Timothe Van Meter
Jordi Moya-Larano
Gabriel Barrionuevo

Weaver User's Guide

June 23, 2020

Contents

Part I

The Docker format

0.1 Introduction

FUNCTIONING on debian:

```
sudo docker run -it --rm -v ~/DOCKER/DockerWeaver/weaver:/root/weaver
-v ~/DOCKER/DockerWeaver/weaver_results:/root/weaver_results gbarrionuevo/weaverrelease
```

```
sudo docker run -it --rm -v ~/DOCKER/weaver-full:/root/weaver -v ~/DOCKER/weaver-full/weaver_resul
timothe92/weaver-full
```

```
cd /root/weaver/Release
make clean
make all
```

We then move to the directory `weaver_results`: `cd /root/weaver_results`

And there we can run the Weaver executable with the following command:
`/root/weaver/Release/weaverGit`

How to edit the docker containers and images
First run the docker image you want to edit in a container as follows

```
sudo docker run -it [image_name]
```

Then install edit or move files between the host and newly created container. To copy files from host machine to the container use the following:

```
sudo docker cp [path_to_file_on_host] [container_ID]:[destination_in_container]
```

Once all modifications are completed you need to commit the changes on the current container:

```
sudo docker commit [container_ID] [username]/[repository_name]:[version]
```

Then tag the image you just created. First you need to identify the image you created:

```
sudo docker images
```

This command will show something like this:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
timothe92/weaver-full	2.0	20102c87beaa	12 minutes ago	2.4GB
timothe92/weaver-full	version1.4	81fa1f280c0b	3 hours ago	2.36GB
timothe92/weaver-full	latest	54c6da543418	5 hours ago	2.36GB
timothe92/weaver-full	version1.3	54c6da543418	5 hours ago	2.36GB
timothe92/weaver-full	version1.2	d7a9bfc46947	6 hours ago	2.35GB
timothe92/weaver-full	version1	cf63936c4918	10 days ago	2.81GB

Choose the image you created it should be `[username]/[repository_name]:[version]`
Then tag it using this command:

```
sudo docker tag [image_ID] [username]/[repository_name]:[version]
```

Finally push the new image to dockerhub () to keep this new version safely stored online:

```
sudo docker push [username]/[repository_name]:[version]
```

Part II

Weaver Wraper

0.2 Contact

All correspondence concerning the wrapper program is to be sent to tvanme2@uic.edu with label "[WEAVER-WRAPPER]" at the beginning of the email's subject. The source code for `weaver-wrapper` can be found in one of the author's private github repository. To obtain access to the source code send an email with subject "DEMAND OF ACCESS TO SOURCE". If you encounter any problems using weaver-wrapper that are not answered within this guide please send an email with subject "BUG".

0.3 Wrapper's Organisation

0.3.1 Folders

configuration_files
This folder contains all the configuration files necessary for the simulation, mostly json files.
simulation_dir
This folder is automatically created upon executing make to store all the simulation, their corresponding configuration files and results.

0.3.2 Makefile

makefile
The makefile has two purposes: Once executed with arguments as follows: `/weaver-wrapper\$ make` it verifies that the files and their dependencies are all up-to-date. It can also be executed as `/weaver-wrapper\$ make clean` which will simply remove all the files with the `.pyc` extension, which are produced during the compilation of the python source code, but also all the files that are dependent on a specific execution of the wrapper, that is `simulation_number`, `sim_summary.csv`, `weaver-wrapper.log`.

0.3.3 User's Files

`wrapper_setup.json`
This json file permits the user to specify: the path of the folder containing the configuration files necessary for the simulation to perform, the path where the generate the `simulation_dir` and therefore all the simulation results, the number of desired replication per simulation and the path to the executable of weaver generated in Eclpise.
parameters.csv
This csv file is automatically created upon executing make, it contains the parameter that the user desire to see vary throughout the simulation. It presents for each parameter three values: a minimum, a maximum and the step value, combined they permit the acquisition of the total range of each parameter.

0.3.4 Temporary Files

`sim_summary.csv`
This csv file is created by the `sim_calculation.py` script. It contains all the combination of parameter value and the corresponding simulations in which those values will be use.
simulation_number
This file is simply a placeholder that permits the different script to communicate to one another the required number of simulation to run.
weaver−wrapper.log
This file is a log, it regroups the internal messages sent from all the different scripts of the wrapper. It provides a feedback and enables the user to find which part of the wrapper caused an error.

0.3.5 Python Scripts

`wrapper_initialisation.py`
This script controls the presence of the files `parameters.csv` and `wrapper_setup.json`, if they are not present it will automatically generate example files that the user can fill in afterwards.
main−build_wrapper.py
This script reads the user input in the `wrapper_setup.json` file and is in charge of createing the `simulation_dir`, one folder per simulation to run and and folders for replication if necessary. All of the folders created contain configuration files as well. The script `parameter_parser.py` is called from here to modify the configuration files with the appropriate value for each simulation.
parameter_parser.py
The parameter parser search for the parameter present in the `parameters.csv` file directly in the configuration files using their name as identifier and then replaces the value in the configuration file by the value corresponding to the simulation to be performed. This last information is obtained through the `sim_summary.csv` file.
sim_calculation .py
This script calculates for any parameter the value necessary for a given simulation, using the simulation number, the range of the parameter of interest and the cumulative range of all parameters, for more information on the exact functioning of this script see section 0.4.
main−run_wrapper.py
This script is in charge of calling the weaver executable with appropriate simulation folder and queuing all the simulations to run.

0.4 Rationale for the functioning of the wrapper

0.4.1 Creating a table of simulations to run (`sim_calculation.py`)

This program is in charge of calculating the number of simulations necessary to have every unique combination of each parameter values (specified in the [parameters.csv file](parameters.csv)) represented in a summary table. The program creates said table presenting for each simulation to be performed the value of each parameter. In the table, presented below, each line corresponds to a simulation to be performed. The number of total simulations, N, is simply the product of the range for every parameter that the user wants to vary in simulations.

$$N = \prod_i (max_i - min_i)/step_i + 1$$

With i the number of parameters to include in the simulations.

0.4.2 Filling Method for the table

To list all possible parameter value combination we explore recursively the range of the parameter from the first position, column position 1 to p, to the last position p.

Do a tikz diagram for the following ... (gonna take a little time)

Cell evaluation function

To evaluate the value needed in any given i, j position in the table we rely on our method to fill it up (see previous section)

$$C_j \cdot \prod_m (C_m)$$

With $\prod_m$, m going from $j - 1$ to p

Part III

Weaver

1

Overview

2

Installation & Set-up

3

Structure

Here, we describe the structure of the source code of Weaver and the dependencies between the different files. The source code consists of 15 files:

main.cpp
World.cpp
World.h
TerrainCell.cpp
TerrainCell.h
Animal.cpp
Animal.h
AnimalSpecies.cpp
AnimalSpecies.h
FungusSpecies.cpp
FungusSpecies.h
Edible.cpp
Edible.h
Random.cpp
Random.h

4

Platform Details

5

Animals Parameters

5.0.1 Experience

The parameter `experiencePerDay` is used to enable the organisms to change their edibility preferences through time by incrementing those edibilities by their encounter per day of certain organisms. The parameter is stored in each species `.json` file:

```
"experienceInfluencePerDay": "0.2",
```

what the it he weight that each day (important, not timeStep but day) puts into the overall experience. A value of 1 means that only the last day affects the experience. Lower values mean longer and longer term experience, as each new day matters less. Conf.

The experience influences a species behaviour by directly modifying the edibilities of other organisms or how likely this species is to feed on another organism:

```
2447    abundanceExperiencedPerSpecies[edibleToBePredated->getSpecies()] += fullDryMassToBeEaten;

2457    updateAbundancesExperiencedPerSpecies(){
            ET CAETERA
}
```

Equivalent expression for the calculation of the `abundanceExperiencedPerSpecies`, noted hereafter $abExpPS$.

$$abExpPS_i = \frac{abExpPS_i}{\sum_{i=0}^{allspecies} (abExpPS_i)} \tag{5.1}$$

```
2472    experienceInfluencePerTimeStep = mySpecies -> getExperienceInfluencePerDay() / timeStepsPe

2473    meanExperiencePerSpecies{
            ET CAETERA
}
```

Equivalent expression for the calculation of the `meanExperiencePerSpecies`, noted hereafter $mExpPS$.

$$mExpPS_i = abExpPS_i \cdot expIPTS + mExpPS \cdot (1 - expIPTS) \tag{5.2}$$