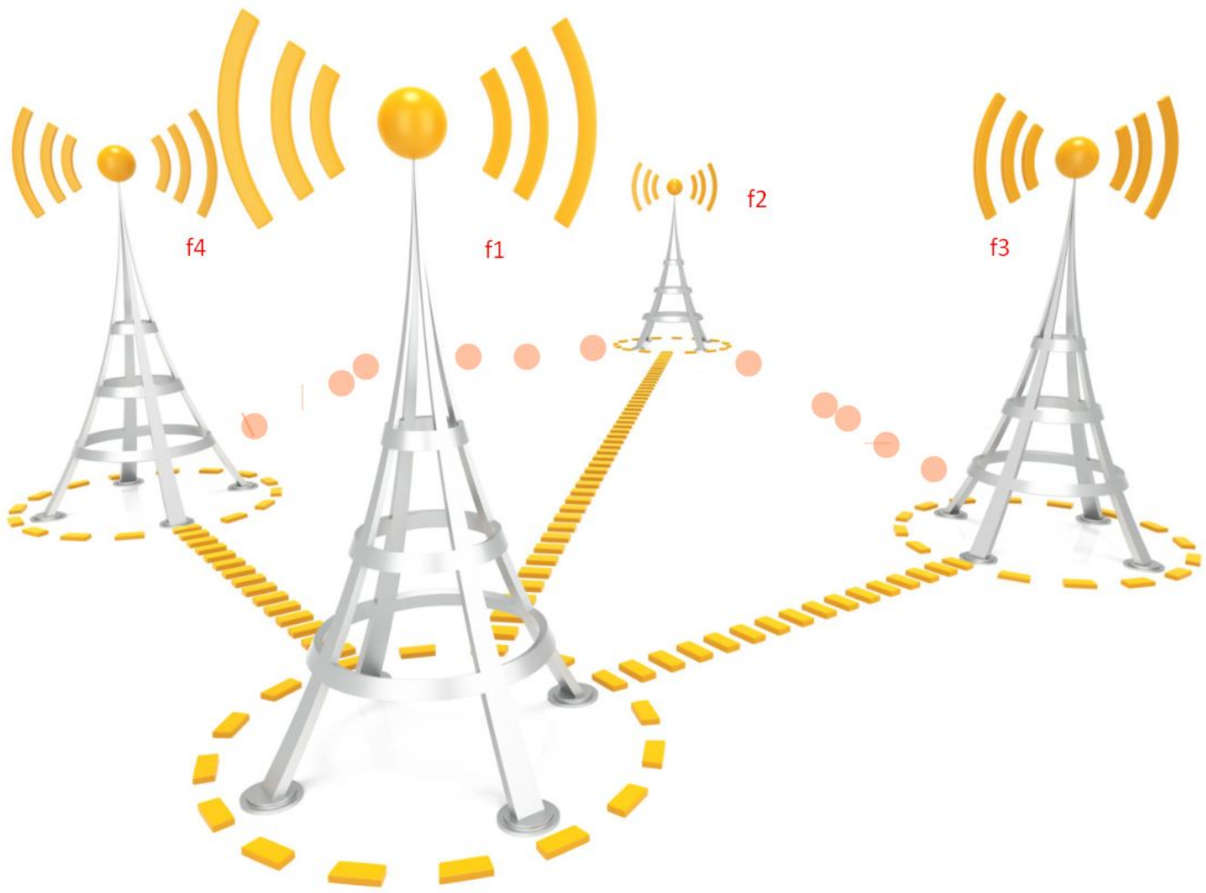
Multi-Agent System Coordination Course

The CELAR Radio Link Frequency Assignment Problems

Abdelhamid ALAOUI, 10/01/2020



Introduction

Objective

We want to develop a multiagent system to solve frequency assignment problems (FAP). We propose to map the FAP as a distributed constraint optimization problem . This DCOP should solve using the DPOP algorithm we have studied during the class . the problem has to deal with a huge amount of constraints and variables. In order for that, we will use an open source framework for distributed constraint optimization in Java called FRODO.

The french "Centre d'Électronique de l'Armement" (CELAR) has made available, in the framework of the European project EUCLID CALMA (Combinatorial Algorithms for Military Applications) s set of Radio Link Frequency Assignment benchmark problems (RLFAP) build from a real network, with simplified data. These benchmarks had been previously designed by the CELAR to assess several different Constraint Programming languages. These benchmarks are extremely valuable as benchmarks for the CSP community and more largely for constraint programming, and are available here.

In this report, we will describe the problem as given in the main problem description, and also provide a description about data we are going to use and finally how we will modelise the problem the way we understood it.

Choice of the technology

There is another powerful frameworks in this field, especially in Python such as the PyDcop library developed by Orange in France, but for some reasons, we are conducted to deal with this type of problems in JAVA using FRODO.

Problem Description

In a nutshell, 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 F1 et F2:

$$|F1 - F2| > k12$$

these two variables represent literally two radio links which supposed to be close to each other, it obviously could cause interferences, so we have to create a network of radio link antennas such as we reduces frequencies so that we don’t get such interferences, which is the main objective of this project.

the constant K12 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 out of the scope of our work. We are more interested in solving the problem instead of the actual physical details.

additionally, for each two radio links, two frequencies must be assigned in a way that one is for the communications from A to B and the other is for the communications from B to A. In the case of the CELAR instances, a technological constraint appears which states that the distance in frequency from A to B and from B to A must be exactly equal to 238. we can consider that as a second constraint in the real physical world.

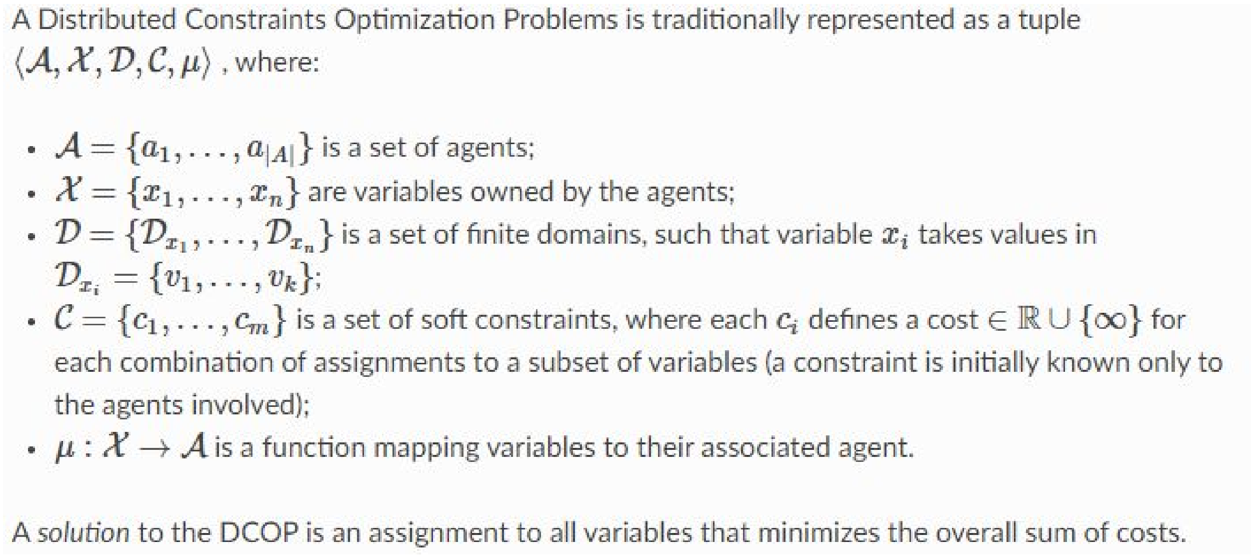
The criteria to optimize

in our problem, to evaluate the obtained frequency assignments, we are directed to propose solutions such as we could add new radio links to our network, easily. at this stage, there is two main criteria proposed in our project.

1. The first one stands for the minimization of the frequency values used: the frequencies above the last frequency used can be tried for new radio links. this criterion is a Min-Max type criteria, it is usually less expensive to optimize from an algorithmic view point.
2. The second criterion is minimizing the number of used frequencies as there will be data links far from each other so they can use the same frequencies. we will be focusing more on this criterion through our project modelisation.

Provided Data

Definition of a DCOP:



A Distributed Constraints Optimization Problems is traditionally represented as a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{C}, \mu \rangle$, where:

- $\mathcal{A} = \{a_1, \ldots, a_{|A|}\}$ is a set of agents;
- $\mathcal{X} = \{x_1, \ldots, x_n\}$ are variables owned by the agents;
- $\mathcal{D} = \{\mathcal{D}_{x_1}, \ldots, \mathcal{D}_{x_n}\}$ is a set of finite domains, such that variable x_i takes values in $\mathcal{D}_{x_i} = \{v_1, \ldots, v_k\}$;
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ is a set of soft constraints, where each c_i defines a cost $\in \mathbb{R} \cup \{\infty\}$ for each combination of assignments to a subset of variables (a constraint is initially known only to the agents involved);
- $\mu : \mathcal{X} \rightarrow \mathcal{A}$ is a function mapping variables to their associated agent.

A *solution* to the DCOP is an assignment to all variables that minimizes the overall sum of costs.

In fact, our domains, variables, constraints and criteria are all provided in four files in a single directory, for each problem instance.

merging together these files in a certain way (XCSP format) will allow to describe the details of the problem.

- The var.txt file describes all the variables in the instance. Each line corresponds to one variable using 4 fields from which only the two first fields are always present [the variable number and the domain number for the variable (which is described in dom.txt file)]
- The dom.txt file describes the domains used by the variables of the problem. Each line describes one domain.
- The ctr.txt file describes the constraints of the instance. Each line defines a binary constraint.
- The cst.txt file It defines the criteria to be optimized on the instance a priori, which actually precise the cost of soft and hard constraints as guidance to the best feasible solution.

More details about those files and syntax are available in the documentation of the CELAR problem (docs).

Modeling the problem

Talking about modeling, we are directed to transform the problem to another format to fit for FRODO.

In order for that, let's model this problem by providing the input for FRODO and describing the meaning of each component.

Objective function

The goal is to Minimize a weighted sum of the violated constraints (such as ***|F1 - F2 | > k12***)

$$\text{Min}: \sum_{i=1}^{4} C_{bi}$$

Cost function

- if ***|F1 - F2 | >= k12*** is verified the cost C equals to **0**.
- else: the cost C equals to **1**

In fact, there is up to 11 instances in this project, but since we are not supposed to evaluate all of them, this modelization concerns instances from 1 to 3 provided.

In the other instances, the change in the violation of a constraint has a variable cost, depending on the type of this constraint (soft or hard).

a0 = 10000 hard

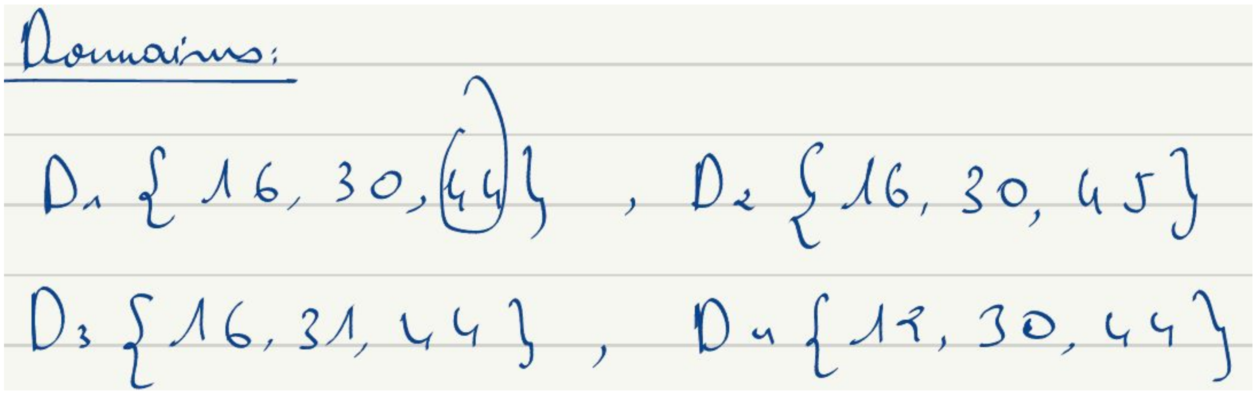
a1 = 1000 soft

a2 = 100 soft

a3 = 10 soft

There is actually a way to know which cost corresponds to which constraint violated (more details are available in the documentation of the CELAR problem (docs).

Example of 4 Domains:



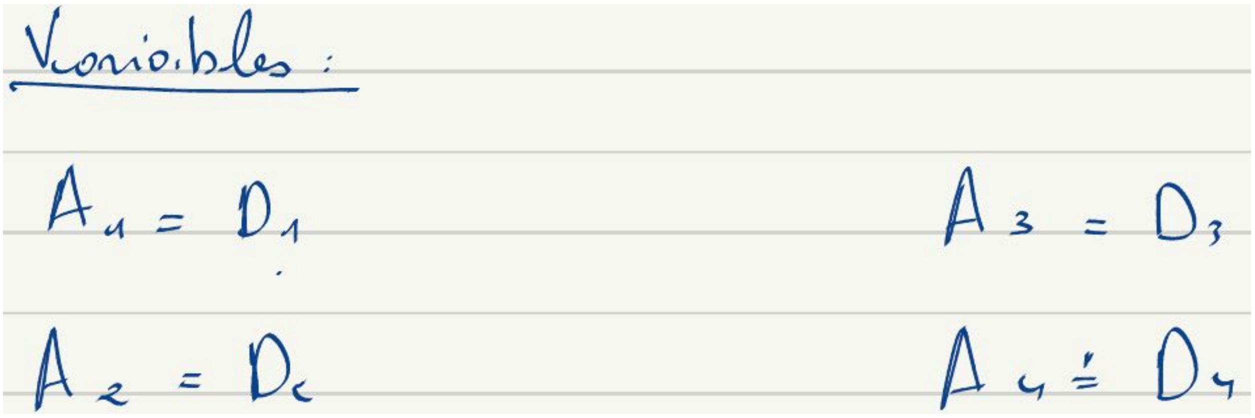
Domains:

$D_1 \{16, 30, \boxed{44}\}$, $D_2 \{16, 30, 45\}$

$D_3 \{16, 31, 44\}$, $D_4 \{12, 30, 44\}$

Agents and Variables

the strategy that i chose since is no indication on that, is that for each variable in the instance, we assign to this an agent, which literally means that each agent is in charge of one antenna. with that said and decided, the problem remain realistically applicable as well as simply modeled.



Variables:

$A_1 = D_1$ $A_3 = D_3$

$A_2 = D_2$ $A_4 = D_4$

Parser/translator

In order to utilize FRODO, we should provide an input XML file respecting a Schema called XCSP, and in order to do that, we should convert some FAP instances into XCSP instances either manually or automatically. After giving Frodo the Input data, we choose an algorithm which solves the problem by choosing a configuration file, there is several Algorithms that we should test with (more details later on).

In fact, its preferable to create instances of FAP with some real world data, instead of little instances, and to do that, we created a translator program (parser) that reads the .txt files **var.txt**, **dom.txt**, **ctr.txt** and **cst.txt** for each provided instance corresponding to the variable names, used numerical domains,... and generate a XML file respecting the XCSP format ready to use as a Frodo input.

The implementation of the parser tool to read the problem's text files is built in a ***Javascript***.

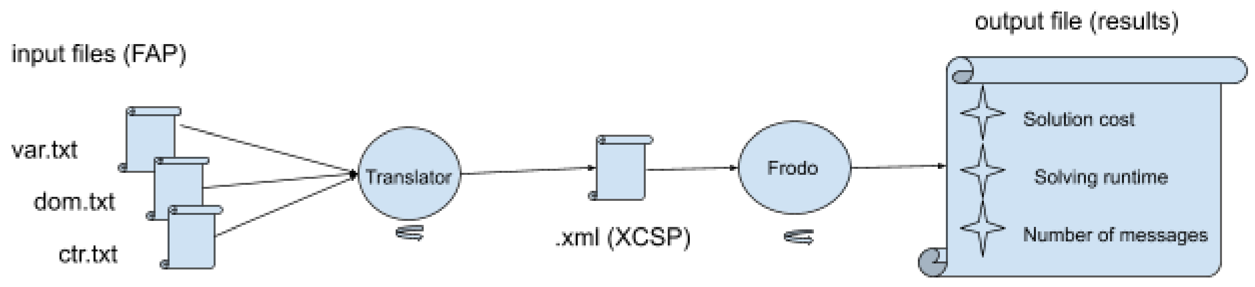
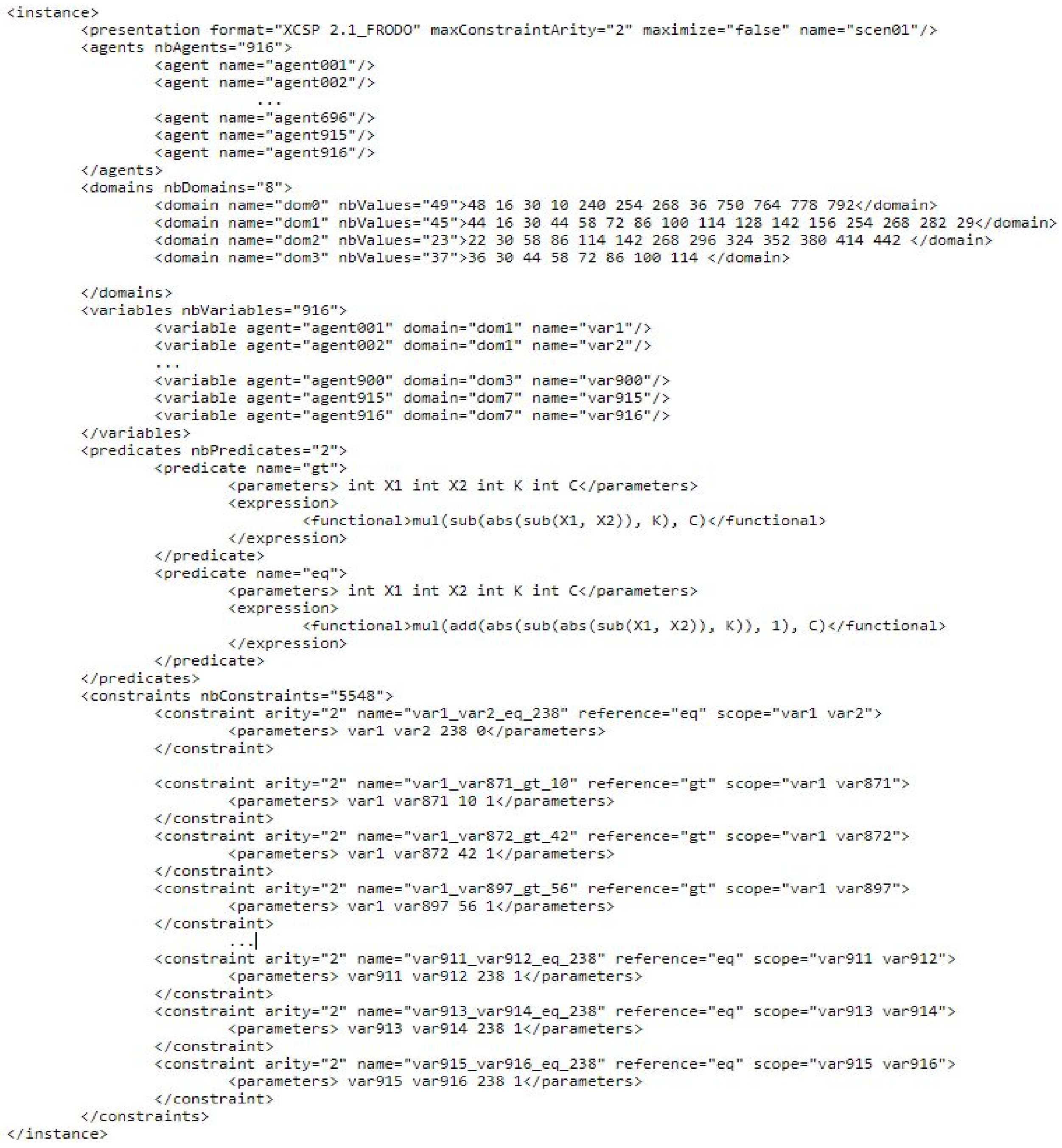


Figure1: *Overall architecture*

As you can see in the Overall architecture, the output of our program is a file that contains variables, agents, constraints and the domains of the instance on which we are applying our parser. (see figure below)



```
<instance>
        <presentation format="XCSP 2.1_FRODO" maxConstraintArity="2" maximize="false" name="scen01"/>
        <agents nbAgents="916">
                <agent name="agent001"/>
                <agent name="agent002"/>
                           ...
                <agent name="agent696"/>
                <agent name="agent915"/>
                <agent name="agent916"/>
        </agents>
        <domains nbDomains="8">
                <domain name="dom0" nbValues="49">48 16 30 10 240 254 268 36 750 764 778 792</domain>
                <domain name="dom1" nbValues="45">44 16 30 44 58 72 86 100 114 128 142 156 254 268 282 29</domain>
                <domain name="dom2" nbValues="23">22 30 58 86 114 142 268 296 324 352 380 414 442 </domain>
                <domain name="dom3" nbValues="37">36 30 44 58 72 86 100 114 </domain>

        </domains>
        <variables nbVariables="916">
                <variable agent="agent001" domain="dom1" name="var1"/>
                <variable agent="agent002" domain="dom1" name="var2"/>
                ...
                <variable agent="agent900" domain="dom3" name="var900"/>
                <variable agent="agent915" domain="dom7" name="var915"/>
                <variable agent="agent916" domain="dom7" name="var916"/>
        </variables>
        <predicates nbPredicates="2">
                <predicate name="gt">
                        <parameters> int X1 int X2 int K int C</parameters>
                        <expression>
                                <functional>mul(sub(abs(sub(X1, X2)), K), C)</functional>
                        </expression>
                </predicate>
                <predicate name="eq">
                        <parameters> int X1 int X2 int K int C</parameters>
                        <expression>
                                <functional>mul(add(abs(sub(abs(sub(X1, X2)), K)), 1), C)</functional>
                        </expression>
                </predicate>
        </predicates>
        <constraints nbConstraints="5548">
                <constraint arity="2" name="var1_var2_eq_238" reference="eq" scope="var1 var2">
                        <parameters> var1 var2 238 0</parameters>
                </constraint>

                <constraint arity="2" name="var1_var871_gt_10" reference="gt" scope="var1 var871">
                        <parameters> var1 var871 10 1</parameters>
                </constraint>
                <constraint arity="2" name="var1_var872_gt_42" reference="gt" scope="var1 var872">
                        <parameters> var1 var872 42 1</parameters>
                </constraint>
                <constraint arity="2" name="var1_var897_gt_56" reference="gt" scope="var1 var897">
                        <parameters> var1 var897 56 1</parameters>
                </constraint>
                        ...
                <constraint arity="2" name="var911_var912_eq_238" reference="eq" scope="var911 var912">
                        <parameters> var911 var912 238 1</parameters>
                </constraint>
                <constraint arity="2" name="var913_var914_eq_238" reference="eq" scope="var913 var914">
                        <parameters> var913 var914 238 1</parameters>
                </constraint>
                <constraint arity="2" name="var915_var916_eq_238" reference="eq" scope="var915 var916">
                        <parameters> var915 var916 238 1</parameters>
                </constraint>
        </constraints>
</instance>
```

In order to use this input file on Frodo, we run this command: **java -cp [frodo.jar path] frodo2.algorithms.AgentFactory [instance XML path] frodo2/agents/MGM/MGMagentJaCoP.xml > [output file path]**

When using the Frodo interface to run instances, it impacts on the runtime of the program, as a result we use the command line above.

Note that because of my computer configuration, we used 12GB of RAM, as result, we couldn't run all the DPOPAgentJacop Algorithms, because of the performance of each algorithm, some of them run in polynomial time, while others are exponential. With that said, at some point (sometimes after 2 hours), the computer could not support and outputs the runtime error.

Finally, we read the output file provided by Frodo and take the results we need for the sake of plotting a visual graph and eventually for analysis.

Results and benchmarking

In this part, we present the output of our experiments. as we mentioned before, we ran our models on 3 problems (2,3,5).

In fact, after several tries to run all the DPOPAgentJacop algorithms in FRODO, we decided to run 3 FAP instances and present a result plot with, only, the MGM Algorithm which is a polynomial time algorithm, since, the instances of the FAP have a huge number of variables and constraints, and also the limited hardware resources that we have.

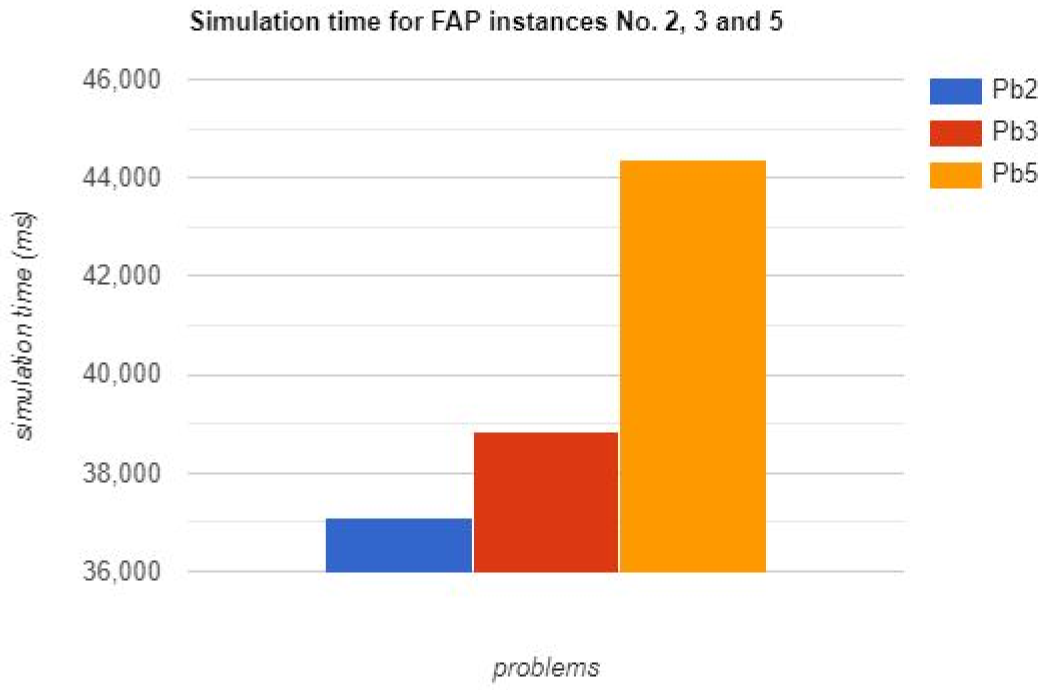
In the following table, there are the results of each experiment:

Problem No.	Variables	Optimal Cost	Simulation Time (in ms)	Messages
2	200	0	37,100	988,000
3	400	32	38,842	1,400,000
5	400	221	44,378	25,800,000

We put forward the number of each instance with the number of variables, the optimal total cost, simulation in time in milliseconds and the number of sent messages among all agents. we can see from this table that the problem 2 has been solved successfully with 0 constraint violation, which means the problem is feasible, not like the other ones, since it doesn't contain any soft constraints with multiple costs (Modeling). Whereas the problem 3 and 5 had some constraint violation.

with that said, we can say that the number of variables and agents is not the only factor for the total optimal cost. it also depends on the used violation cost.

Finally, we observed that the number of messages per agent is proportional to the number of agents involved in solving the FAP.



References

1. https://stackoverflow.com/questions/14340894/create-xml-in-javascript
2. http://www7.inra.fr/mia/T/schiex/Doc/CELAR.shtml
3. https://manual.frodo-ai.tech/FRODO_User_Manual.html#x1-80004