mathematical model of electromagnetic waves propagation which is still very "rough". Therefore, the constants $k_{12}$ are not necessarily correct (it is possible that the minimum difference in frequency between $F_1$ and $F_2$ that does not yield interferences is actually different from $k_{12}$). In practice, $k_{12}$ is often overestimated in order to effectively guarantee the absence of interference. For each pair of sites, two frequencies must be assigned: one for the communications from *A* to *B*, the other one for the communications from *B* to *A*. In the case of the CELAR instances, a technological constraint appears: the distance in frequency between the *A*->*B* link and the *B*->*A* link must be exactly equal to 238. In all CELAR instances, these pairs of links are represented as pairs of variables numbered *2k* and *2k+1*. The possibility of expressing constraints such as $>|F_1 - F_2| > k_{12}$ suffices to express the graph coloring problem and it is therefore clear that the RLFAP is NP-hard.

## The criteria optimized

In order to facilitate the later addition of new radio links, one tries to build solutions that leave room for these new links. The pure satisfaction problem is therefore not crucial in itself (even if...). Two essential criteria are usually considered for feasible instances:

1. **Minimization of the maximum frequency used**: the frequencies above the last frequency used can be tried for new radio links.

2. **Minimization of the number of frequencies useds**: the different frequencies unused can be tries for new radio links. Here, it is likely that the various frequencies available will be more distant one from the other and therefore it is more likely that a new radio link can be inserted without any modification of the previous setup. In practice, this criteria seems therefore more interesting than the previous one.

## 3 - Data presentation

Eleven CELAR instances are available. We use the CELAR scen01 as our data.

*Ref: http://www7.inra.fr/mia/T/schiex/Export/FullRLFAP.tgz*

In our instance, we have 4 txt files:

Each instance is distributed in four files in a single directory.

`var.txt`: gives the list of the variables:
We use all the antennas that work in domain 5.

| |
|---|
| *141, 142, 251, 252, 293, 292* |

`dom.txt`: gives the domain's definition,
The frequencies we can use are:

| |
|---|
| *6 142 170 240 380 408 478* |

`ctr.txt`: gives the list of all constraints,

| **Constraint table** |
|---|
| **141 251 C > 13** |
| **141 252 F > 40** |
| **142 251 F > 40** |
| **142 252 C > 13** |
| **251 252 D = 238** |
| **293 294 D = 238** |

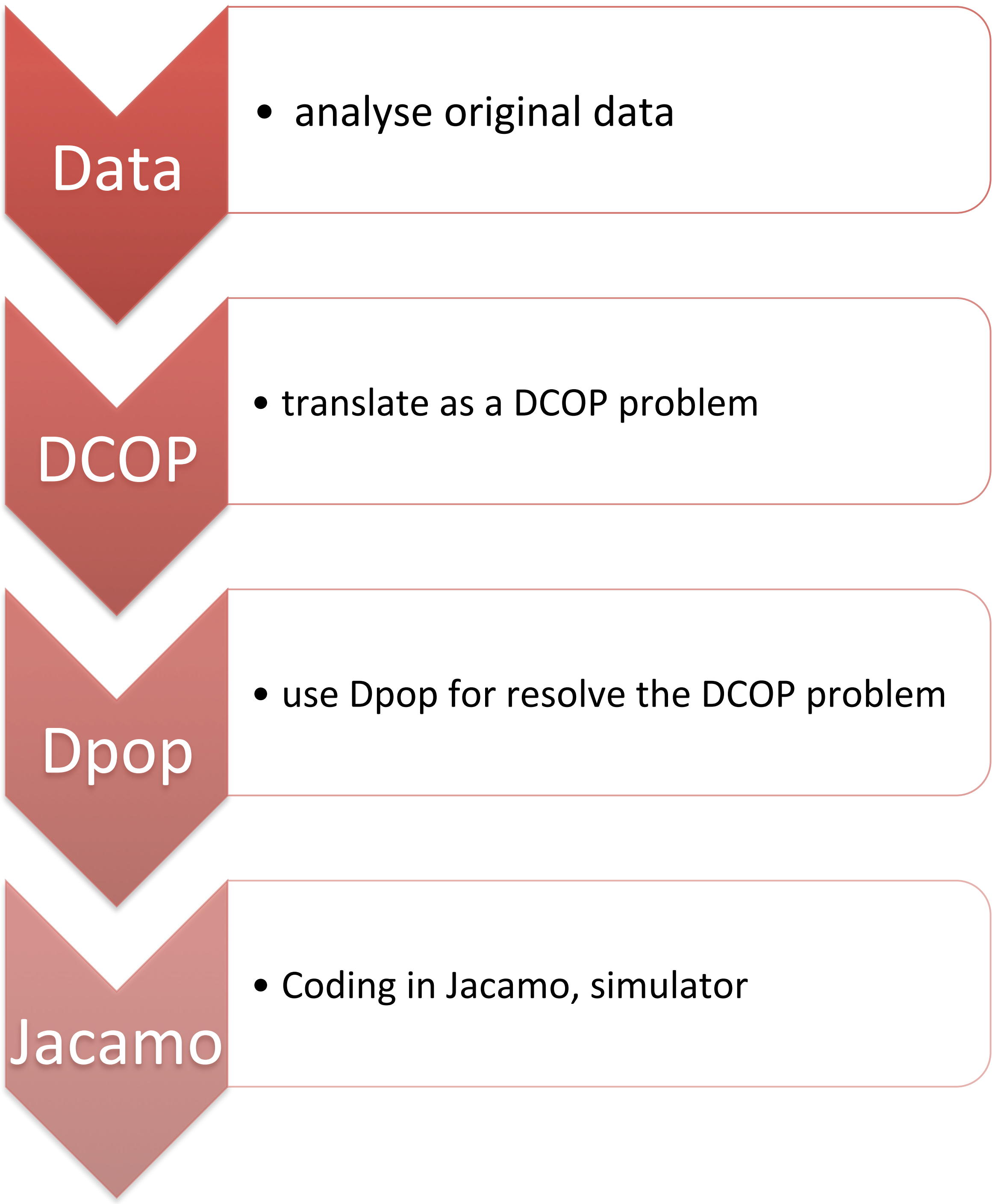
`cst.txt`: defines the criteria to be optimized *a priori* on this instance.
We don't use in our example.

## 4 - Goal

We want to develop a multi-agent simulator to solve frequency assignment problems (FAPP),

We use Jacamo to model antennas as agents that have to cooperate to find the optimal frequency assignment.

We will translate this problem as a distributed constraint optimization problem. And we will use the DPOP algorithm to resolve this question.


Data
• analyse original data
DCOP
• translate as a DCOP problem
Dpop
• use Dpop for resolve the DCOP problem
Jacamo
• Coding in Jacamo, simulator

## 5 - Data analyse

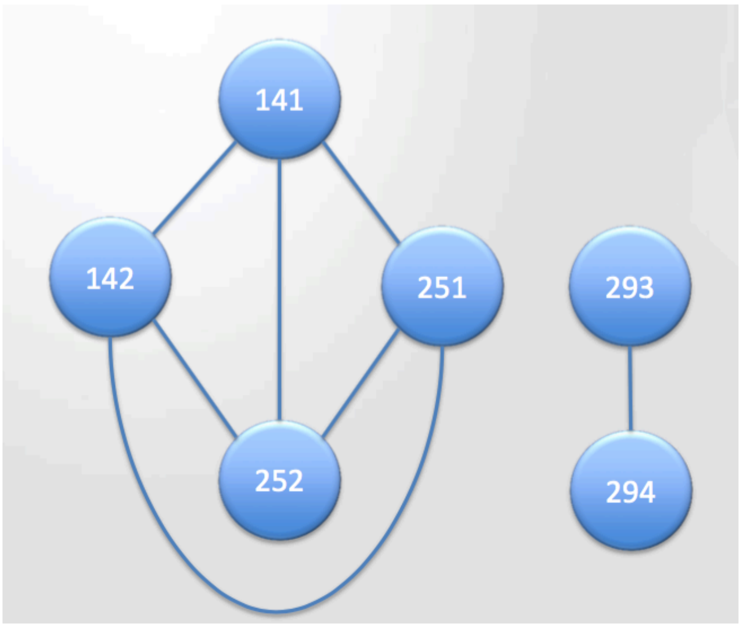
First, we have to read the file .txt and quickly collect the data needed (antennas in the same domain and the restrictions), and then we will input the data that we get in the first step into the other Java project for getting the DFS tree (the relationship of parent, pseudo-parent, children and pseudo-children).

Second, we will input the DFS tree and the restrictions of every two antennas into the Jacamo project. After the end of the project execution, we can get the final result that satisfier the problem FAPP.

As we mentioned, we use 6 agents to simulate this question. And we can get the graph by the Constraint table.

In fact, each constraint definite a lien between 2 agents. So we can get our graph like this:

| Constraint table |
|---|
| 141 251 C > 13 |
| 141 252 F > 40 |
| 142 251 F > 40 |
| 142 252 C > 13 |
| 251 252 D = 238 |
| 293 294 D = 238 |



## 6 - DCOP (solution: DPOP)

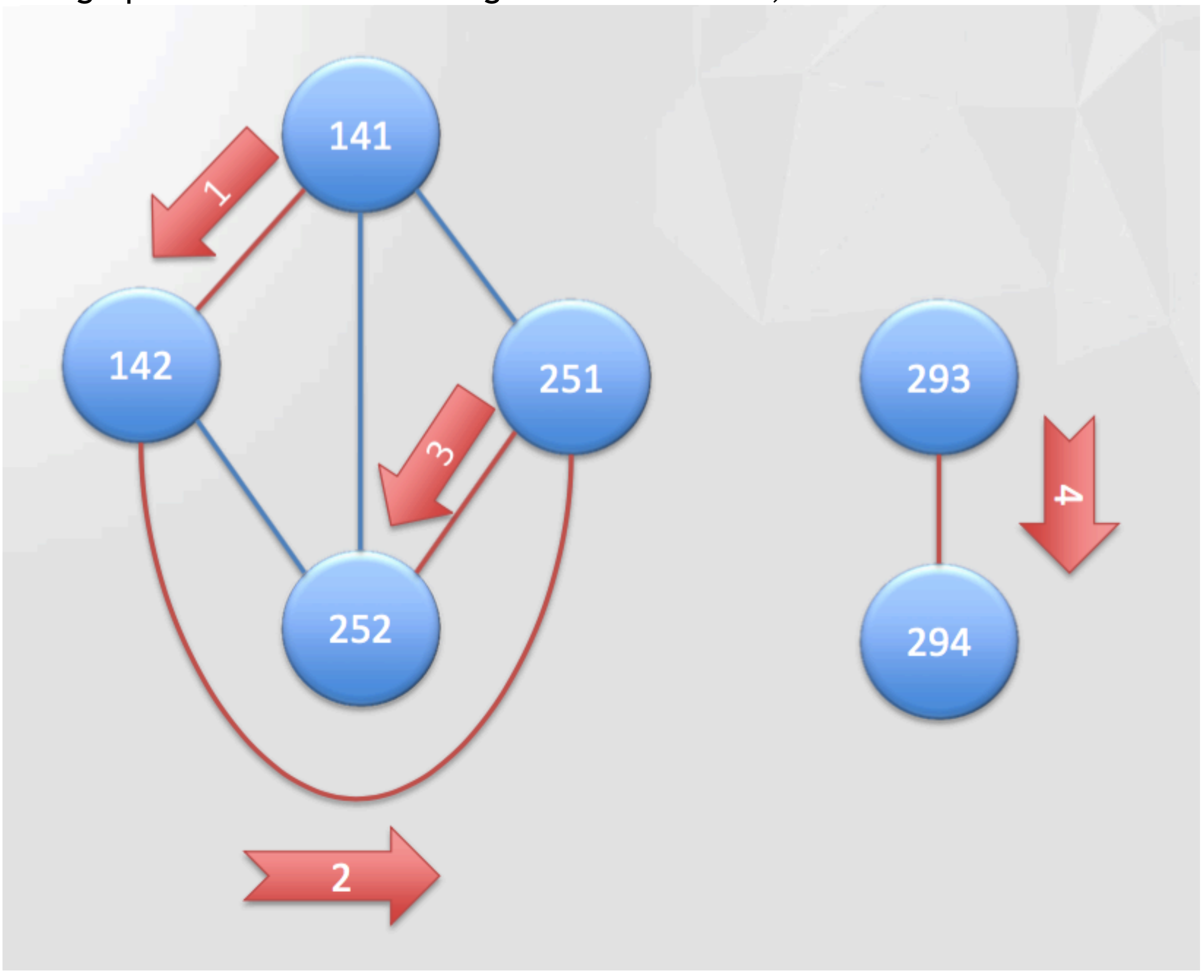
This part will realise also in java.

DPOP (dynamic programming optimization protocol)
There are 3-phases pour Dpop:
(1) Create DFS Tree
(2) Utile message Phase: from leaves to the root.
(3) Value message Phase: from root to leaves.

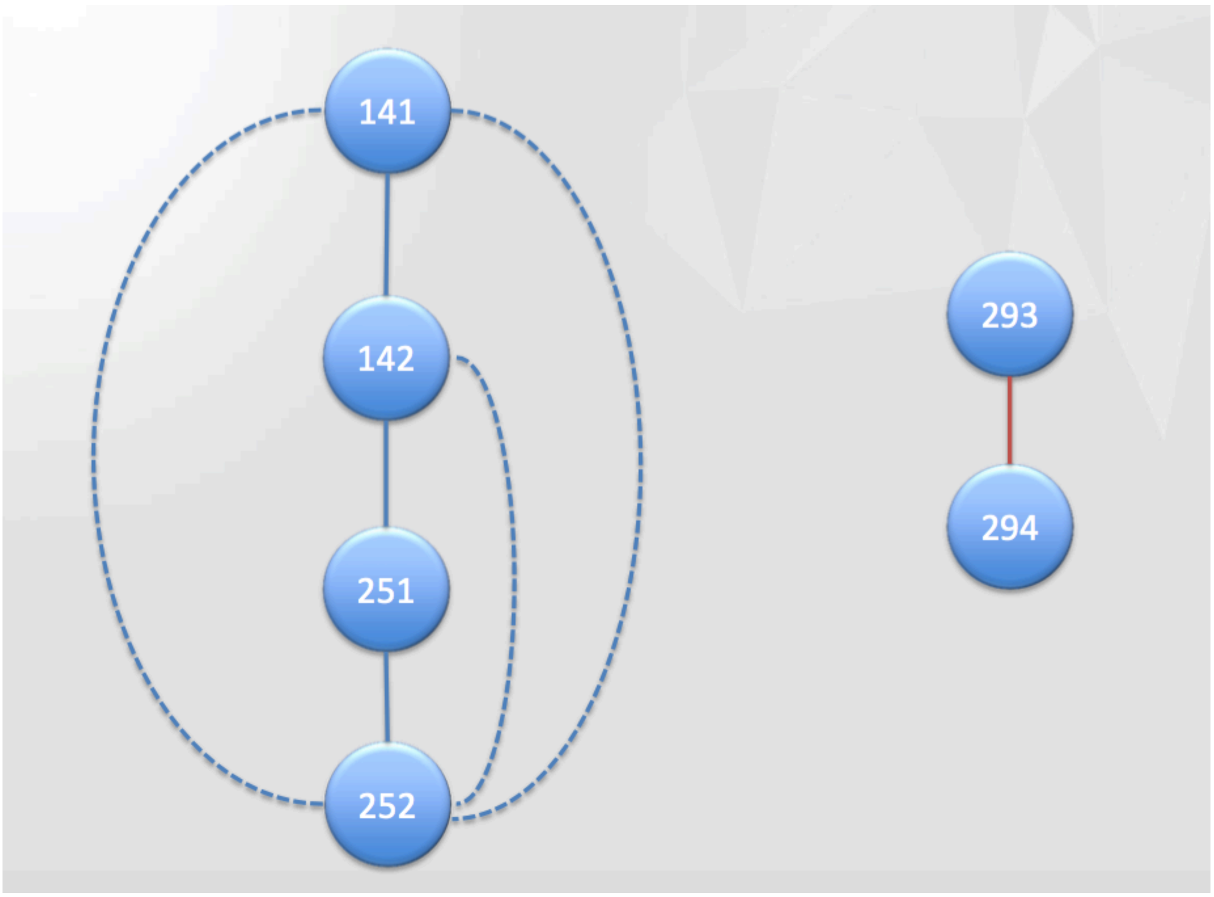
Firstly we use java to trait our graph. And we can get all relations for each agent.

For each agent, we will definite the 4 relations:
**Parent**
**PseudoParent**
**Children**
**PseudoChildren**

This graph shows us how the algorithm DFS works,



It will choose a point as its root and walk through all points in sense depth, and we can get a new DFS tree graph:



Each solid line means a relation between parent and children.
Each dotted line means a relation between pseudo parent and pseudo children.

## 7 - Jacamo plate-form

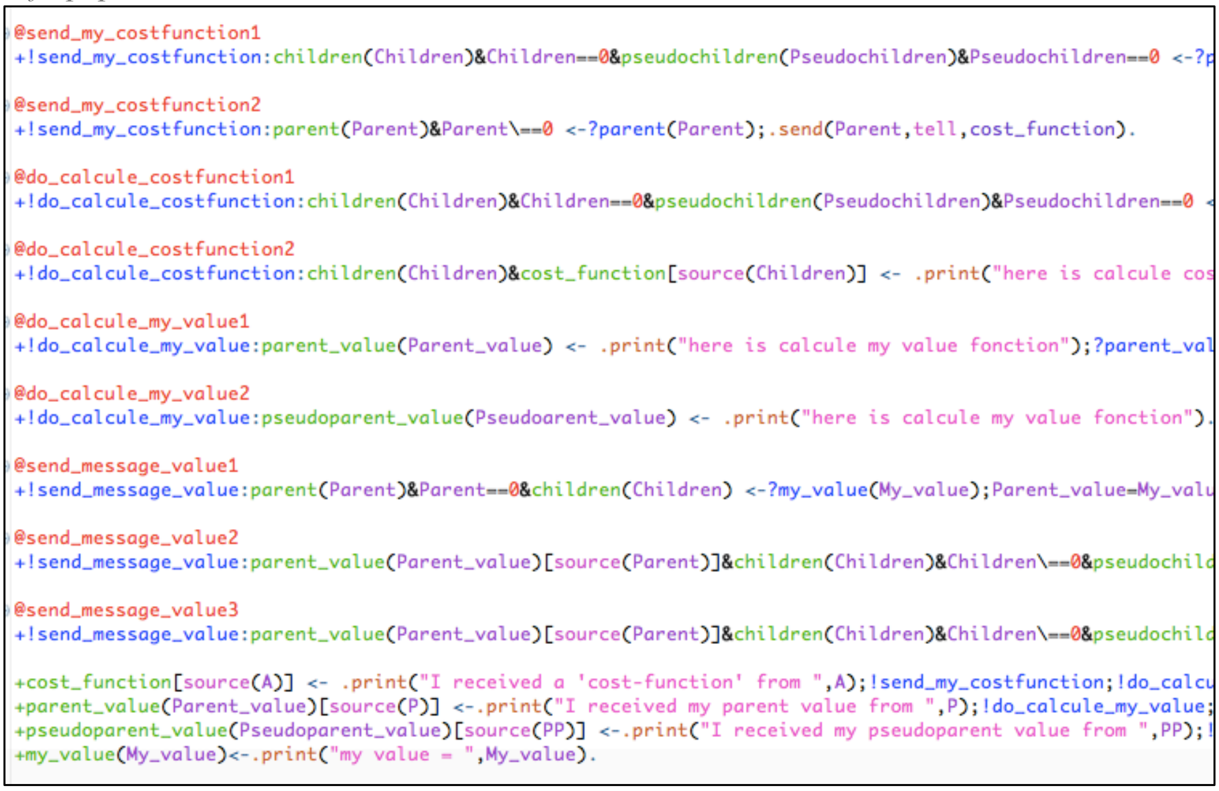
Then we will use Jacamo to simulate the **Utile message Phase** and the **Value message Phase**.

**Utile message Phase** (From leaves to the root.):
Each agent receives from their children's cost function and calculates theirs own cost function and delivery it to their parent.

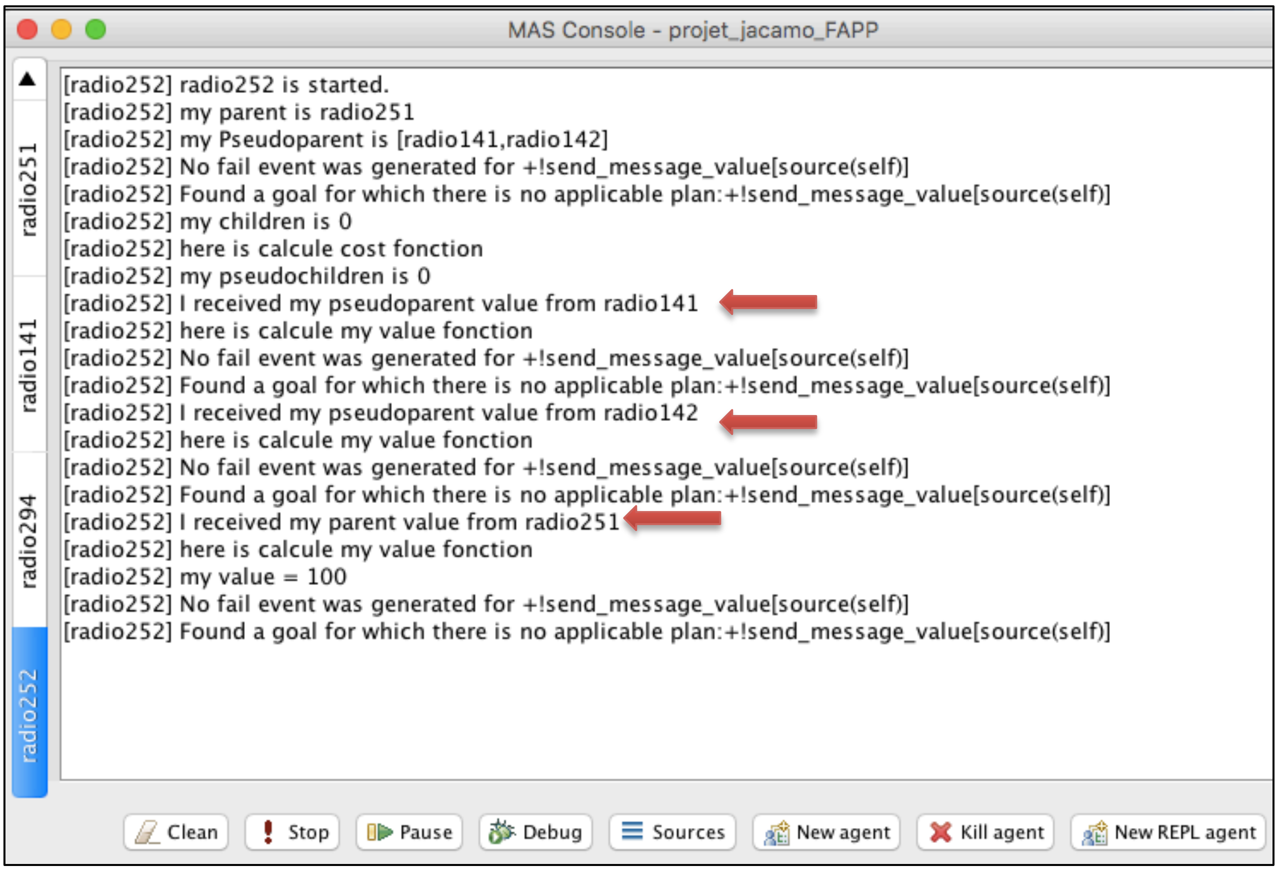
**Value message Phase** (From root to the leaves.)**:**
The root begins to inform theirs children and pseudo children the value, which it will choose. And the children will calculate the cost function and choose the value, which have a lowest cost. Then deliver it to their Children and pseudo Children.

We realise these processes by using Jason **.send** function. We add the different belief and goals in Dpop file and it will execute when the context is satisfied.

*Ref:dpop.asl*



```
@send_my_costfunction1
+!send_my_costfunction:children(Children)&Children==0&pseudochildren(Pseudochildren)&Pseudochildren==0 <-?p

@send_my_costfunction2
+!send_my_costfunction:parent(Parent)&Parent\==0 <-?parent(Parent);.send(Parent,tell,cost_function).

@do_calcule_costfunction1
+!do_calcule_costfunction:children(Children)&Children==0&pseudochildren(Pseudochildren)&Pseudochildren==0 <

@do_calcule_costfunction2
+!do_calcule_costfunction:children(Children)&cost_function[source(Children)] <- .print("here is calcule cos

@do_calcule_my_value1
+!do_calcule_my_value:parent_value(Parent_value) <- .print("here is calcule my value fonction");?parent_val

@do_calcule_my_value2
+!do_calcule_my_value:pseudoparent_value(Pseudoarent_value) <- .print("here is calcule my value fonction").

@send_message_value1
+!send_message_value:parent(Parent)&Parent==0&children(Children) <-?my_value(My_value);Parent_value=My_valu

@send_message_value2
+!send_message_value:parent_value(Parent_value)[source(Parent)]&children(Children)&Children\==0&pseudochild

@send_message_value3
+!send_message_value:parent_value(Parent_value)[source(Parent)]&children(Children)&Children\==0&pseudochild

+cost_function[source(A)] <- .print("I received a 'cost-function' from ",A);!send_my_costfunction;!do_calcu
+parent_value(Parent_value)[source(P)] <-.print("I received my parent value from ",P);!do_calcule_my_value;
+pseudoparent_value(Pseudoparent_value)[source(PP)] <-.print("I received my pseudoparent value from ",PP);!
+my_value(My_value)<-.print("my value = ",My_value).
```



```
[radio252] radio252 is started.
[radio252] my parent is radio251
[radio252] my Pseudoparent is [radio141,radio142]
[radio252] No fail event was generated for +!send_message_value[source(self)]
[radio252] Found a goal for which there is no applicable plan:+!send_message_value[source(self)]
[radio252] my children is 0
[radio252] here is calcule cost fonction
[radio252] my pseudochildren is 0
[radio252] I received my pseudoparent value from radio141
[radio252] here is calcule my value fonction
[radio252] No fail event was generated for +!send_message_value[source(self)]
[radio252] Found a goal for which there is no applicable plan:+!send_message_value[source(self)]
[radio252] I received my pseudoparent value from radio142
[radio252] here is calcule my value fonction
[radio252] No fail event was generated for +!send_message_value[source(self)]
[radio252] Found a goal for which there is no applicable plan:+!send_message_value[source(self)]
[radio252] I received my parent value from radio251
[radio252] here is calcule my value fonction
[radio252] my value = 100
[radio252] No fail event was generated for +!send_message_value[source(self)]
[radio252] Found a goal for which there is no applicable plan:+!send_message_value[source(self)]
```

In our example, it's easy to find that 252 as the lowest leaves, it gets the value from their **pseudoparent(141,142)** and **parent**(251) and then it will calculate its own value.

Although we meet some problem at the coding the calculate processes of cost function in our Artifact, En generally, we have succeed the process Dpop.

We realise the 3 phrases:
(1) Create DFS Tree
(2) Utile message Phase: from leaves to the root.
(3) Value message Phase: from root to leaves.

And we can see the whole processes of communication between different agents.

## 8 - Behaviours

Each time we start the project, each agent will update their belief base and add new goals. Then they will try to realise every goals. But some goals can't be realise immediately, so it should wait for the other agents send the new belief to him. When their context de goals are satisfied, they will achieve theirs goals.

1
- Step 1 – Perceiving the Environment

2
- Step 2 – Updating the Belief Base

3
- Step 3 – Receiving Communication from Other Agents

4
- Step 4 – Selecting 'Socially Acceptable' Messages

5
- Step 5 – Selecting an Event

6
- Step 6 – Retrieving all Relevant Plans

7
- Step 7 – Determining the Applicable Plans

8
- Step 8 – Selecting One Applicable Plan

9
- Step 9 – Selecting an Intention for Further Execution

10
- Step 10 – Executing One Step of an Intention

## 9 - Conclusion

For this study, firstly we analyse a real world problem and translate it as a DCOP. Then Jacamo and Java permit us to make a simulation to find the solution of the problem. We realise a whole process of find a solution which is from the real world.

**- In our case:**

(1) The frequency allocation.

(2) DFS

(3) Theory DPOP

(4) Programming Jacamo

We understand how the multi-agent system works and we can use Jacamo to resolve the real world problem.