Multi-Agent System

Quentin Bissuel and Maleine Patural

January 2020

Contents

1 Introduction

The multi-agent oriented is a new way of programming. Before there was the imperative method and after that, the oriented object method.
Both are good but they fail to solve problems that have always been a hot topic in computer science : the travelling salesman problem.
They do not fail to solve it strictly speaking, but they fail to do it in an appropriate amount of time, if the graph is big enough.

Multi-agent collaboration system allow us to compute larger problems, without taking as much time as the others methods. By giving each agent a task and another agent to refer to, sometimes more than one, it allow us to compute a single problem on several CPU, and therefore splitting the said problem into smaller pieces, easier to solve.

This project's objective is to show some examples on how to use multi-agent collaboration to solve some graph problems, using the FRODO framework and Python.

2 Setup

2.1 FRODO

FRODO is a framework used to convert a structured XML file into a multi-agent problem. It was developed by Thomas Léauté, Brammert Ottens and Radoslaw Szymanek.
It is possible to produce a visual graph with graphviz, but FRODO is mainly used to give us a solution to a problem, writing this solution on a file

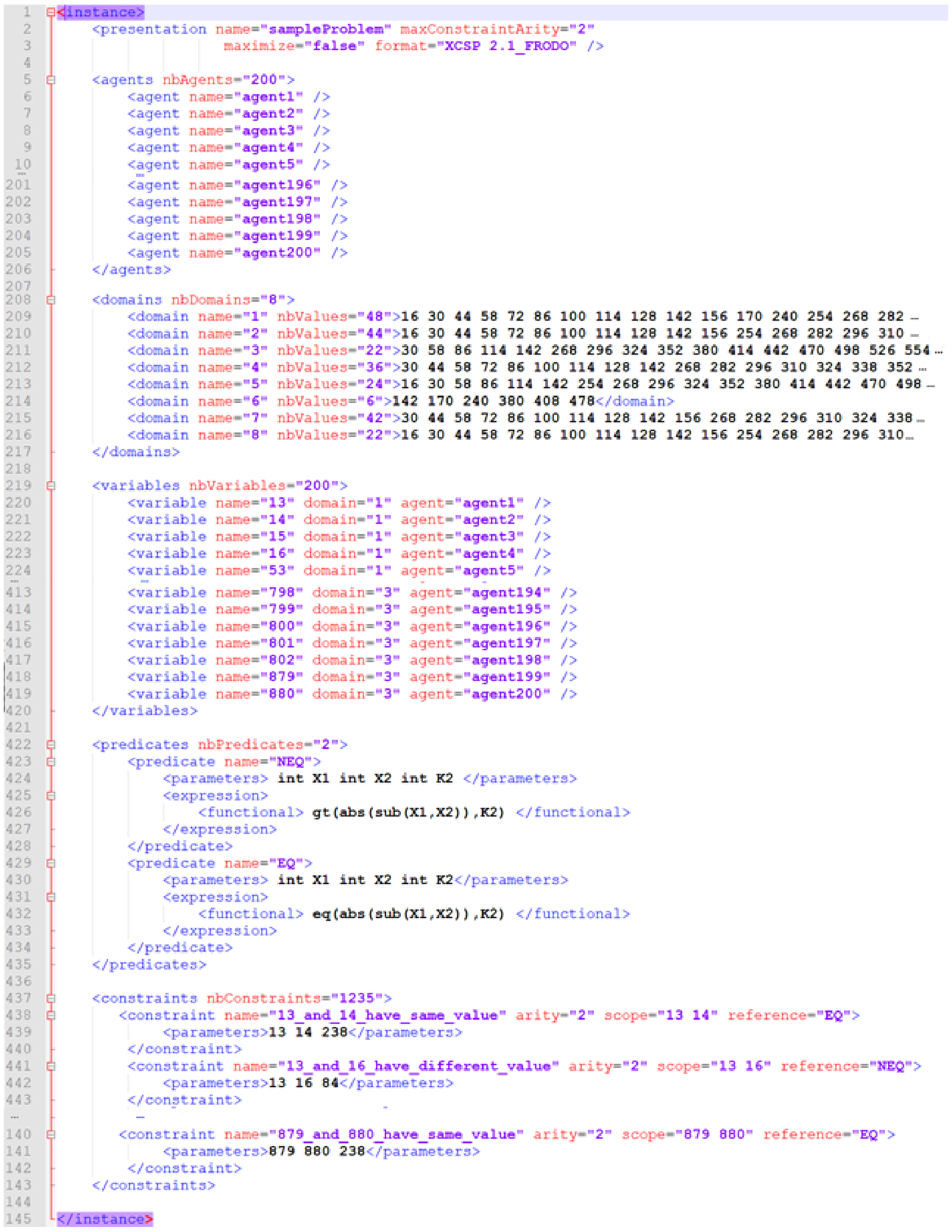
2.2 CELAR

CELAR is the french "Centre d'Électronique de l'Armement", it has made available a lot of data that we use. We used it to create our XML files problems.
The data is written on 4 text files.

- var : gives the list of variables
- dom : gives the domain's definition
- ctr : gives all the constraints
- cst : define the problem, the criteria to optimize

3 Parser and XML files

The XML are written with the parsing of the 3 CELAR files. Here is an



```xml
<instance>
    <presentation name="sampleProblem" maxConstraintArity="2"
                maximize="false" format="XCSP 2.1_FRODO" />

    <agents nbAgents="200">
        <agent name="agent1" />
        <agent name="agent2" />
        <agent name="agent3" />
        <agent name="agent4" />
        <agent name="agent5" />
        <agent name="agent196" />
        <agent name="agent197" />
        <agent name="agent198" />
        <agent name="agent199" />
        <agent name="agent200" />
    </agents>

    <domains nbDomains="8">
        <domain name="1" nbValues="48">16 30 44 58 72 86 100 114 128 142 156 170 240 254 268 282 …
        <domain name="2" nbValues="44">16 30 44 58 72 86 100 114 128 142 156 254 268 282 296 310 …
        <domain name="3" nbValues="22">30 58 86 114 142 268 296 324 352 380 414 442 470 498 526 554 …
        <domain name="4" nbValues="36">30 44 58 72 86 100 114 128 142 268 282 296 310 324 338 352 …
        <domain name="5" nbValues="24">16 30 58 86 114 142 254 268 296 324 352 380 414 442 470 498 …
        <domain name="6" nbValues="6">142 170 240 380 408 478</domain>
        <domain name="7" nbValues="42">30 44 58 72 86 100 114 128 142 156 268 282 296 310 324 338 …
        <domain name="8" nbValues="22">16 30 44 58 72 86 100 114 128 142 156 254 268 282 296 310…
    </domains>

    <variables nbVariables="200">
        <variable name="13" domain="1" agent="agent1" />
        <variable name="14" domain="1" agent="agent2" />
        <variable name="15" domain="1" agent="agent3" />
        <variable name="16" domain="1" agent="agent4" />
        <variable name="53" domain="1" agent="agent5" />
        <variable name="798" domain="3" agent="agent194" />
        <variable name="799" domain="3" agent="agent195" />
        <variable name="800" domain="3" agent="agent196" />
        <variable name="801" domain="3" agent="agent197" />
        <variable name="802" domain="3" agent="agent198" />
        <variable name="879" domain="3" agent="agent199" />
        <variable name="880" domain="3" agent="agent200" />
    </variables>

    <predicates nbPredicates="2">
        <predicate name="NEQ">
            <parameters> int X1 int X2 int K2 </parameters>
            <expression>
                <functional> gt(abs(sub(X1,X2)),K2) </functional>
            </expression>
        </predicate>
        <predicate name="EQ">
            <parameters> int X1 int X2 int K2</parameters>
            <expression>
                <functional> eq(abs(sub(X1,X2)),K2) </functional>
            </expression>
        </predicate>
    </predicates>

    <constraints nbConstraints="1235">
       <constraint name="13_and_14_have_same_value" arity="2" scope="13 14" reference="EQ">
            <parameters>13 14 238</parameters>
        </constraint>
        <constraint name="13_and_16_have_different_value" arity="2" scope="13 16" reference="NEQ">
            <parameters>13 16 84</parameters>
        </constraint>

       <constraint name="879_and_880_have_same_value" arity="2" scope="879 880" reference="EQ">
            <parameters>879 880 238</parameters>
        </constraint>
    </constraints>

</instance>
```

example of what our XML files look like. There is 4 important part:

- The agents
- The variables
- The domains
- The constraints

The agents and the variables have the same number, since each variable define a domain for each agent. The domains are used to define the values available, and so they are the parameters of the problem. The constraints are the most important parameters of the problem. Without them, the agent are useless, because they just choose a random value and that is it. They are used to define the problem, to define which agents cannot have the same value as another. But the problem is define by every of these 4 parts of the XML file.

4 Problems

Those problems are extracted from the CELAR database. We have 11 different problems that I will detailed.

4.1 Problem implemented and tried

4.1.1 2^{nd} problem

The parameters are:

- Agents and Variables : 200
- Domains : 8
- Constraints : 1235

The goal of this graph is to find a solution with the minimal number of used values.

4.2 Problems parsed

4.2.1 1^{st} problem

The goal of this graph is to find a solution which minimize the number of different values used for every variables.

4.2.2 3^{rd} problem

The goal of this graph is to find a solution which minimizes the number of different assigned values.

4.3 4^{th} problem

The goal of this graph is to find a solution which minimizes the largest assigned value, and if there is no solution, it minimizes the cost function with some coefficient.

4.3.1 5^{th} problem

The goal of this graph is to find a solution which minimizes the largest assigned value, and if there is no solution, it minimizes the cost function with some coefficients.

4.3.2 6^{th} problem

The goal of this graph is to find a solution which use the minimum number of values, and if there is no solution, it minimizes the cost function with some coefficients.

4.3.3 7^{th} problem

The goal of this graph is to find a solution which use the minimal number of assigned values, and if there is no solution, it minimizes the cost function according to some coefficients.

4.3.4 8^{th} problem

The goal of this graph is to find a solution which uses the minimal number of assigned values, and if there is no solution, it minimizes the cost function with some coefficients.

4.3.5 9^{th} problem

The goal of this graph is to find a solution which uses the minimal number of assigned values, and if there is no solution, it minimizes the cost function with some coefficients.

4.3.6 10^{th} problem

The goal of this graph is to find a solution which minimizes the number of assigned values, and if there is no solution, it minimizes the cost function with some coefficients.

4.4 11^{th} problem

The problem described by this graph is the CELAR radio link frequency assignment problem instances. This problem is that two frequencies should be far enough to not parasite the other, while having as much frequencies as possible.

5 Solving

5.1 What we solved

Since we do not have powerful computers, most of the problems timed out, but there was some that didn't, and the output files are included in the project. One problem that did work was the 2^{nd} one, with 3 different type of agent :

- DSA : 36 966 ms
- MGM : 55 204 ms
- AFB : 345 ms

5.2 Agents

The DSA, MGM and AFB agent system have something in common since they all use search algorithms from DCOP algorithm. Thought they have differences too, since both DSA and MGM are incomplete algorithm where AFB is a complete algorithm.
With these information, we can admit that if there is a solution, DSA and MGM will probably not find the best solution, but they would find a solution in an appropriate amount of time while AFB will probably find the best solution, but it would take more time than the others, so it would not be viable for an optimized program.

5.3 Comparison

It seems that the AFB is the most efficient one, which is not very logical. Even when we look inside the solution file, it seems that the AFB solution is way better according to the goal of the project : using a minimum of value. Instead of DSA and MGM, which have incomplete algorithm, AFB has a complete algorithm, and therefore use more time and space to test everything. But this time, it seems that it was more powerful the way it began.
However, it is possible that all the constraints are not verified in the AFB algorithm, which would explain why it was so powerful even thought it is a complete algorithm.

5.4 In addition

With our computers, we weren't able to compute much more files with differents agents. However, we manage to use Google Colaboratory, which basically allow us to use the Google computers to execute our programs, in order to compute more files, with the MGM agent system. We were able to get some solutions for every problem from 1 to 7. Unfortunately, with this method we couldn't get any information about the time it took precisely, but it wasn't a short amount of time.

6 Conclusion

Multi-agent collaboration is an interesting way of programming. It opens a lot of possibilities for every graph-oriented problems and more. We had a lot of problems and only one could be solve in an appropriate amount of time, which is not what we thought would happen. Even more intriguing, the multi-agent system that was the more powerful and efficient was the one that should have taken a lot of time, trying every value. However, it is interesting, because it shows us that, depending on how the algorithm starts, it could work a lot without getting any results, or work very little time and have the best solution. Maybe it is not the best solution, but considering the amount of time it needed, comparing to the others, it could be the best solution in some contexts.