AFDivider : Manual and Documentation

Sylvie Doutre, Mickaël Lafages, Marie-Christine Lagasquie-Schiex

AFDivider : Manual and Documentation

Sylvie Doutre

Mickaël Lafages

Marie-Christine Lagasquie-Schiex

Université de Toulouse, IRIT

{sylvie.doutre,mickael.lafages,lagasq}@irit.fr

Tech. Report

IRIT/RR–2022–02–FR

Contents

Introduction

The first part of Mickaël Lafages' PhD Thesis [11] is concerned about solving more efficiently argumentation problems that are expressed in Dung's Argumentation Framework (AF) and its semantics. In Dung's AF, solutions of an argumentation problem are sets of arguments (defined under the notion of *extension*) which, when considered together, win the argumentation. Finding all the possible solutions of an argumentation problem, *i.e.* all its winning sets of arguments, can be very time consuming. Many argumentation problem instances, particularly large,[1] are too hard to be solved in an acceptable time, as shown by the results of the ICCMA argumentation solver competition.[2] This hardness is not relative to the current state of the art but rather to the intrinsic theoretical complexity of the argumentation semantics that are tackled [10].

Considering the foreseen scaling-up challenge in addition to the complexity, there is a need for heuristics, methods and algorithms efficient enough to tackle such issues and make possible the use of automated argumentation models, even in such settings. Enhancing the computational time of enumerating the solutions of an argumentation framework has been the object of study of many works, resulting in the elaboration of several recent algorithms such as [1, 5, 13, 2] (see [6] for an overview).

To address this issue, we propose the *AFDivider* algorithm, a *distributed* and *clustering*-based algorithm that has for main purpose to find all the possible solutions of an argumentation problem. Those solutions are defined in terms of semantics labelling [4, 3], a three status based function mapping that assigns to each argument of an AF an acceptance status: *accepted*, *rejected* or *undecided*. An empirical analysis of the *AFDivider* algorithm shows that the new approach of computing Dung-like semantics is relevant and very appropriate for some types of argumentation problems. This work led to several publications: [12, 8, 7, 9].

The present report describes how to use, install and generate benchmarks with *AFDivider* solver. Chapter 1 on the next page describes the user manual of the *AFDivider* solver. Chapter 2 on page 10 describes the installation procedure of the *AFDivider* solver. Chapter 3 on page 14 describes *AFDivider* project structure. Finally, Chapter 4 on page 17 concludes this report.

For details about documentation on theoretical and analysis explanations, refer to the PhD manuscript [11]. For details on source code documentation, refer to Section 3.2 on page 16.

[1] This notion of largeness of an argumentation framework is related to the fact that the computation of the solutions is complex either because of the number of arguments, or of the number of interactions, or because of the structure of the argumentation framework.

[2] `http://argumentationcompetition.org`

Chapter 1

Solver Manual

1.1 General Presentation

The *AFDivider* solver enumerates extension-based semantics, even though the internal algorithm presented below is based on labelling semantics. It has been designed for Dung's original semantics: the ***complete***, the ***stable*** and the ***preferred*** semantics. In this section, the algorithm is briefly presented. For more information on *AFDivider* , see [11].

Given a Dung's argumentation framework $\mathcal{AF} = \langle A, K \rangle$ and a semantics $\sigma \in \{complete, stable, preferred\}$, the *AFDivider* algorithm, rather than building labellings that cover the whole AF (which could be time consuming), computes the semantics labellings using a distributed and clustering-based method. Here are its four major steps graphically represented in Figure 1.1:

1. A pretreatment on $\mathcal{AF}$ removes "trivial" parts of it.
2. Clusters in $\mathcal{AF}$ are identified.
3. The labellings under semantics σ in each of these clusters are computed **in parallel**.
4. The results of each cluster are reunified to get the labellings of $\mathcal{AF}$.

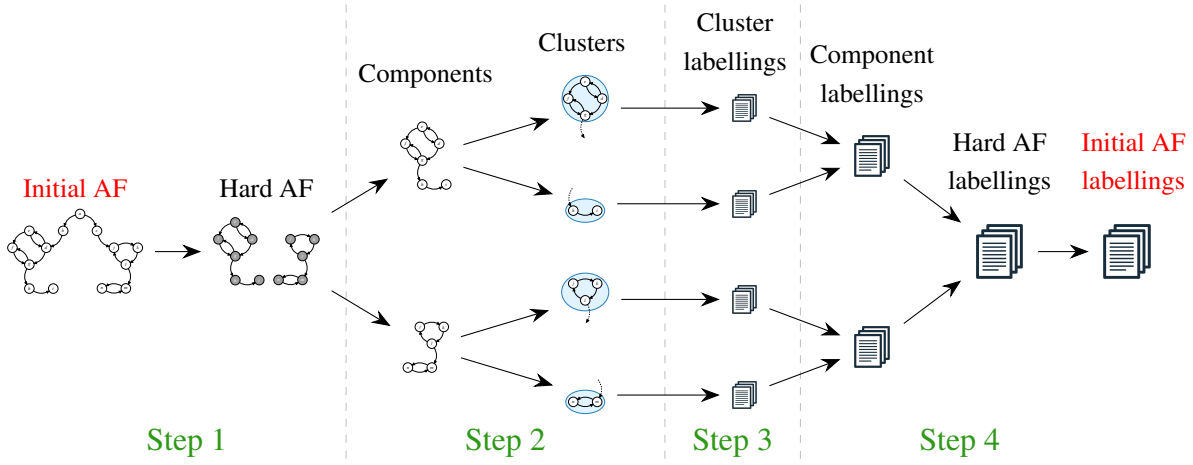

Figure 1.1: *AFDivider* operating diagram

Algorithms 1 and 2 on this page and on the next page give the formal definition of the *AFDivider* algorithm. They are said to be generic algorithms in the sense that:

- Any clustering method can be used to split the AF.
- Any sound and complete procedure that computes the semantics σ, can be used to compute the labellings of the different clusters.

Algorithm 1: *AFDivider* algorithm.

Input: Let $\mathcal{AF} = \langle A, K \rangle$ be an AF and σ be a semantics
Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(\mathcal{AF})}$: the set of the σ-labellings of $\mathcal{AF}$
Local variables:
- ℓ'_{gr}: the *grounded* labelling restricted to the arguments labelled `in` and `out`
- *CCSet*: the set of connected components of $\mathcal{AF}_{hard}$ ($\mathcal{AF}$ after the removal of the arguments labelled `in` and `out` in the *grounded* labelling)
- af_i: a connected component
- *ClustSet*: the set of cluster structures of af_i
- $\mathscr{L}_\sigma(af_i)$: the set of all σ-labellings of af_i

1 $\ell'_{gr} \leftarrow$ `ComputeGroundedLabelling`$(\mathcal{AF})$
2 $CCSet \leftarrow$ `SplitConnectedComponents`$(\mathcal{AF}, \ell'_{gr})$
3 **for all** $af_i \in CCSet$ **do in parallel**
4 $\quad ClustSet \leftarrow$ `ComputeClusters`(af_i)
5 $\quad \mathscr{L}_\sigma(af_i) \leftarrow$ `ComputeCompLabs`$(\sigma, ClustSet)$
6 $\mathscr{L}_\sigma \leftarrow \varnothing$
7 **if** $\nexists af_i \in CCSet$ *s.t.* $\mathscr{L}_\sigma(af_i) = \varnothing$ **then** $\mathscr{L}_\sigma \leftarrow \{\ell'_{gr}\} \times \prod_{af_i \in CCSet} \mathscr{L}_\sigma(af_i)$
8 **return** $\mathscr{L}_\sigma$

1.2 Input Format

So far, the AF encoding format recognized by our solver is the so-called "APX Format". The APX encoding of the AF illustrated in Figure 1.2 on page 5 is given in Figure 1.3 on page 5.

1.3 Functionalities

1.3.1 Required Arguments

Following are the required arguments and their meaning:

- `-s`: Path to the external solver

  This option specifies the path to the external solver used for computing the labelling at cluster level. See Sections 4.2.3 and 5.1 of [11], for more details.

- `-f`: Path to AF file

  This option specifies the path to the encoded AF. See the *Input Format* option.

Algorithm 2: `ComputeCompLabs` algorithm.

Input: Let *ClustSet* be a set of cluster structures for a component af, σ be a semantics

Result: $\mathscr{L}_\sigma \in 2^{\mathscr{L}(af)}$: the set of the σ-labellings of af

Local variables:

- κ_j: a cluster structure
- $\mathscr{L}_\sigma^{\kappa_j}$: the set of all σ-labellings of κ_j
- $\mathscr{P}^{\kappa_j}$: the set of configurations corresponding to the σ-labellings of κ_j
- $\mathscr{P}$: the set of all reunified labelling profiles

for all $\kappa_j \in ClustSet$ **do in parallel**
  $\mathscr{L}_\sigma^{\kappa_j} \leftarrow$ `ComputeClustLabs`(σ, κ_j)
  $\mathscr{P}^{\kappa_j} \leftarrow$ `IdentifyConfigs`$(\mathscr{L}_\sigma^{\kappa_j}, \kappa_j)$
$\mathscr{L}_\sigma = \varnothing$
$\mathscr{P} =$ `ReunifyCompConfigs`$(\bigcup_{\kappa_j \in ClustSet} \mathscr{P}^{\kappa_j}, ClustSet)$
for all $p \in \mathscr{P}$ **do**
  $\mathscr{L}_\sigma \leftarrow \mathscr{L}_\sigma \cup \left(\prod_{\xi \in p} \{\ell | \ell \in$ `ProfileLabellings`$(\xi, \bigcup_{\kappa_j \in ClustSet} \mathscr{L}_\sigma^{\kappa_j})\} \right)$
if $\sigma = pr$ **then** $\mathscr{L}_\sigma \leftarrow \{\ell | \ell \in \mathscr{L}_\sigma$ s.t. $\nexists \ell' \in \mathscr{L}_\sigma$ s.t. $in(\ell) \subset in(\ell')\}$
return $\mathscr{L}_\sigma$

- `-fo`: Input Format.

  So far, the solver recognizes only the APX format described in Section 1.2 on the previous page.

- `-p`: Problem Type

  This option specifies the AF problem type. A choice has to be made between the following values:

  - `EE-PR`: enumeration of the *preferred* semantics
  - `EE-CO`: enumeration of the *complete* semantics
  - `EE-ST`: enumeration of the *stable* semantics

- `--clustering-mode`: Clustering Mode

  This option specifies the clustering method to use for solving the problem. A choice has to be made between the following values:

  - `spectral`: The *spectral* clustering method is usually used for data mining. It clusters data, here AF arguments, following a certain "similarity" criterion that captures how much an argument is connected to another. For more details, see Section 5.2.1 of [11].
  - `uscc_chain`: The *USCC Chain* is a home made clustering method. It consists of creating clusters following *Strongly Connected Component* (**SCC**). Each part of the partition is an union of SCC. It differs from *USCC Tree* clustering method by the way SCC are joined together. For more details, See Section 5.2.2 of [11].
  - `uscc_tree`: The *USCC Tree* is a home made clustering method. It consists of creating clusters following SCC. Each part of the partition is an union of SCC. It differs from *USCC Chain* clustering method by the way SCC are joined together. For more details, See Section 5.2.2 of [11].

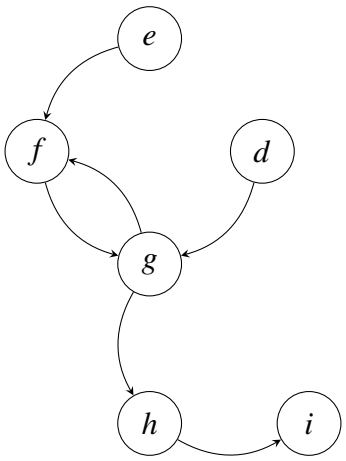

Figure 1.2: Example of an AF

```
1  arg(d).
2  arg(e).
3  arg(f).
4  arg(g).
5  arg(h).
6  arg(i).
7  att(d,g).
8  att(e,f).
9  att(f,g).
10 att(g,f).
11 att(g,h).
12 att(h,i).
```

Figure 1.3: AF Encoding

- `full_random`: The *Random* clustering method is, as indicated by its name, a random partition generation method.

1.3.2 Options

1. `--pool-size`: Thread pool size

   This option specifies the thread pool size. As explained in the source code documentation (see Section 3.2 on page 16), the tasks made in parallel, firstly at component level then at cluster level, are performed by so-called "workers". The thread-pool size indicates the number of tasks that can be performed simultaneously. For instance, if this value option is 4, then the labelling of at most 4 clusters will be computed simultaneously.

   `Note:` In the current version (March 2022), the operation of writing a particular induced AF as a text file is not made using this tread-pool but an ephemeral thread. Some development and experiments may be done to see if it is worth to make it using the thread-pool.

   As specified in Section 1.1 on page 2, the *AFDivider* algorithm can use any sound and complete procedure to compute labellings at cluster level for *induced AF*, that is the AF induced by a precise

cluster context. See Sections 4.2.3 and 5.1 of [11], for more details. The `--pool-size` option does not encompass the number of threads created by this sound and complete procedure (which is specified by the option `-s`). As an example, we used in our experiments several solver that were themselves multithreaded.

In theory, the number of the thread-pools should be related to the number of cores allocated to the solver. However, it is difficult to determined the best value given that the external sound and complete procedure may be multithreaded.

When executed on Osirim in slurm environment (See Chapter 4 on page 17), if the slurm environment variable SLURM_CPUS_PER_TASK is defined then its value is used for the pool size.

If nothing is specified the pool size is fixed to 3.

The priority for setting the pool size value is as follows:

$$\text{SLURM\_CPUS\_PER\_TASK value} \succ \texttt{--pool-size} \text{ value} \succ 3 \text{ (default value)}$$

2. `--stats-filename`: Statistics log file

   This option specifies the path to the statistics log file that gathers all the execution statistics. Figure 1.4 on page 8 shows an example of file. Here is its meaning, given line by line.

   Line 1 specifies the size of the thread pool, that is the maximal number of workers. Line 2 specifies the CPU time and the Real time used to cut and cluster the AF. *`Note:` In most of the time logs, the CPU time is not relevant and should be ignored for the current version (March 2022). Indeed, the CPU time measurement should be adapted to the fact that several tasks may be done in parallel and that tasks may have subtasks executed in different threads. This value should be the sum of all CPU time of related tasks. Some development is needed to make this measurement fairer. See classes `ComponentWorker`, `ClusterWorker`, `Worker` and `WorkerManager`. In contrast all real time measurements are reliable.*

   Lines 3-12 specify the clustering made of the AF. The *FixPart* correspond to the *`in`*-labelled and *`out`*-labelled arguments of the *grounded* labelling. The component and their respective clusters are specified.

   In lines 14-42 are given the computation details of each component. The *cutting*, *clustering*, *labellings*, *CSP solving* and *maximality check* (if needed) times are first given. The *component labelling reunification* is the sum of the two last steps and then the *total component labelling* time is given. Finally, the number of labellings computed for the component is given.

   Line 44 specifies the time of all component labellings. For the real time value, it coincides thus with the time of the total component labelling time of hardest-solving component.

   Line 45 specifies the number of labellings that will be created by the cartesian product of the computed component labellings. This value is printed before the product operation.

   Line 46 specifies the time of enumeration construction.

   Line 47 specifies the printing time, whether on `stdout` or storage file (See `--storage-filename` option).

   Line 48 specifies the total resolution time.

   If no path is specified for the `--stats-filename` option, then the statistics file is printed on `stdout`.

3. `--storage-filename`: Data result storage file

   This option specifies the path to the data log file that gathers all the computed extensions.

   The default solution format (see option `--representation`) of the current version (Mars 2022) is the one used in ICCMA 2017. Figure 1.5 on page 9 gives an example of data file. Each extension is delimited by brackets as well as the set of extensions.

   If no path is specified for the `--storage-filename` option, then the data file is printed on `stdout`.

4. `--representation`: Data output format

   This option specifies the type of output to compute. A choice has to be made between the following values:

   - `verbose`: This mode is the default one. It enumerates all the extensions of an AF.
   - `compact`: This mode computes the *Compact Enumeration Representation* (See Chapter 6 of [11]) instead of enumerating all the extensions.
   - `both`: This mode computes both output type. It has been implemented to compare the resolution time of each output mode. Indeed to make a fair comparison, it has to be according to the same AF partition and all clustering methods implemented on the current version of *AFDivider* (Mars 2022) are non-deterministic.

5. `--clustering-randomness`: Clustering Random Deviation

   This option accepts as value a real between 0 and 1. When specified, a clustering is first made based on the chosen method (see option `--clustering-mode`). Then the result is altered to produce a new partition with a difference rate (from the first one) near the value specified in option.

   In the process of finding the partition with the specified deviation rate, the arguments are changed from one part to another with respect to the AF structure. The idea is to produce connected parts. An argument can be moved to another part only if it attacks or it is attacked by an argument of that part.

   The difference rate used is the recall value. See `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html` for more informations.

   The partition found with the nearest difference value to the one specified is the final chosen one.

   By default the value of the `--clustering-randomness` option is 0. That is the partition produces by the method is not altered.

```
[2022−02−22 14:52:08,679] INFO: Nb workers: 3
[2022−02−22 14:52:09,498] INFO: Cutting and clustering time: 0.723503 (CPU), 0.821713 (Real)
[2022−02−22 14:52:09,499] INFO: Clustering partition:
    FixPart has 1 cluster(s):
        ['a1', 'a10', 'a12', 'a13', 'a14', 'a16', 'a18', 'a19', 'a2', 'a20', 'a22', 'a23', 'a24', 'a26', 'a27', '
            a28', 'a33', 'a35', 'a36', 'a38', 'a39', 'a4', 'a40', 'a41', 'a44', 'a45', 'a46', 'a47', 'a5', 'a50
            ', 'a52', 'a53', 'a56', 'a6', 'a7', 'a9']
    Component 1 has 1 cluster(s):
        ['a11', 'a25', 'a42', 'a48', 'a49', 'a55']
    Component 2 has 1 cluster(s):
        ['a37', 'a54']
    Component 0 has 2 cluster(s):
        ['a17', 'a3', 'a8']
        ['a15', 'a21', 'a29', 'a30', 'a31', 'a32', 'a34', 'a43', 'a51']

[2022−02−22 14:52:10,566] INFO: Statistics of CompWorker1
    Cutting time: 0.001769 (CPU), 0.001697 (Real)
    Clustering time: 0.000558 (CPU), 0.000532 (Real)
    Labelling time: 0.231098 (CPU), 0.228234 (Real)
    CSP solving time: 0.000348 (CPU), 0.000345 (Real)
    Maximality check time: 0.000005 (CPU), 0.000004 (Real)
    Component labelling reunification time: 0.000353 (CPU), 0.000349 (Real)
    Total Component labelling time: 0.233778 (CPU), 0.230812 (Real)
    Number of extensions: 1

[2022−02−22 14:52:10,566] INFO: Statistics of CompWorker2
    Cutting time: 0.000670 (CPU), 0.000604 (Real)
    Clustering time: 0.000148 (CPU), 0.000137 (Real)
    Labelling time: 0.264533 (CPU), 0.261374 (Real)
    CSP solving time: 0.000121 (CPU), 0.000119 (Real)
    Maximality check time: 0.000005 (CPU), 0.000004 (Real)
    Component labelling reunification time: 0.000126 (CPU), 0.000123 (Real)
    Total Component labelling time: 0.265477 (CPU), 0.262239 (Real)
    Number of extensions: 2

[2022−02−22 14:52:10,566] INFO: Statistics of CompWorker0
    Cutting time: 0.701852 (CPU), 0.698055 (Real)
    Clustering time: 0.000559 (CPU), 0.000539 (Real)
    Labelling time: 0.069286 (CPU), 1.067135 (Real)
    CSP solving time: 0.001307 (CPU), 0.001346 (Real)
    Maximality check time: 0.000005 (CPU), 0.000005 (Real)
    Component labelling reunification time: 0.001312 (CPU), 0.001351 (Real)
    Total Component labelling time: 0.773009 (CPU), 1.767079 (Real)
    Number of extensions: 2

[2022−02−22 14:52:10,567] INFO: Components labelling time: 0.070453 (CPU), 1.068502 (Real)
[2022−02−22 14:52:10,567] INFO: Nb extensions to construct: 4
[2022−02−22 14:52:10,567] INFO: Enumeration construction time: 0.000230 (CPU), 0.000229 (Real)
[2022−02−22 14:52:10,567] INFO: Printing time: 0.000153 (CPU), 0.000154 (Real)
[2022−02−22 14:52:10,567] INFO: Total time: 0.794339 (CPU), 1.890597 (Real)
```

Figure 1.4: Statistics log file example

```
1 [[a37,a48,a42,a49,a15,a30,a43,a8,a4,a5,a7,a9,a10,a12,a13,a20,a24,a27,a39,a41,a46,a47,a52,a53],[a37,a48,a42,a49,a21,
    a30,a15,a8,a4,a5,a7,a9,a10,a12,a13,a20,a24,a27,a39,a41,a46,a47,a52,a53],[a54,a48,a42,a49,a15,a30,a43,a8,a4,a5,
    a7,a9,a10,a12,a13,a20,a24,a27,a39,a41,a46,a47,a52,a53],[a54,a48,a42,a49,a21,a30,a15,a8,a4,a5,a7,a9,a10,a12,a13
    ,a20,a24,a27,a39,a41,a46,a47,a52,a53]]
```

Figure 1.5: Data file example

Chapter 2

Project Installation

2.1 Pre-requisite Software Installations

The following softwares must be installed on your client development environment:

- Python3
- An SSH client
- The version control system *Git*. In addition to Git, you can also install a graphical interface to git, such as: *SourceTree* or *Tortoise Git* or any other.
- An IDE will help a lot. The development of *AFDivider* have been made using *Pycharm*. In the following, we will use Pycharm in the installation guide.

`Note:` *All the mentioned softwares have a fully functional and free version.*

`Note:` *You will find online installation guide for each software and following your operating system.*

2.2 Git Repository: Download and Installation

The git repository is accessible at: `https://gitlab.irit.fr/argumentation/afdivider`

We recommend you to use *ssh* to connect to the repository and to use an ssh-key.
To do that:

1. Generate an ssh public key, if not done yet, as explain there: `https://www.scaleway.com/en/docs/console/my-project/how-to/create-ssh-key/`
2. Copy your public key (see the previous tutorial), go to: `https://gitlab.irit.fr/-/profile/keys` and add your key to the IRIT Gitlab.
3. To verify that every think is ok, enter the following command in your terminal:

```
1        ssh -T git@gitlab.irit.fr
```

Then, to clone *AFDivider* git repository from command line, do the following:

```
1  $ git clone git@gitlab.irit.fr:argumentation/afdivider.git
```

You can also clone the git repository directly from your graphical git tool.

```
# For linux: sudo apt-get install libblas-dev liblapack-dev mpich

brew install openblas lapack mpich

pip3 install numpy mpi4py
pip3 install petsc petsc4py
pip3 install slepc slepc4py
pip3 install scikit-learn

python3 -mpip install matplotlib
pip3 install graphviz
pip3 install script
pip3 install scipy
pip3 install futures
pip3 install python-constraint
pip3 install unittest
pip3 install sphinx sphinx-argparse sphinx-rtd-theme
```

Figure 2.1: Third libraries installation script

2.3 Third Party Libraries Installation

In the git repository there is a script to install the required third library package, called `packageInstall.sh`. This script has been written for mac but indications are given to adapt it for linux, as shown in Figure 2.1.

`Note:` The installation of `petsc` and `slepc` may take some time

2.4 Importing the Project into Pycharm

To import *AFDivider* project into Pycharm, follow this tutorial:

`https://www.jetbrains.com/help/pycharm/importing-project-from-existing-source-code.html`

2.5 Launch Configuration Settings

You can create as examples the following launch configurations :

- *AFDivider* Solver launch configuration:
  - Script path: `<PathToYour>/AFDivider/main.py`
  - Parameters: `-s <pathToInternalSolver> -fo apx -p EE-PR -f <pathToAfFile> --clustering-mode <clusteringMode> --representation <outputMode>`
  - Python interpreter: `<path_to_your_python_3>`
- Data plot configuration:

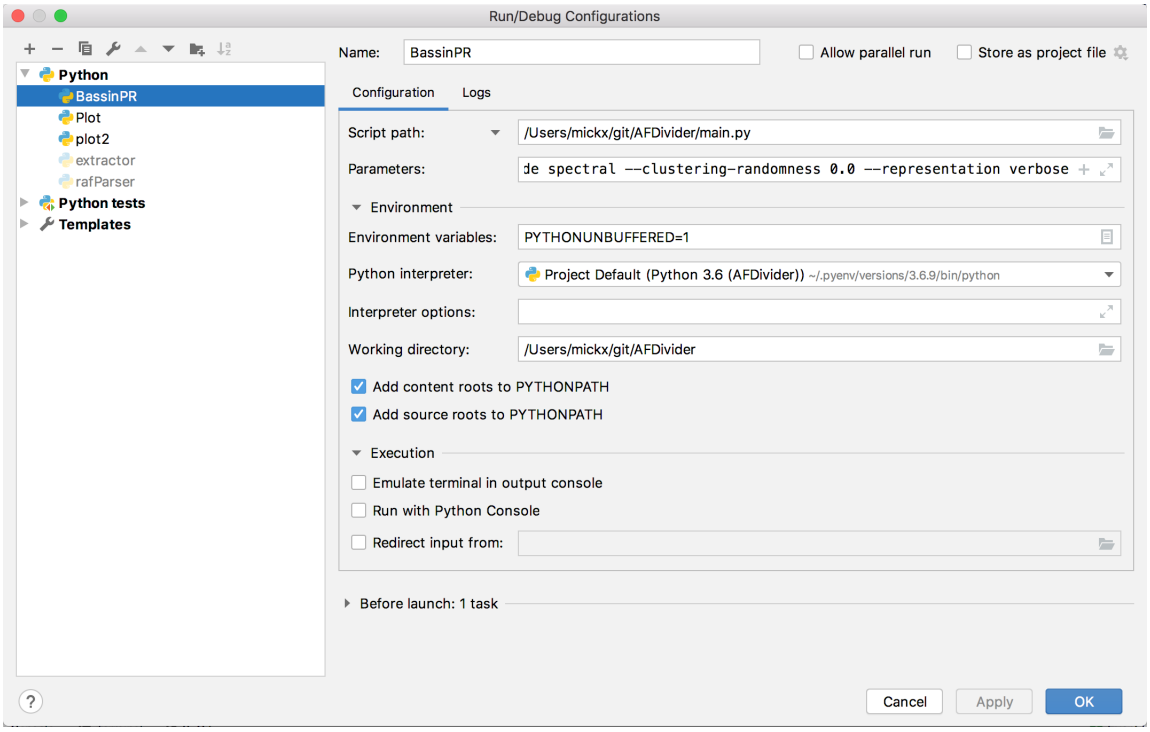

Figure 2.2: *AFDivider* launch configuration example

- Script path: `<PathToYour>/AFDivider/dataScript/extractor.py`
- Parameters: `--mode plot --input-dataframePaths <list of dataframes>`
- Python interpreter: `<path_to_your_python_3>`

2.6 Internal solvers

Solvers can be found on Osirim at the following location:

`/users/adria/mlafages/afdivider/out/solvers`

They can be copied to your client with the following command to your current directory:

```
1  scp <your osirim account>@osirim-slurm.irit.fr:/users/adria/mlafages/afdivider/out/solvers/* .
```

Then, when running *AFDivider* a solver can be specified with the option `-s`.

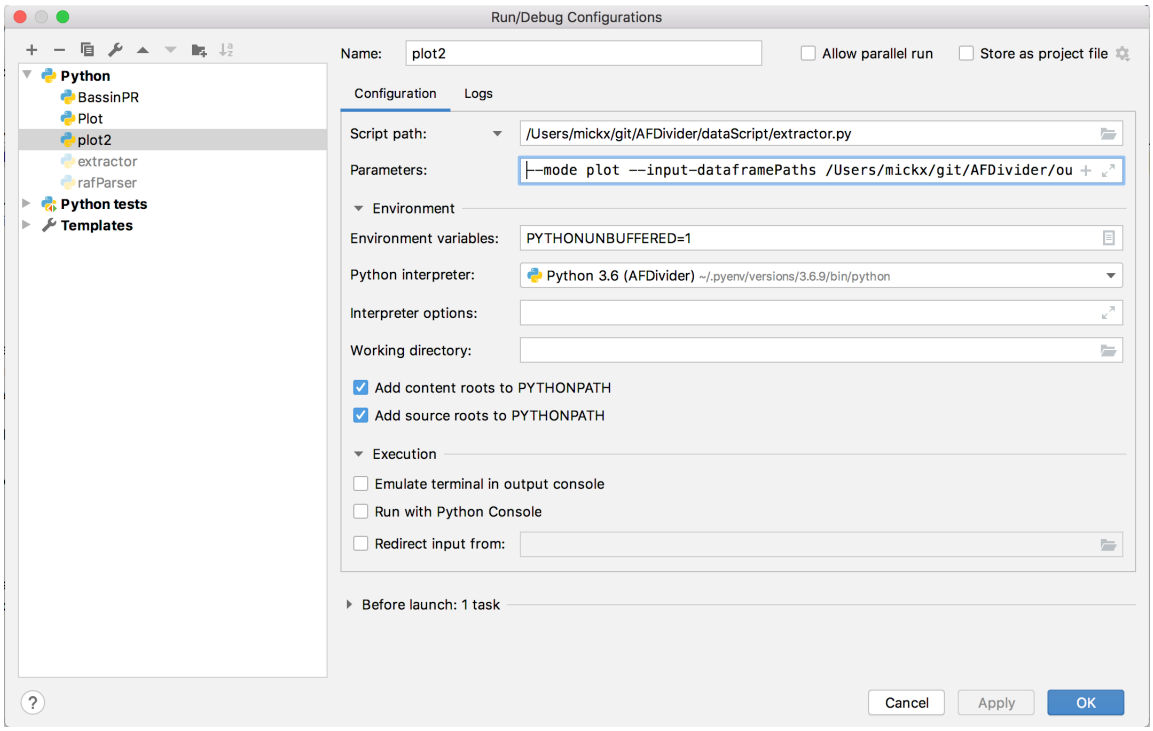

Figure 2.3: Data plot configuration example

Chapter 3

Project Documentation

3.1 Project Structure

AFDivider project structure is illustrated in Figure 3.1 on the following page.

At the directory root, we have:

- The python script `main.py`: the entry point of *AFDivider* solver.
- The `packageInstall.sh` script that installs the third party library as explained in Section 2.3 on page 11
- The `.gitignore` file

The meaning of the sub-folders is:

- The `dungAf` folder contains the data structures and algorithms that are directly linked to Dung's Argumentation Framework.
- The `tools` folder contains algorithms and tools used by the main algorithm.
- The `dataScript` folder contains all that is related to data extraction and presentation.
- The `slurm` folder contains all that is related to Osirim and Slurm. Chapter 4 on page 17 details everything about it.
- The `test` folder contains automatic test scripts.
- The `dataframe` folder contains the processed data produced by different experiments on Osirim, stored as dataframes.
- The `sphinx` folder contains all the sphinx configuration file to automatically generate a documentation website for *AFDivider* project. See Section 3.2 on page 16 for more information.
- The `instances` folder contains AF instances.
- The `ChocoConnector` folder contains everything related to Choco, that is a CSP solver.

  `Note:` Experiments showed that using the python library "`constraint`" met our needs and finally, Choco is not used in AFDivider . *However we let the files in the project if needed.*

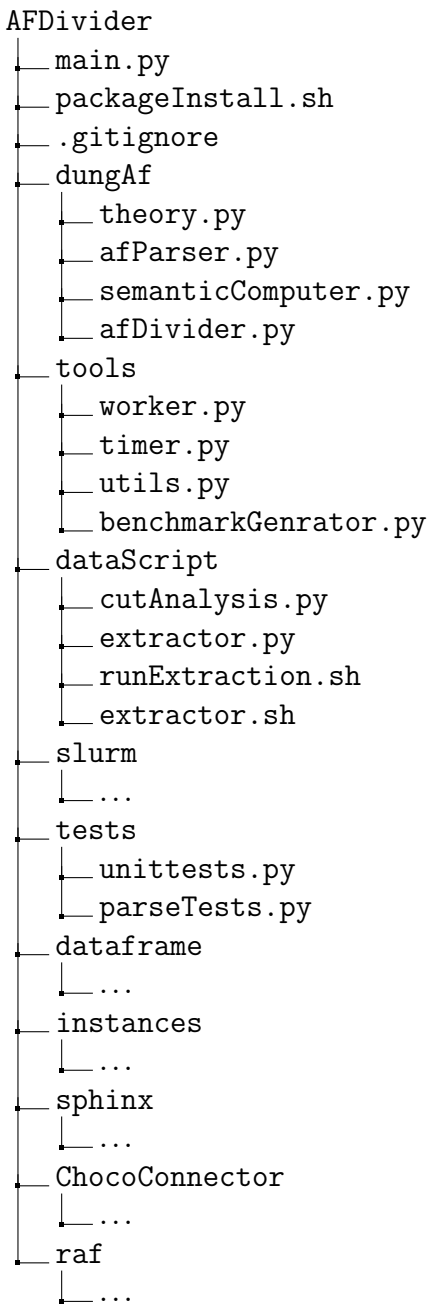

Figure 3.1: AFDivider project directory tree view

- The `raf` folder should contain the data structures and algorithms that are directly linked to Recursive Argumentation Framework.

  `Note:` *The adaptation of* AFDivider *algorithm for RAF is a perspective fully feasible relying on the work made in [11] .*

A detailled documentation of the *AFDivider* project code source is available by generating a website as explained in Section 3.2. This documentation concerns the code source of *AFDivider* algorithms as well as the data retrieval and the analysis tools developed for *AFDivider* .

3.2 Code Documentation

To generate code source documentation, we use a tool named *sphinx* that builds a local website from the python docstrings, present in the different files, describing packages, modules, classes and functions. For docstrings formatting we use the default sphinx formatting language: the *reStructuredText* language.

A full documentation of how to use sphinx and reStructuredText language is available here: `https://www.sphinx-doc.org/en/master/contents.html`.

You can see in the project code source plenty of examples on how docstrings should be made.

Given that the sphinx configuration has already been made and that this configuration is stored in the git repository, in this section we focus on how to update the documentation site after some code and docstring changes. If needed a short tutorial on how to generate sphinx documentation website is available here:
`https://betterprogramming.pub/auto-documenting-a-python-project-using-sphinx-8878f9ddc6e9`

In order to generate your new documentation local website, follow these steps:

```
1 cd <path_to_your_local_project_directory>/sphinx
2 # If new files have been added do:
3 sphinx-apidoc -o -f source/ ../
4 make html
5 # To open the new generated website on your web browser do:
6 open build/index.html
```

If you do not have the `open` command installed, open with your browser the file:
`file:///<path_to_your_git_repository>/sphinx/build/html/index.html`.

Chapter 4

Experimental Environment Documentation

This chapter presents the experimental environment used to analyse *AFDivider* performances. Notice that this chapter has been written in March 2022. Some changes may have occurred since.

4.1 Osirim

4.1.1 Presentation

Osirim (Observatoire des Systèmes d'Indexation et de Recherche d'Information Multimédia)[1] is one of IRIT's platforms. It offers a homogeneous environment for indexing and searching information in multimedia contents (text, sound, image, video, digital signal). Its main objectives are to host scientific projects requiring the storage and sharing of several terabytes of data to perform experiments on these large volumes, the sharing of software tools, in particular for evaluation, the sharing of reference corpora, and the dissemination of results and software.

Osirim gathers a set of corpora, evaluation tools, analysis software, and search engines with the main goal of offering a space of mutualization where researchers can exchange their knowledge and benefit from results obtained in other laboratories.

The platform is open to IRIT researchers and students working in these fields but also to anyone outside IRIT wishing to use its hardware or software resources under certain conditions.

It is accessible at: `https://osirim.irit.fr/`.

4.1.2 Infrastructure

The Osirim platform is composed of a computing cluster of 1120 cores spread over 26 computing servers + 28 Nvidia GTX 1080 TI cards spread over 7 servers as well as 3 Nvidia Quadro RTX6000 cards (24 GB RAM). This cluster is structured as follows and Figure 4.1 on the next page provides an illustration of it.

4.1.2.1 User connection nodes

These nodes ("Nœuds de connexion utilisateurs" in Figure 4.1 on the next page) are used to validate programs before launching them on the computing cluster. These interactive nodes are shared among all users and should not be used for long jobs.

[1] Translation: Observatory of Multimedia Information Indexing and Retrieval Systems

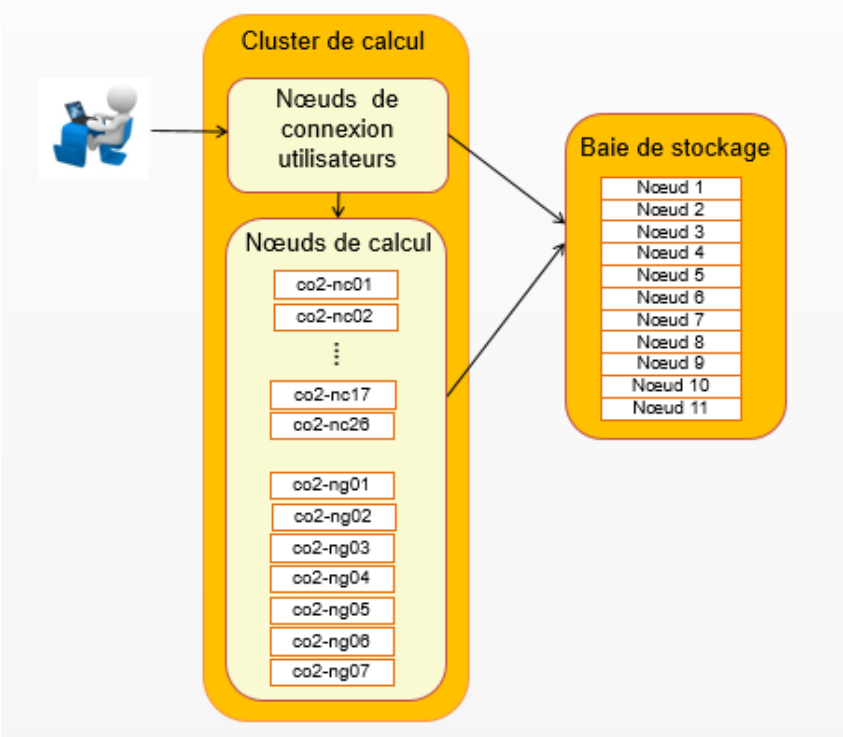

Figure 4.1: Osirim Infrastructure

4.1.2.2 CPUs compute nodes (co2-nc01, ..., co2-nc26)

These nodes are servers dedicated to computation. A process running on a compute node accesses data hosted on the storage area ("Baie de stockage" in Figure 4.1), performs processing and saves the result on this area.

The technical characteristics of the 10 compute nodes co2-nc01 to co2-nc10 are as follows:

Ten **IBM X3755 M3** servers each consisting of:

- 4 AMD Opteron 6262HE processors with 16 cores at 1.6Ghz
- 512 GB of RAM
- 2 x 300 GB of disk in RAID1
- 2 x 10 Gb/s network

The technical characteristics of the 12 co2-nc11 to co2-nc22 compute nodes are as follows:

Three **DELL C6420** chassis containing 12 servers each composed of:

- 2 Intel Xeon Gold 6136 processors with 12 cores at 3 Ghz
- 192 GB of RAM
- 2 x 256 GB of disk in RAID1
- 10 Gb/s network

The technical characteristics of the 4 co2-nc23 to co2-nc26 compute nodes are as follows:

One **DELL C6525** chassis containing 4 servers each composed of:

- 2 AMD EPYC Rome 7402 processors with 24 cores at 2.8 Ghz
- 512 GB of RAM
- 2 x 256 GB of disk in RAID1
- 10 Gb/s network

4.1.2.3 GPU computing nodes (co2-ng01, ..., co2-ng07)

These nodes are servers dedicated to GPU calculations. Each of the 7 servers is equipped with 4 Nvidia Geforce GTX 1080TI graphics cards.

The technical characteristics of these 7 servers are as follows:

Seven **DELL T630** servers:

- 2 Xeon 2640 V4 processors (20 threads)
- 2 x 400 GB SSD in RAID1
- 1 x 10 Gb/s network
- 4 Nvidia GTX 1080 TI GPUs (3584 cuda cores, 11 GB RAM)

4.1.2.4 The co2-ng09 GPU computing node

This node (that is not presented in Figure 4.1 on the previous page) is dedicated to GPU computing. It is equipped with 3 Nvidia Quadro RTX6000 graphics cards.

The technical characteristics of this server are as follows:

DELL R740 server:

- 2 Xeon Gold 6230 20 cores processors
- 2 x 223 GB SSD in RAID1
- network 1 x 10 Gb/s
- 3 Nvidia Quadro RTX 6000 GPUs (4608 cuda cores, 576 NVIDIA Tensor cores, 72 NVIDIA RT cores, 24 GB RAM)

4.1.2.5 1 PB storage array

A process running on a compute node accesses data hosted on the storage area, performs processing, and saves the result to the storage area. Data storage on the Osirim platform is provided by an EMC Isilon array consisting of:

- 11 X400 nodes with 36 SATA disks of 3 TB each
- each node is connected to the network via a trunk of 2 links of 10Gb/s
- the useful volume is about 1 PB

OneFS is the OS of the storage array that integrates the file system, volume management and data security. The whole system is a single distributed file system with a single namespace that has the ability to present data to servers using multiple protocols. NFS and HDFS are the access protocols used on Osirim.

4.1.3 Slurm

Slurm (Simple Linux Utility for Resource Management) is an open source computer task scheduling solution that allows you to create clusters of Linux servers with fault tolerance, such as ip-failover, compute farm, task scheduling system. This solution can be used on clusters of various sizes, from two to several thousand servers.

Installed on Osirim, it is via Slurm that jobs can be launched on Osirim compute nodes.

As a cluster resource manager, Slurm has three main functions:

1. It allocates access to resources (compute nodes) for users for a defined period of time so that they can perform tasks;
2. It provides a framework for starting, executing, and monitoring tasks (normally tasks that are parallel) across all assigned nodes;
3. It arbitrates access to resources by managing a queue of ongoing tasks.

See `https://slurm.schedmd.com/documentation.html` for more information on Slurm.

4.1.4 Singularity

Singularity is a container platform installed on Osirim. It allows you to create and run containers that package up pieces of software in a way that is portable and reproducible. You can thus build a container using Singularity on your computer, and then run it on any server/clusters/computer supporting Singularity. Your container is a single file in which are embedded all required library and software dependencies. As a consequence, when using Singularity, installing third libraries on Osirim is no more required. This is the practice recommended by Osirim technical team.

Figure 4.2 on the following page illustrates the difference between containers and virtual machines. Virtual machines and containers are represented by the top blocks of same color. We can see that each virtual machine (on the left side of the schema) embeds an OS, an application and software dependencies while containers (in the right side) only embed an application and software dependencies.

AFDivider project started before the installation of singularity on Osirim. Currently *AFDivider* program and dependencies are directly installed on Osirim compute cluster. However the external solvers used in the step 3 of the algorithm are in Singularity containers (See Section 4.2.2 on page 22).

Developments to embed *AFDivider* into a container is encouraged to facilitate future features. Furthermore the ICCMA competition also uses containers but *Docker* ones. There exists a way to go from one to the other container format.

4.2 *AFDivider* Experiment Environment: Osirim Resources

As an overview, for *AFDivider* experiments we used two directories in which all sources, input and output data, required and produced by the different operations are stocked:

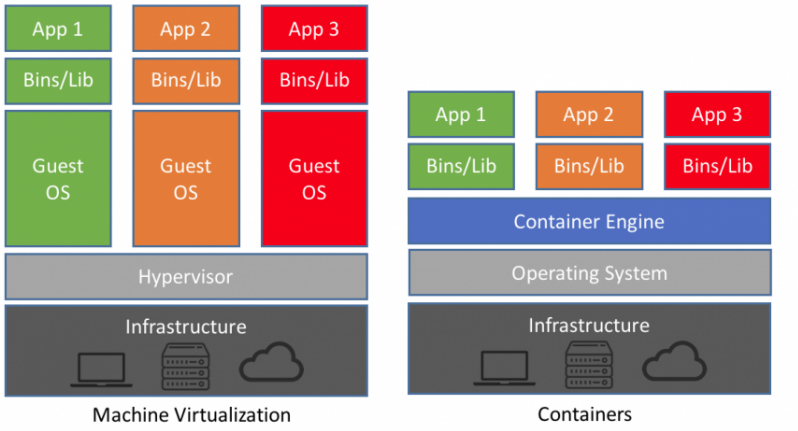

Figure 4.2: Virtual Machines vs Containers

- `/users/adria/mlafages` (in the following it will be referenced as "∼"): this directory contains all input sources.

  It contains the following directories:

  - `afdivider`: it contains all *AFDivider* repository and other operating files (See Section 4.2.1).
  - `dataframes`: it contains dataframes representing the condensed data of experiments. Those dataframes are produced by the ∼`/afdivider/datascript/runExtraction.sh` script.
  - `env`: it contains python3 environment.
  - `solvers_iccma2019`: it contains all the other solvers used in the experiments, embedded into singularity containers.

- `/projets/adria/afdivider/benchmark`: this directory contains the data produced by the experimentation (statistics and extension semantics). Each of its sub-directories contains the experiments of a given solver and is named after that solver (See Section 4.2.3 on page 23 for more details).

4.2.1 *AFDivider* Directory

The project sources are located in ∼`/afdivider`. It is actually the local git repository. In addition to the architecture described in Section 3.1 on page 14 are two directories that are ignored by git:

- ∼`/afdivider/out`: it contains three directories

  - `slurm`: it contains all slurm job log files. The name of the file corresponds to the job ID. This ID is useful to keep track of your running experiments (See Section 4.3.9 on page 40).
  - `solvers`: it contains the different solvers used in our experiments. There, the solver programs are not embedded into singularity containers, contrary to those in ∼`/solvers_iccma2019`.
  - `tmpData`: it contains all the temporary induced AFs generated to compute the semantics. This directory should be regularly emptied.

- ∼/afdivider/runs: it contains log files that indicate succinctly the information related to an experiment or an extraction job launch. The detailed log file made during those processes are stocked into ∼/afdivider/out/slurm.

  Following, the ∼/afdivider/datascript/runExtraction.sh script (see Section 3.1 on page 14 for more details) the log file of an extraction job is stored at ∼/afdivider/runs/extraction. Figure 4.3 is an example of extraction log file. It simply specifies the job ID of the extraction. Then this ID can be used to monitor the job. Notice that this file is erased at each extraction, unless the ∼/afdivider/datascript/runExtraction.sh script is modified.

```
1 Submitted batch job 6618839
```

Figure 4.3: Extraction

Following, the scripts ∼/afdivider/slurm/runAll, that launches the experiments for the other solvers, and ∼/afdivider/slurm/runAllAFDiv, that launches the experiments for *AFDivider* solver, the log files are stored in that directory. Figure 4.4 gives an example of one of those files. It indicates the Osirim compute partition type on which the experiments have been launched (Partition: 24CPUNodes), the number of CPU cores used (NbCPU: 6), the maximal number of simultaneous experiments (MaxParallel: 4), the number of experiment launches to do, which is the size of the job array (NbTasks: 24), the timeout of each task (Timeout per tasks: 60m) and to finish the ID of the first task of the job array. In this example, we use at most 24 cores and so, the equivalent of one full CPU of the 24CPUNodes partition.

Note: The names of those log files can be changed if needed by modifying ∼/afdivider/slurm/runAll and ∼/afdivider/slurm/runAllAFDiv scripts. Changing those names can be useful in order to keep track of all experiment settings without erasing them.

```
1 Partition: 24CPUNodes
2 NbCPU: 6
3 MaxParallel: 4
4 NbTasks: 24
5 Timeout per tasks: 60m
6 Submitted batch job 6330565
```

Figure 4.4: Experiment log file

4.2.2 Solver Containers

So far, in ∼/solvers_iccma2019 are stored the singularity containers of the following solvers:

- coquiaas_iccma2019.sif
- pyglaf_iccma2019.sif
- argpref_iccma2019.sif
- eqargsolver_iccma2019.sif
- taas-dredd_iccma2019.sif

```
[mlafages@co2-slurm-client1 ~]$ cd /projets/adria/afdivider/benchmark/
[mlafages@co2-slurm-client1 benchmark]$ ls
AFDiv              aspartix_iccma2019.sif     main.py                 taas-dredd_iccma2019.sif
ArgSemSAT          cegartix-run.sh            mu-toksia_iccma2019.sif yonas_iccma2019.sif
argmat-dvisat.out  coquiaas_iccma2019.sif     pyglaf.py
argmat-sat.out     eqargsolver_iccma2019.sif  pyglaf_iccma2019.sif
```

Figure 4.5: Data repository

- `aspartix_iccma2019.sif`
- `mu-toksia_iccma2019.sif`
- `yonas_iccma2019.sif`

See [11], Section 5.3.1 “Success Count Comparison” for details and references.
Those solver containers are used as follows:

```
1  $ singularity run $solver $to -p $problem -fo $format -f $file
```

The value of the `$solver` variable can be as an example: `~/solvers_iccma2019/coquiaas_iccma2019.sif`. `$to` is the timeout in seconds. The argument `-p` specifies the problem type, `-fo` the format of the input AF (whether APX or TGF), `-f` the path to the AF.

4.2.3 Data Repository

4.2.3.1 Structure

The directory `/projets/adria/afdivider/benchmark/` contains all experiments statistics and data. As shown by Figure 4.5, each of its sub-directories contains the data of some solver.

As shown by Figure 4.6 on the next page, AFDiv directory contains all the experiments of *AFDivider*. Each first depth directory corresponds to experiments made using a given external solver. Each second depth directory corresponds to experiments made for a given output mode: verbose (which here is called enum), compact or both output mode (See Section 1.3.2 on page 5 `--representation` option). Each third depth directory corresponds to experiments made for different semantics. It is either the *complete*, or the *preferred* or the *stable* semantics. To finish, each fourth depth directory corresponds to experiments made for a given clustering method: spectral, uscc-chain or uscc-tree.

Notice that each statistics file name is built following this pattern:

`<AF file name>-<clustering method>-<output mode>-<deviation rate>Rand.<slurm job ID>.stat`

Inside `AFDiv/aspartix2019/enum/AFDiv-Asp-spectral/EE-PR`, we have as an example the following statistic file:

`BA_100_60_2.apx-spectral-verbose-0.0Rand.6617391.stat`

The data repository of each other solver is as represented by Figure 4.7 on the next page. Notice that each statistics file name is built following this pattern:

`<AF file name>.<slurm job ID>.stat`

As an example we have the directory `/projets/adria/afdivider/benchmark/cegartix-run.sh/EE-PR` following statistic file: `ttc_20151217_1257.gml.80.apx.6340218.stat`.

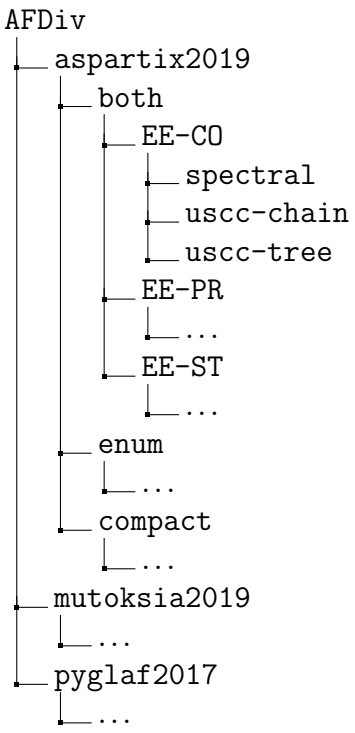

Figure 4.6: AFDiv directory tree view

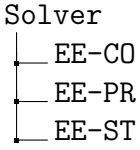

Figure 4.7: Other solver directory tree view

```
Job id: 6541002
Nb cpu: 6
Solver name: AFDivider
Semantic: EE−PR
AF name: BA_160_20_4.apx
Timeout: 3600
Clustering mode: spectral
Clustering randomness: 0.0
Output mode: no−enumeration
[2020−05−14 17:31:45,302] INFO: Nb workers: 6
[2020−05−14 17:31:45,448] INFO: Cutting and clustering time: 0.135596 (CPU), 0.169449 (Real)
[2020−05−14 17:31:45,448] INFO: Clustering partition:
    FixPart has 1 cluster(s):
        ['a1', ..., 'a99']
    Component 1 has 1 cluster(s):
        ['a44', 'a56', 'a62', 'a76']

    ...

    Component 14 has 1 cluster(s):
        ['a155', 'a160']
[2020−05−14 17:31:45,625] INFO: Clusters labelling time: 0.093569 (CPU), 0.177065 (Real)
[2020−05−14 17:31:45,626] INFO: Nb extensions to construct: 24576
[2020−05−14 17:31:45,626] INFO: Enumeration construction time: 0.000813 (CPU), 0.000781 (Real)
[2020−05−14 17:31:45,626] INFO: Printing time: 0.000229 (CPU), 0.000242 (Real)
[2020−05−14 17:31:45,626] INFO: Total time: 0.230207 (CPU), 0.347537 (Real)
{{[a116,a150],[a73]},{[a103],[a66]},{[a124],[a35]},{[a36,a43,a95,a53],[a49,a36,a95,a53],[a15,a49]},{[a16],[a111,a59
    ]},{[a29,a83],[a0,a45]},{[a9],[a148]},{[a155],[a160]},{[a144],[a65]},{[a56,a76],[a44,a62]},{[a33],[a128]},{[a63
    ],[a135]},{[a120,a126],[a118]},{[a17,a90],[a30,a145,a12]},{[a151,a97,a68,a106,a139,a159,a131,a69,a119,a127,
    a115,a107,a21,a113,a75,a91,a4,a143,a146,a98,a104,a100,a18,a101,a130,a80,a46,a74,a147,a48,a84,a47,a20,a82,
    a86,a121,a132,a92,a96,a142,a38,a34,a122,a55,a158,a14,a99,a93,a61,a57,a87,a50,a152,a136,a42,a154,a78,a133,
    a138,a137,a26,a141,a125,a51,a108,a32,a114,a129,a81,a39,a11,a25,a64,a54,a28,a110,a156,a22,a24,a71,a109,a7,
    a157]}}


0.865u 0.676s 0:01.30 117.6% 0+0k 16+48io 0pf+0w
```

Figure 4.8: *AFDivider* data file example

```
Job id: 6444052
Nb cpu: 6
Solver name: aspartix_iccma2019.sif
Problem: EE−PR
AF name: BA_160_20_4.apx
Timeout: 3600
Format: apx

[ [a116, ...] ... ]

real 0m 2.51s
user 0m 3.52s
sys 0m 0.34s
```

Figure 4.9: Other solver data file example

4.2.3.2 Data File

The data file produced by *AFDivider* contains three types of information:

- The experiment parameters
- The statistics
- The computed semantics

Figure 4.8 on the previous page shows an example of data file with the compact enumeration representation as output format. We can observe that it is basically a statistics file combined with a storage file, as presented in Section 1.3.2 on page 5, with some more information. As header, we have the experiment parameters: job ID, number of cores, solver name, semantics, AF name, timeout (in second), clustering mode, deviation rate and output format. The last line of the file is the output of the GNU time command.

`Note:` *There is a time gap between the real time given by the GNU time command and the real time given by* AFDivider *program itself. This time gap has not yet been identified. It is probably due to the fact that the program does not not kill all threads immediately after the computation of the semantics.*

The data file produced by other solvers are as the example given in Figure 4.9 on the previous page. Notice that the output of the time command is different from the previous example. It is because at that time, the shell installed on Osirim and thus the time command were not the same.

4.3 Experiment Process Manual

In this section are presented the different processes to properly conduct experiments.

4.3.1 Overview

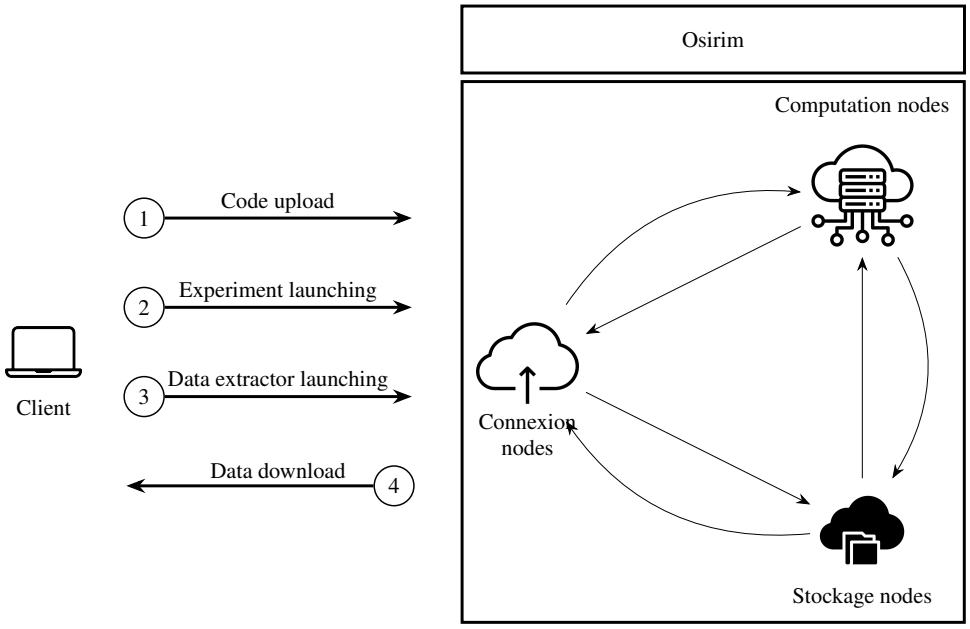

Figure 4.10: Experiment operating diagram

First of all, let consider the diagram illustrated in Figure 4.10 on the previous page. It gives, with some simplifications, the main steps to conduct experiments:

1. Code uploading: the git repository has to be updated from the remote repository in order to have the latest version of *AFDivider* and others operating files.

2. Experiment launching: Once the files updated, experiments are launch on Osirim compute cluster with slurm jobs.

3. Data extractor launching: When the experiment jobs are finished, an extraction job has to be launched in order to process the data and to produce a dataframe with all experiment results.

4. Data download: To finish the produced dataframe is download on the client computer to be analysed. From it, charts and high level statistics are produced.

In the following subsections each of these steps are detailed.

4.3.2 Connection to Osirim

In order to connect to Osirim, you need to install the *ssh* client on your computer. Install also the *scp* command that will be useful to download and upload files that are not in the git repository.

In your terminal, enter the following command:

```
1  $ ssh <your osirim account>@osirim-slurm.irit.fr
```

Then enter your password.

Once connected, you can explore the two repositories where *AFDivider* project files are stocked: `/users/adria/mlafages` and `/projets/adria/afdivider/`.

4.3.3 Project Refresh

After your *commit* and *push* from your client, to refresh your project on Osirim, connect and enter the following command:

```
1  $ cd /users/adria/mlafages/afdivider
2  $ git pull
```

4.3.4 Slurm Job

Before presenting the various script and configuration files used in the experiment process, let introduce some basic notions of slurm.

A slurm job is simply a computer process, that is a script, a program to be executed. To launch slurm jobs, we use the commands `srun` or `sbatch`. The difference between the two is that `srun` is interactive and blocking (you get the result in your terminal and you cannot write other commands until it is finished), while sbatch is batch processing and non-blocking (results are written to a file and you can submit other commands right away).

Both have same options except that sbatch admits *"job arrays"* or *"task arrays"*. A slurm task array is a collection of jobs that differ from each other by only a single index parameter. Creating a task array provides an easy way to group related jobs together. For example, if you have a parameter study that

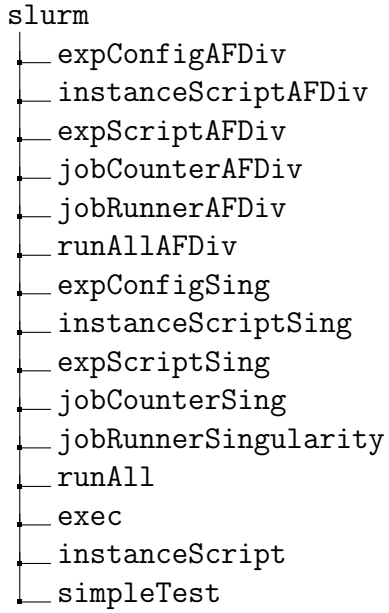

Figure 4.11: Tree view of `slurm` folder

requires you to run your application five times, each with a different input parameter, you can use a task array instead of creating five separate slurm scripts and submitting them separately.

As an example, the following command line will launch 1000 times the `testarray.sh` with a maximum of 5 tasks in parallel:

```
1 $ sbatch --array [1-1000]%5 testarray.sh
```

Both `srun` and `sbatch` commands have various options. They are all presented in slurm documentation available here: `https://slurm.schedmd.com/`.

They can be specified:

- As arguments of the `srun` and `sbatch` commands. See the script presented in Figure 4.17 on page 32 as an example.
- With a script header. See the script presented in Figure 4.15 on page 31 as an example.

4.3.5 Experiment Configuration and Scripts

The configuration and script files for the experiments are stored in the folder: `~/afdivider/slurm`. Figure 4.11 lists its content.

Let explain the utility of each file:

- The `exec` script, shown in Figure 4.12 on the following page, allows to wrap a command to execute inside the `timeout` command. It limits the execution of the command given in parameter to the timeout specified as the first parameter. A priori there is no need to modify this file.
- The `expConfigAFDiv` file specifies the type of experiments to run for *AFDivider* solver. Figure 4.13 on the following page shows an example of configurations. Each line represents a configuration with the following information, separated by a white space:
  1. Problem type

```
1 #!/bin/tcsh
2 timeout $argv:q
3 echo
```

Figure 4.12: The `exec` script

```
1 EE−PR spectral compact instances/AF/hardInstances/ 0.0 out/solvers/aspartix19/aspartix−V−interface−2019.sh
2 EE−PR uscc_chain compact instances/AF/hardInstances/ 0.0 out/solvers/aspartix19/aspartix−V−interface−2019.sh
3 EE−PR uscc_tree compact instances/AF/hardInstances/ 0.0 out/solvers/aspartix19/aspartix−V−interface−2019.sh
```

Figure 4.13: `expConfigAFDiv` file example

2. Clustering method to used
3. Output format
4. Folder in which the AF instances to solved are located (all will be solved)
5. Clustering randomness to apply
6. Internal solver to used at cluster level

- The `instanceScriptAFDiv` file, shown is Figure 4.14 on the next page, is the script that launches a task to solve one instance, using the `exec` script. It prints some information that will be stored in the statistics file, before the experiment launch.

  A priori there is no need to modify this file. The variable parameters for *AFDivider* experiment are specified in `expConfigAFDiv` and `runAllAFDiv` files and in `expScriptAFDiv` script header.

- The `expScriptAFDiv` script, shown in Figure 4.15 on page 31, constructs the shell command lines corresponding to the experiment configurations specified in `expConfigAFDiv`. For each configuration and for AF instance, it:

  – Creates the directory to store the statistics file produced (if not existing)
  Notice that the experiment statistics files are stored in `/projets/adria/afdivider/benchmark/rawExpeAFDiv/`. Afterward the checked statistics files should be placed manually in the `/projets/adria/afdivider/benchmark/AFDiv/` directory.

  – Specifies the statistics file name according to the experiment configuration

  – Prepares the shell command line for the resolution of the AF instance

  Each shell command line created is stored in an array `$commands`. Then, the slurm batch called in `jobRunnerAFDiv` will execute each command using the current task ID `$SLURM_ARRAY_JOB_ID`.

  `Note:` A timeout is specified for the slurm task with the `--time` parameter (line 44) however, experiments shew that it was not reliable. That is why we wrap the different tasks into the `slurm/exec` script.

  A priori there is no need to modify `expScriptAFDiv` script, **except** to change its header. The lines 2-5 are slurm batch parameters:

  – "`#SBATCH --mail-type=END`" indicates that a mail will be send when the slurm batch will be fully executed

```
#!/bin/tcsh
set solver = "AFDivider"
set semantic = "$1"
set file = "$2"
set clusteringMode = "$3"
set clusteringRandomness = "$4"
set nbcpu = "$5"
set to = "$6"
set outputMode = "$7"
set internalSolver = "$8"

echo "Job id:" "$SLURM_JOB_ID"
echo "Nb cpu:" "$nbcpu"
echo "Solver name:" "$solver:t"
echo "Semantic:" "$semantic"
echo "AF name:" "$file:t"
echo "Timeout:" "$to"
echo "Clustering mode:" "$clusteringMode"
echo "Clustering randomness:" "$clusteringRandomness"
echo "Output mode:" "$outputMode"
echo "Solver used at cluster level:" "$internalSolver"

time /users/adria/mlafages/afdivider/slurm/exec "$to" /logiciels/Python-3.5.2/bin/python3 /users/adria/mlafages/
    afdivider/main.py -s "$internalSolver" -p "$semantic" -fo apx -f "$file" --clustering-mode "$clusteringMode"
    --clustering-randomness "$clusteringRandomness" --representation "$outputMode"
```

Figure 4.14: The `instanceScriptAFDiv` script

  - "`#SBATCH --mail-user=mlafages@irit.fr`" specifies to which address to send the mail
  - "`#SBATCH --output=/users/adria/mlafages/afdivider/out/slurm/%A.out`" specifies the output log file corresponding to the slurm batch. The `%A` will be replaced by the job ID, which is also the ID of the first task of the task array.
  - "`#SBATCH --mem-per-cpu=7500M`" specifies the RAM allocated to each task per CPU used. As an example if 6 cores are allocated to a task and that for each cpu 7500Mo are allocated, then the task will have access to at most 45 Go of RAM.

- The `jobCounterAFDiv` script, shown in Figure 4.16 on page 32, counts, following the experiment configurations specified in `expConfigAFDiv`, the number of tasks to do, that is the total number of instances to solve. The computed number is used in the `runAllAFDiv` script then in the `jobRunnerAFDiv` script. A priori there is no need to modify this file.

- The `jobRunnerAFDiv` script, shown in Figure 4.17 on page 32, displays the slurm parameters for the job (which are received in parameters by the `runAllAFDiv` script):

  - The slurm partition on which the batch has to be launched
  - The number of CPU per task
  - The maximal number of tasks in parallel
  - The number of tasks
  - The timeout for each task

```
#!/bin/tcsh
#SBATCH --mail-type=END
#SBATCH --mail-user=mlafages@irit.fr
#SBATCH --output=/users/adria/mlafages/afdivider/out/slurm/%A.out # STDOUT
#SBATCH --mem-per-cpu=7500M

set nbcpu = $1
set to = $2
@ srunTo = $to / 60 + 1

set firstJobId = "$SLURM_ARRAY_JOB_ID"
echo "First job id" $firstJobId

set commands = ()

foreach config ("`cat /users/adria/mlafages/afdivider/slurm/expConfigAFDiv`")
    set config = ($config)
    set semantic = "$config[1]"
    set clusteringMode = "$config[2]"
    set outputMode = "$config[3]"
    set instanceFolder = "$config[4]"
    set clusteringRandomness = "$config[5]"
    set internalSolver = "$config[6]"

    echo $semantic $clusteringMode $outputMode $clusteringRandomness

    foreach file (`ls -d $instanceFolder/*`)
        set dir = /projets/adria/afdivider/benchmark/rawExpeAFDiv/"$internalSolver:t"/$semantic
        if (! -d $dir) then
            mkdir -p $dir
        endif

        set outFilename = $dir/$file:t-"$clusteringMode"-"$outputMode"-"$clusteringRandomness"Rand."
            $firstJobId".stat

        echo "OutFilename:" "$outFilename"

        set currentCommand = "$outFilename /users/adria/mlafages/afdivider/slurm/instanceScriptAFDiv $semantic
            $file $clusteringMode $clusteringRandomness $nbcpu $to $outputMode $internalSolver"
        set commands = ($commands:q "$currentCommand")
    end
end

set currentCommand = ($commands[$SLURM_ARRAY_TASK_ID])

srun --time=$srunTo -o $currentCommand:q
```

Figure 4.15: The `instanceScriptAFDiv` script

```
#!/bin/tcsh

set nbTask = 0
foreach config (”‘cat /users/adria/mlafages/afdivider/slurm/expConfigAFDiv‘”)
    set firstChar = ‘echo $config | cut −c1‘
    if ( ”$firstChar” == ”#” ) then
        continue
    endif
    set config = ($config)
    set fileFolder = ”$config[4]”

    set nbFile = ‘ls $fileFolder | wc −l‘
    @ nbTask = ($nbTask + $nbFile)
end

echo $nbTask
```

Figure 4.16: The `jobCounterAFDiv` script

```
#!/bin/tcsh

set partition = $1
set nbcpu = $2
set maxParallel = $3
set nbTasks = $4
set to = $5

echo ”Partition:” ”$partition”
echo ”NbCPU:” ”$nbcpu”
echo ”MaxParallel:” ”$maxParallel”
echo ”NbTasks:” ”$nbTasks”
echo ”Timeout:” ”$to sec”

sbatch −−partition=”$partition” −−cpus−per−task=”$nbcpu” −−array=1−”$nbTasks”%”$maxParallel” /users/adria/
    mlafages/afdivider/slurm/expScriptAFDiv ”$nbcpu” ”$to”
```

Figure 4.17: The `jobRunnerAFDiv` script

Then it launches the slurm batch to execute all the slurm task array specified in the `expScriptAFDiv` script. A priori there is no need to modify this file.

- The `runAllAFDiv` script, dispayed in Figure 4.18 on the following page, is the most high level script to launch all the mechanism. It calls the `jobRunnerAFDiv` script with the required parameters (slurm partition, number of CPU per task, maximal number of tasks in parallel, number of tasks, timeout for each task) and saves the log of the batch launch in `/users/adria/mlafages/afdivider/runs/runAFDiv.log`.

  This file has to be modified if needed to change those parameters and the location of the log file.

All the previous scripts and configuration files have their counterparts for the experiments of other solvers:

- The `expConfigSing` file specifies the type of experiments to run for other solvers. Figure 4.19

```
1 /users/adria/mlafages/afdivider/slurm/jobRunnerAFDiv 24CPUNodes 6 4 '/users/adria/mlafages/afdivider/slurm/
      jobCounterAFDiv' 3600 > /users/adria/mlafages/afdivider/runs/runAFDiv.log
```

Figure 4.18: The `runAllAFDiv` script

```
1 /users/adria/mlafages/solvers_iccma2019/aspartix_iccma2019.sif EE-PR /users/adria/mlafages/afdivider/instances/AF/
      hardInstances/ apx
2 /users/adria/mlafages/solvers_iccma2019/mu-toksia_iccma2019.sif EE-PR /users/adria/mlafages/afdivider/instances/AF
      /hardInstances/ apx
```

Figure 4.19: The `expConfigSing` file

shows an example of configurations. Each line represents a configuration with the following information, separated by a white space:

1. Path to Solver
2. Problem type
3. Folder in which the AF instances to solved are located (all will be solved)
4. Input format

- The `instanceScriptSing` file, shown in Figure 4.20 on the next page, is the script that launches a task to solve one instance, using the `exec` script. It prints some information that will be stored in the statistics file, before the experiment launch.

- The `expScriptSing` script, shown in Figure 4.21 on page 35, constructs the shell command lines corresponding to the experiment configurations specified in `expConfigSing`. For each configuration and for AF instance, it:

  - Verifies if the configuration is valid (that is, that the solver and instance folder exist)
  - Creates the directory to store the resulting statistics file (if not existing)
    Notice that the experiment statistics files are stored in `/projets/adria/afdivider/benchmark/<solverName>`.
  - Specifies the statistics file name according to the experiment configuration
  - Prepares the shell command line for the resolution of the AF instance

  Each shell command line created is stored in an array `$commands`. Then, the slurm batch called in `jobRunnerSingularity` will execute each command using the current task ID stored in `$SLURM_ARRAY_JOB_ID`.

  `Note:` A timeout is specified for the slurm task with the `--time` parameter (line 44) however, experiments shew that it was not reliable. That is why we wrap the different tasks into the `slurm/exec` script.

  A priori there is no need to modify the `expScriptSing` script, **except** to change its header. The lines 2-5 are slurm batch parameters:

```
1  #!/bin/tcsh
2  set solver = "$1"
3  set problem = "$2"
4  set file = "$3"
5  set nbcpu = "$4"
6  set to = "$5"
7  set format = "$6"
8
9  echo "Job id:" "$SLURM_JOB_ID"
10 echo "Nb cpu:" "$nbcpu"
11 echo "Solver name:" "$solver:t"
12 echo "Problem:" "$problem"
13 echo "AF name:" "$file:t"
14 echo "Timeout:" "$to"
15 echo "Format:" "$format"
16
17 singularity run $solver $to -p $problem -fo $format -f $file
```

Figure 4.20: The `instanceScriptSing` script

- "`#SBATCH --mail-type=END`" indicates that a mail will be send when the slurm batch will be fully executed
- "`#SBATCH --mail-user=mlafages@irit.fr`" specifies to which address to send the mail
- "`#SBATCH --output=/users/adria/mlafages/afdivider/out/slurm/%A.out`" specifies the output log file corresponding to the slurm batch. The `%A` will be replaced by the job ID, which is also the ID of the first task of the task array.
- "`#SBATCH --mem-per-cpu=7500M`" specifies the RAM allocated to each task per CPU used. As an example if 6 cores are allocated to a task and that for each cpu 7500Mo are allocated, then the task will have access to at most 45 Go of RAM.

- The `jobCounterSing` script, shown in Figure 4.22 on page 36, counts, following the experiment configurations specified in `expConfigSing`, the number of tasks to do, that is the total number of instances to solve. The computed number is used in the `runAll` script then in the `jobRunnerSingularity` script. A priori there is no need to modify this file.

- The `jobRunnerSingularity` script, shown in Figure 4.23 on page 36, displays the slurm parameters for the job (which are received in parameters by the `runAll` script):
  - The slurm partition on which the batch has to be launched
  - The number of CPU per task
  - The maximal number of tasks in parallel
  - The number of tasks
  - The timeout for each task

  Then it launches the slurm batch to execute all the slurm task array specified in the `expScriptSing` script. A priori there is no need to modify this file.

```
#!/bin/tcsh
#SBATCH --mail-type=END
#SBATCH --mail-user=mlafages@irit.fr
#SBATCH --output=/users/adria/mlafages/afdivider/out/slurm/%A.out # STDOUT
#SBATCH --mem-per-cpu=7500M

set nbcpu = $1
set to = $2
@ srunTo = $to / 60 + 1

set firstJobId = "$SLURM_ARRAY_JOB_ID"
echo "First job id" $firstJobId

set commands = ()

    ... # Configuration verification

foreach config ("`cat /users/adria/mlafages/afdivider/slurm/expConfigSing`")
    set firstChar = `echo $config | cut -c1`
    if ( "$firstChar" == "#" ) then
        continue
    endif
    set config = ($config)
    set solver = "$config[1]"
    set problem = "$config[2]"
    set fileFolder = "$config[3]"
    set format = "$config[4]"

    foreach file (`ls -d $fileFolder/*.$format`)
        set dir = /projets/adria/afdivider/benchmark/$solver:t/$problem
        if (! -d $dir) then
            mkdir -p $dir
        endif

        set outFilename = $dir/$file:t."$firstJobId".stat

        echo "OutFilename:" "$outFilename"

        set isNotSif = `echo $solver | grep '.sif$'`
        if ($isNotSif == "") then
            set currentCommand = "$outFilename /users/adria/mlafages/afdivider/slurm/instanceScript $solver
                $problem $file $nbcpu $to $format"
        else
            set currentCommand = "$outFilename /users/adria/mlafages/afdivider/slurm/instanceScriptSing $solver
                $problem $file $nbcpu $to $format"
        endif

        set commands = ($commands:q "$currentCommand")
    end
end

set currentCommand = ($commands[$SLURM_ARRAY_TASK_ID])
srun --time=$srunTo -o $currentCommand:q
```

Figure 4.21: The `expScriptSing` script

```
#!/bin/tcsh

set nbTask = 0
foreach config ("`cat /users/adria/mlafages/afdivider/slurm/expConfigSing`")
    set firstChar = `echo $config | cut -c1`
    if ( "$firstChar" == "#" ) then
        continue
    endif
    set config = ($config)
    set fileFolder = "$config[3]"

    set nbFile = `ls $fileFolder | wc -l`
    @ nbTask = ($nbTask + $nbFile)
end

echo $nbTask
```

Figure 4.22: The `jobCounterSing` script

```
#!/bin/tcsh

set partition = $1
set nbcpu = $2
set maxParallel = $3
set nbTasks = $4
set to = $5

echo "Partition:" "$partition"
echo "NbCPU:" "$nbcpu"
echo "MaxParallel:" "$maxParallel"
echo "NbTasks:" "$nbTasks"
echo "Timeout:" "$to sec"

sbatch --partition="$partition" --cpus-per-task="$nbcpu" --array=1-"$nbTasks"%"$maxParallel" /users/adria/
    mlafages/afdivider/slurm/expScriptSing $nbcpu $to
```

Figure 4.23: The `jobRunnerSingularity` script

```
/users/adria/mlafages/afdivider/slurm/jobRunnerSingularity 24CPUNodes 6 4 '/users/adria/mlafages/afdivider/slurm/
    jobCounterSing' 3600 > /users/adria/mlafages/afdivider/runs/runSing.log
```

Figure 4.24: The `runAll` script

```
#!/bin/tcsh
set solver = "$1"
set semantic = "$2"
set file = "$3"
set nbcpu = "$4"
set to = "$5"
set format = "$6"

echo "Job id:" "$SLURM_JOB_ID"
echo "Nb cpu:" "$nbcpu"
echo "Solver name:" "$solver:t"
echo "Semantic:" "$semantic"
echo "AF name:" "$file:t"
echo "Timeout:" "$to"
echo "Format:" "$format"

time /users/adria/mlafages/afdivider/slurm/exec $to $solver -p $semantic -fo $format -f $file
```

Figure 4.25: The `instanceScript` script

- The `runAll` script, dispayed in Figure 4.24, is the most high level script to launch all the mechanism. It calls the `jobRunnerSingularity` script with the required parameters (slurm partition, number of CPU per task, maximal number of tasks in parallel, number of tasks, timeout for each task) and saves the log of the batch launch in `/users/adria/mlafages/afdivider/runs/runSing.log`.

  This file has to be modified if needed to change those parameters and the location of the log file.

Finally, two more scripts exist in order to make simple tests:

- The `instanceScript` script, shown in Figure 4.25, launches a task to solve one instance. It prints some information before the experiment launch.
- The `simpleTest` script, shown in Figure 4.26 on the next page, creates a simple slurm task while calling the `instanceScript` script. It takes two parameters: the AF instance to solve and the solver to use.

4.3.6 Experiment Launching

Following Section 4.3.5 on page 28, here are the following procedures to launch experiments for *AFDivider* or other solvers.

4.3.6.1 *AFDivider*

For *AFDivider* here are the following steps. Notice that some steps may be non necessary:

1. Connect to Osirim

```
#!/bin/tcsh
#SBATCH --mail-type=END
#SBATCH --mail-user=mlafages@irit.fr
#SBATCH --output=/users/adria/mlafages/afdivider/out/slurm/%A.out # STDOUT
#SBATCH --mem-per-cpu=7500M

set file = $2
set solver = $1
srun /users/adria/mlafages/afdivider/slurm/instanceScript $solver EE-PR $file
```

Figure 4.26: The `simpleTest` script

2. Refresh your project
3. Modify the `expScriptAFDiv` script to set the wanted mail address to be notified at the end of the experiment
4. Modify the `expScriptAFDiv` script to set the wanted amount of memory allocated per CPU
5. Set your experiment configurations in the `expConfigAFDiv`
6. Modify the `runAllAFDiv` script to your convenience
7. Execute the `runAllAFDiv` script as follows:

```
$ /users/adria/mlafages/afdivider/slurm/runAllAFDiv
```

4.3.6.2 Other solvers

For other solvers, here are the following steps. Notice that some steps may be non necessary:

1. Connect to Osirim
2. Refresh your project
3. Modify the `expScriptSing` script to set the wanted mail address to be notified at the end of the experiment
4. Modify the `expScriptSing` script to set the wanted amount of memory allocated per CPU
5. Set your experiment configurations in the `expConfigSing`
6. Modify the `runAll` script to your convenience
7. Execute the `runAll` script as follows:

```
$ /users/adria/mlafages/afdivider/slurm/runAll
```

4.3.7 Data Extraction Launching

To launch a data extraction, do the following steps:

1. Connect to osirim

2. Edit the `extractor.sh` file to specify:

   - The directory path where solver's data are stored, with the option `--dirPath`.

     Notice that the path should be the one in which the sub-directories are the solver data archive. Practically, this path is either `/projets/adria/afdivider/benchmark` for other solvers or a path for *AFDivider* following this pattern:

     ```
     /projets/adria/afdivider/benchmark/AFDiv/<internalSolver>/<outputMode>
     ```

     As an example:

     ```
     /projets/adria/afdivider/benchmark/AFDiv/aspartix2019/enum
     ```

     In the case all sub-directories are not to be processed, the option `--retrieval-selection` must be specified.

     `Note:` Inside a solver directory, only three sub-directories will be processed: `EE-PR`, `EE-ST`, `EE-CO`.

   - The solver selection with the option `--retrieval-selection`, as the following example shows it:

     ```
     srun /users/adria/mlafages/afdivider/dataScript/extractor.py --mode retrieval --dirPath /projets/adria/afdivider/benchmark --output-dataframePath /users/adria/mlafages/dataframeTest.csv --retrieval-selection aspartix_iccma2019.sif coquiaas_iccma2019.sif
     ```

   - The output dataframe file, with the option `--output-dataframePath`.

3. Edit if needed the `runExtraction.sh` script to specify the extration log file. If not modified, the log file will be `/users/adria/mlafages/afdivider/runs/extraction`.

4. Run the `runExtraction.sh` script:

   ```
   $ /users/adria/mlafages/afdivider/dataScript/runExtraction.sh
   ```

4.3.8 Data Retrieval

Finally to retrieve the dataframe created to your client. Do the following command from your client terminal:

```
$ scp <your osirim account>@osirim-slurm.irit.fr:/users/adria/mlafages/dataframeTest.csv .
```

This will copy the dataframe to your current directory.

4.3.9 Slurm Monitoring

To monitor slurm jobs you can connect to `https://osirim.irit.fr/etat-de-la-plateforme/` with your osirim account credentials. From there you can:

- See the cluster workload of each compute nodes of Osirim (see Section 4.1.2 on page 17).
- For each node type, see the running jobs and the job queue.
- Cancel jobs using their ID.

To facilitate the job identification, you can go to ∼`/afdivider/out/slurm`. The greatest file ID (the ID is an integer number) is the newest launch. Notice that all slurm log files go there, whether it is for data extraction or experiments jobs. Figure 4.28 on page 42 represents an extraction file log while Figure 4.27 on the following page an experiment one. Let explain each of them.

Consider Figure 4.27 on the following page. The first line indicates the ID of the job or the ID of the first task if the job is a task array (See Section 4.3.4 on page 27 for more details). The second line specifies the task. Here, *AFDivider* solver is called to compute the *Compact Enumeration Representation* of *preferred* semantics using the spectral clustering method. The "0.0" correspond to the wanted random deviation rate from the clustering produced by the spectral clustering (See Section 1.3.2 on page 5, option: `--clustering-randomness`). As a consequence, in this experiment batch no changes are made. The following lines specify the paths to the statistics files produced by the experiments.

Now, consider Figure 4.28 on page 42. The first line indicates the command executed by the job. Here, the data of the statistics/storage file are retrieved from the directory: `/projets/adria/afdivider/benchmark/AFDiv/aspartix2019/enum` and the dataframe created has been stored at this location: `/users /adria/mlafages/dataframeAspartixAFDiv.csv`. The following lines specify the tree view of the explored files. In the case where a bug would happen, this tree view is very useful to troubleshoot, to see from the parsing of which file the problem comes from.

```
1  First job id 6619180
2  EE-PR spectral compact 0.0
3  OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/amador-transit_20151216_1706.
       gml.80.apx-spectral-compact-0.0Rand.6619180.stat
4  OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/auckland-regional-transport-
       authority_20110207_0347.normalized.gml.80.apx-spectral-compact-0.0Rand.6619180.stat
5  OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_100_60_2.apx-spectral-
       compact-0.0Rand.6619180.stat
6  OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_100_80_3.apx-spectral-
       compact-0.0Rand.6619180.stat
7  OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_120_70_1.apx-spectral-
       compact-0.0Rand.6619180.stat
8  OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_120_80_2.apx-spectral-
       compact-0.0Rand.6619180.stat
9  OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_120_90_5.apx-spectral-
       compact-0.0Rand.6619180.stat
10 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_140_60_2.apx-spectral-
       compact-0.0Rand.6619180.stat
11 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_140_90_3.apx-spectral-
       compact-0.0Rand.6619180.stat
12 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_160_20_4.apx-spectral-
       compact-0.0Rand.6619180.stat
13 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_160_50_1.apx-spectral-
       compact-0.0Rand.6619180.stat
14 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_160_60_2.apx-spectral-
       compact-0.0Rand.6619180.stat
15 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_160_60_5.apx-spectral-
       compact-0.0Rand.6619180.stat
16 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_180_40_2.apx-spectral-
       compact-0.0Rand.6619180.stat
17 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_180_60_4.apx-spectral-
       compact-0.0Rand.6619180.stat
18 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_180_70_1.apx-spectral-
       compact-0.0Rand.6619180.stat
19 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/BA_200_70_4.apx-spectral-
       compact-0.0Rand.6619180.stat
20 OutFilename: /projets/adria/afdivider/benchmark/main.py/aspartix2019/EE-PR/basin-or-us.gml.20.apx-spectral-
       compact-0.0Rand.6619180.stat
```

Figure 4.27: Slurm Experiment log file

```
python extractor.py --mode=retrieval --dirPath=/projets/adria/afdivider/benchmark/AFDiv/aspartix2019/enum --
    output-dataframePath /users/adria/mlafages/dataframeAspartixAFDiv.csv
AFDiv-Asp19-UsccTree
        EE-CO
                ER_500_50_1.apx-uscc_tree-verbose-0.0Rand.6617391.stat
                ER_400_30_1.apx-uscc_tree-verbose-0.0Rand.6617391.stat
                auckland_0347.apx-uscc_tree-verbose-0.0Rand.6617391.stat
        EE-ST
                ER_500_50_1.apx-uscc_tree-verbose-0.0Rand.6617391.stat
                ER_400_30_1.apx-uscc_tree-verbose-0.0Rand.6617391.stat
                auckland_0347.apx-uscc_tree-verbose-0.0Rand.6617391.stat
        EE-PR
                ER_500_50_1.apx-uscc_tree-verbose-0.0Rand.6617391.stat
                ER_400_30_1.apx-uscc_tree-verbose-0.0Rand.6617391.stat
                auckland_0347.apx-uscc_tree-verbose-0.0Rand.6617391.stat
AFDiv-Asp19-UsccChain
        EE-CO
                ER_400_30_2.apx-uscc_chain-verbose-0.0Rand.6617391.stat
                WS_400_32_50_10.apx-uscc_chain-verbose-0.0Rand.6617391.stat
                WS_500_16_90_30.apx-uscc_chain-verbose-0.0Rand.6617391.stat
        EE-ST
                ER_400_30_2.apx-uscc_chain-verbose-0.0Rand.6617391.stat
                WS_400_32_50_10.apx-uscc_chain-verbose-0.0Rand.6617391.stat
                WS_500_16_90_30.apx-uscc_chain-verbose-0.0Rand.6617391.stat
        EE-PR
                ER_400_30_2.apx-uscc_chain-verbose-0.0Rand.6617391.stat
                WS_400_32_50_10.apx-uscc_chain-verbose-0.0Rand.6617391.stat
                WS_500_16_90_30.apx-uscc_chain-verbose-0.0Rand.6617391.stat
AFDiv-Asp19-Spectral
        EE-CO
                ER_400_10_5.apx-spectral-verbose-0.0Rand.6617391.stat
                WS_400_16_70_70.apx-spectral-verbose-0.0Rand.6617391.stat
                bw2.pfile-3-07.pddl.1.cnf.apx-spectral-verbose-0.0Rand.6617391.stat
        EE-ST
                ER_400_10_5.apx-spectral-verbose-0.0Rand.6617391.stat
                WS_400_16_70_70.apx-spectral-verbose-0.0Rand.6617391.stat
                bw2.pfile-3-07.pddl.1.cnf.apx-spectral-verbose-0.0Rand.6617391.stat
        EE-PR
                ER_400_10_5.apx-spectral-verbose-0.0Rand.6617391.stat
                WS_400_16_70_70.apx-spectral-verbose-0.0Rand.6617391.stat
                bw2.pfile-3-07.pddl.1.cnf.apx-spectral-verbose-0.0Rand.6617391.stat
```

Figure 4.28: Slurm Experiment log file

Chapter 5

Conclusion and Perspectives

This document provides everything needed to get started with *AFDivider* solver use and improvement. Mickaël Lafages' PhD Thesis [11] opens a bunch of feature-oriented perspectives for the *AFDivider* algorithm. In addition to those, we give here some technical perspectives to enhance *AFDivider* project as a whole:

- Create an installer for *AFDivider* solver and/or make it available as a package.
- Embed *AFDivider* solver into a Singularity container will facilitate its portability. No Osirim support for new package installations will be needed and a simple conversion to Docker container will make it suitable for ICCMA competition.
- Configure Gitlab CI/CD to facilitate *AFDivider* solver tests and deployments, connecting Osirim to Gitlab CI/CD (continuous integration/continuous deployment).
- Make the code documentation available on line, for example with Gitlab Pages (if possible, for continuous integration and deployment).

Bibliography

[1] Gianvincenzo Alfano, Sergio Greco, and Francesco Parisi. Efficient computation of extensions for dynamic abstract argumentation frameworks: An incremental approach. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 49–55, 2017.

[2] Mario Alviano. The pyglaf argumentation reasoner. In *OASIcs-OpenAccess Series in Informatics*, volume 58. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[3] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011.

[4] Martin Caminada. On the issue of reinstatement in argumentation. In *JELIA*, pages 111–123, 2006.

[5] Federico Cerutti, Ilias Tachmazidis, Mauro Vallati, Sotirios Batsakis, Massimiliano Giacomin, and Grigoris Antoniou. Exploiting parallelism for hard problems in abstract argumentation. In *AAAI*, pages 1475–1481, 2015.

[6] Günther Charwat, Wolfgang Dvořák, Sarah A Gaggl, Johannes P Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation–a survey. *Artificial intelligence*, 220:28–63, 2015.

[7] OHAAI Collaboration, Federico Castagna, Timotheus Kampik, Atefeh Keshavarzi Zafarghandi, Mickaël Lafages, Jack Mumford, Christos T. Rodosthenous, Samy Sá, Stefan Sarkadi, Joseph Singleton, Kenneth Skiba, and Andreas Xydis. Online handbook of argumentation for ai: Volume 1, 2020.

[8] S. Doutre, M. Lafages, and M-C. Lagasquie-Schiex. A distributed and clustering-based algorithm for the enumeration problem in abstract argumentation. In *Proc. of PRIMA*, pages 87–105. Springer, 2019.

[9] Sylvie Doutre, Mickaël Lafages, and Marie-Christine Lagasquie-Schiex. A Distributed and Clustering-based Algorithm for the Enumeration Problem in Abstract Argumentation (JIAF 2020). In *14èmes Journées d'Intelligence Artificielle Fondamentale (JIAF 2020)*, Actes des JIAF 2020, pages 99–108, Angers, France, 2020. AFIA.

[10] Wolfgang Dvorak and Paul E. Dunne. Computational problems in formal argumentation and their complexity. In *Handbook of formal argumentation*, pages 631–688. College publication, 2018.

[11] Mickaël LAFAGES. *Algorithms for Enriched Abstract Argumentation Frameworks for Large-scale Cases*. PhD thesis, Université de Toulouse, 2022.

[12] Mickaël Lafages, Sylvie Doutre, and Marie-Christine Lagasquie-Schiex. Clustering and distributed computing in abstract argumentation. Rapport de recherche IRIT/RR–2018–11–FR, IRIT, Université Paul Sabatier, Toulouse, décembre 2018.

[13] Beishui Liao. Toward incremental computation of argumentation semantics: A decomposition-based approach. *Annals of Mathematics and Artificial Intelligence*, 67(3-4):319–358, 2013.