D3.4 – THIRD SPECIFICATION OF NEW METHODS, TOOLS AND MECHANISMS PROPOSED FOR THE SUPPORT OF THE APPLICATION USER AND PROGRAMMER

Grant Agreement	676547
Project Acronym	CoeGSS
Project Title	Centre of Excellence for Global Systems Science
Topic	EINFRA-5-2015
Project website	http://www.coegss-project.eu/
Start Date of project	2015-10-01
Duration	36 months
Deliverable due date	2018-04-30
Actual date of submission	2018-06-30
Dissemination level	Public
Nature	Report
Version	0.9
Work Package	WP3
Lead beneficiary	Chalmers
Responsible scientist	Patrik Jansson (CHALMERS)

Contributor(s)	Cezar Ionescu, Patrik Jansson, Sólrún Einarsdóttir, Michał Pałka, Johan Lodin (CHALMERS), Margaret Edwards (COSMO), Sergiy Gogolenko, Ralf Schneider (HLRS), Marcin Lawenda, Rafal Lichwala (PSNC), Eva Richter (UP), Fabio Saracco (IMT), Enrico Ubaldi, Luca Rossi (ISI)
Internal reviewers	Paweł Wolniewicz (PSNC), Michael Gienger (HLRS)
Keywords	HPC, Domain Specific Language, Synthetic Information System, Scalability, Visualisation, Co-design
Total number of pages:	68

The CoeGSS ("Centre of Excellence for Global Systems Science") project is funded by the European Union. For more information on the project please see the website http:// http://coegss-project.eu/

Version History

	Name	Partner	Date
v0.1	Initial version	Chalmers	2018-03-15
v0.2	Contributions from GCF, UP, IMT, PSNC, HLRS and IMT	Chalmers	2018-04-10
v0.3	Contributions from CoSMo, UP, HLRS and Chalmers	Chalmers	2018-04-16
V0.4	Merge of GCF, UP, and initial Chalmers input after first review	Chalmers	2018-04-23
V0.5	Table of contents changes requested by the internal reviewers	Chalmers	2018-04-30
V0.6	Addressed changes requested by the internal reviewers	Chalmers	2018-06-08

V0.7	Minor updates; sent for internal review.	Chalmers	2018-06-13
V0.8	Minor updates after internal review; submitted	Chalmers	2018-06-28
V1.0	Approved by ECM		2018-06-30

Abstract

Work package 3 (WP3) is a research and development work package with an overall aim to provide a set of tools and methods supporting the work of application programmers and GSS end users. These tools and methods can then be integrated into the portfolio of the centre and, where applicable, as direct services in the CoeGSS portal. This report is a living document, and the release at project month 6 was Deliverable 3.2, the release at month 21 was Deliverable 3.3, and the release at month 31 is **Deliverable 3.4** of WP3.

The first WP3 deliverable (Deliverable 3.1) was about the state-of-the-art: methods, tools and mechanisms (MTMs) available off-the-shelf at the start of the CoeGSS project. With this deliverable we continue to capture the status of new MTMs developed and planned by CoeGSS in WP3 (tasks T3.1–T3.6) and how WP3 has been utilized by the pilots in WP4. We proceed through the main areas covered by the six tasks of WP3, followed by a chapter on how WP3 and WP4 interact.

Table of Contents

Table of Abbreviations

ABM	Agent-Based Model
API	Application Programming Interface (a collection of subroutines and tools for building applications)
CKAN	Comprehensive Knowledge Archive Network (a data management system)
CoeGSS	Centre of excellence for Global System Science
COVISE	COllaborative VIsualization and Simulation Environment
CSV	Comma-Separated Values (a text-based file format)
DoA	Description of Action
DSL	Domain Specific Language
GIS	Geographic Information System
HDF5	Hierarchical Data Format, version 5 (a smart data container)
HDFS	Hadoop Distributed File System
HLRS	High-Performance Computing Centre Stuttgart (a site in CoeGSS)
HPC	High Performance Computing
JSON	JavaScript Object Notation (open-standard file format)
LAD	Local Authority District (one of the NUTS levels)
M	Month
MoTMo	Mobility Transformation Model
MS	Milestone
MTMs	methods, tools and mechanisms
NUTS	Nomenclature of Territorial Units for Statistics (an EU geocode standard)
OpenCOVER	Open COVISE Virtual Environment – a virtual reality rendering engine for COVISE
R	Review recommendation
SDP	Sequential Decision Problem
SEDAC	Socioeconomic Data and Applications Center (a NASA data centre)
SIS	Synthetic Information System
SQL	Structured Query Language (DSL designed to manage relational databases)
VR	Virtual Reality
WP	Work Package

List of Tables

List of Figures

1 Introduction

WP3 is a research and development work package supporting, directly and indirectly, the work of application programmers and GSS end users. As these tools and methods mature, they are integrated into the portfolio of the centre and, where applicable, are made available as direct services in the CoeGSS portal. The overall objectives of this work package for the full three-year period are the following according to the Description of Action (slightly edited[1]):

- To propose a prototype version of a heterogeneous environment consisting of HPC infrastructure and cloud storage to be used for scientific use cases (see D3.6).
- To provide enhanced fault tolerance skills in the proposed architecture (Chapter 2).
- To keep appropriate scalability for future large applications demanding a big data approach by increasing data efficiency (Chapter 2).

To develop data layer tools and services with a unified interface to the underlying technologies (see Deliverable 3.6 and Chapter The Deliverable 3.3 described the workflow and architecture, and capabilities of the CKAN platform. In this chapter, we focus on the comparison of platform performance for single and HA configurations. We also performed comparison tests for various PostgreSQL database configuration parameters. In Section 2.2 we provided performance tests of the synthetic population generation tool implemented for Health Habits use case. Tests are performed against different parameters like: creation of different synthetic population, splitting in the agents and locations, clustering procedure.

2.1 Performance of data management system

The data management system is of at most importance for every data centric GSS system. Extensive usage of data requires high performance solutions that is why we paid a lot of attention for appropriate tuning system parameters. In Section 2.1.1 we tested overhead related with implementation of High Availability (HA) infrastructure, while in Section 2.1.2 we concentrated on PostgreSQL database itself.

1.1.1 CKAN performance for single and HA configurations

The Comprehensive Knowledge Archive Network (CKAN) implementation used for data management is a powerful system that provides access to data – by providing tools to streamline publishing, sharing, finding, and using data. CKAN targets data publishers (national and regional governments, companies, and organizations), which desire making their data open and available.

The chapter describes the conversion time of input CSV (Comma-Separated Values) files from a MIDAS source to the database in a single node of the CKAN platform and a CKAN instance in high availability (HA) configuration.

[1] This list is from the WP3 objectives box on page 18 of DoA = Annex 1 of the Grant Agreement (p. 101/274).

In the CKAN HA configuration the data is replicated to a backup node. In single mode the data is saved only in local database instance.

After a successful CSV file upload, the CKAN datapusher plugin converts input data (comma-separated values) to a database representation – one input CSV file – one table in PostgreSQL database. We made 10 repetitions for each individual input files.

Table 1: Conversion time of CSV input files. MIN – minimum processing time, MAX – maximum processing time, AVG – average processing time.

	Single mode processing			HA mode processing		
CSV LINES	MIN [s]	MAX [s]	**AVG [s]**	MIN [s]	MAX [s]	**AVG [s]**
84	1	1	**1**	1	1	**1**
167553	267	287	**277**	275	296	**284**
640261	1042	1073	**1058**	1055	1089	**1070**
765022	1370	1427	**1401**	1398	1468	**1434**
1336855	3028	3085	**3057**	3091	3169	**3136**
1751555	3669	3756	**3710**	3747	3853	**3799**

The conversion process in HA mode takes 2% more than processing in a single node instance. The difference in time increases with the number of lines in the input files.

1.1.2 Optimization of the PostgreSQL database parameters

In the CKAN HA configuration the database records are replicated from instance N1 to instance N2.

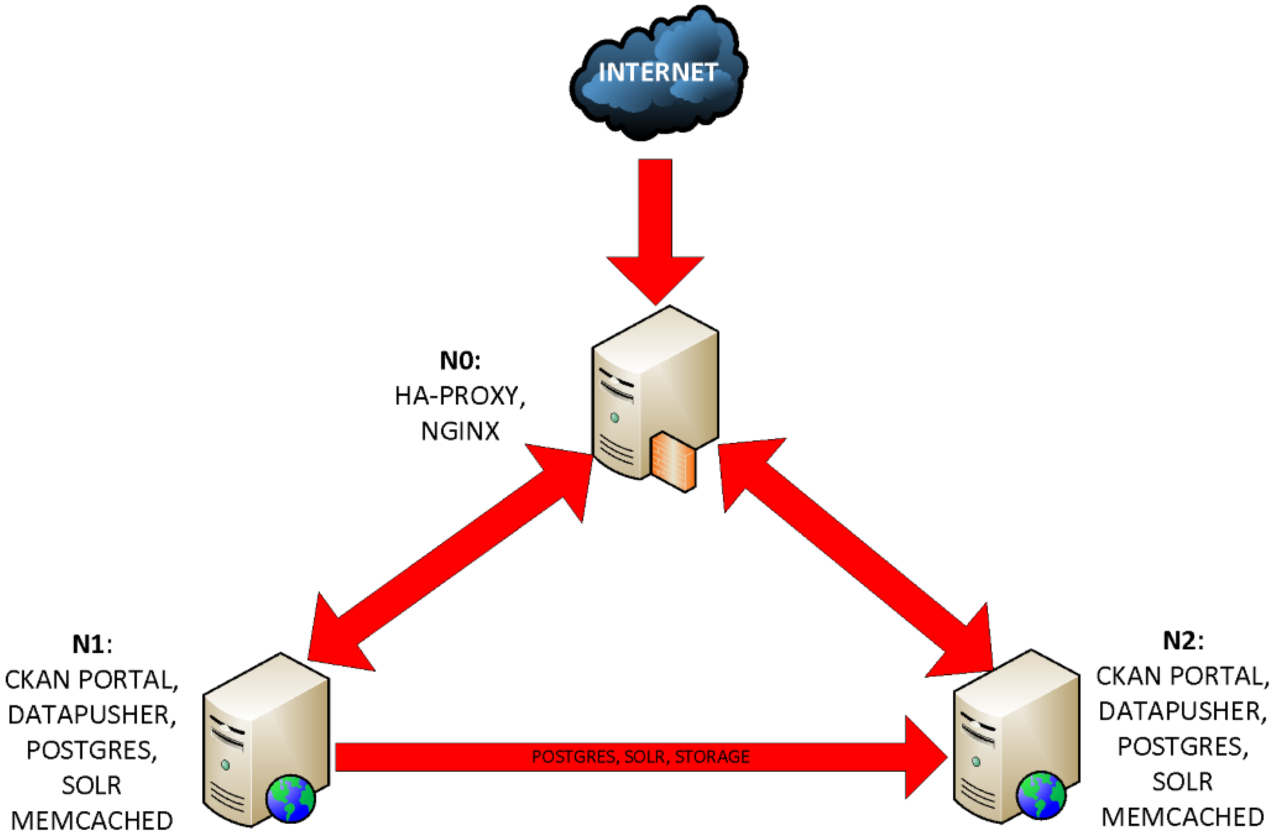


Figure 3: The CKAN HA configuration.

In the HA CKAN configuration we use Bi-Directional Replication for PostgreSQL (Postgres-BDR, or BDR), which is the open source multi-master replication system for PostgreSQL to reach full production status. BDR is specifically designed for geographically distributed clusters. It uses highly efficient asynchronous logical replication and supports anything from 2 to more than 48 nodes in a distributed database.

The PostgreSQL performance parameters are described on its wiki page . We analysed all PostgreSQL parameters and selected two of them to make performance tests:

- work_mem - amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files,
- shared_buffers - amount of memory the database server uses for shared memory buffers.

The other parameters in our environment turned out to be set as recommended.

The test input file was *household_sardegna.csv* from MIDAS which contains 640261 CSV records. We made 10 repetitions for individual parameter values.

Table 2: Test results of processing time of the optimization database parameters. MIN – minimum processing time, MAX – maximum processing time, AVG – average processing time.

shared_buffers / work_mem	MIN [s]	MAX [s]	AVG [s]
128MB / 4MB	924	1298	1 075
256MB / 4MB	1007	1114	1 060
512MB / 4MB	772	785	779
1GB / 4MB	725	1008	895
2GB / 4MB	708	1101	933
4GB / 4MB	711	720	720
8GB / 4MB	754	816	782
4GB / 8MB	714	988	895
4GB / 16MB	704	738	718
4GB / 32MB	712	1021	821

Some of the results differ significantly from the average value. The reason may be the load on the OpenStack environment on which the CKAN is run in the HA configuration.

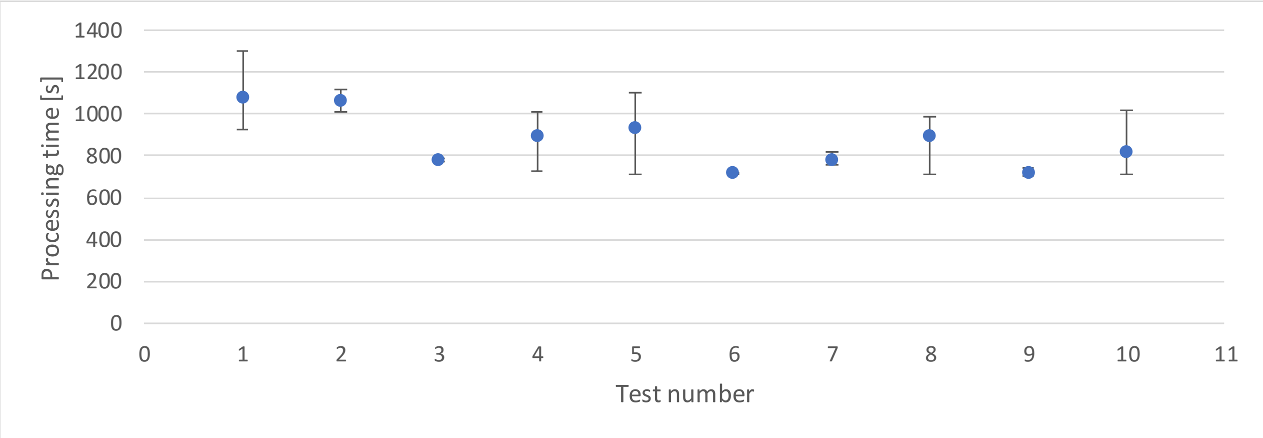


Figure 4: Graph is presenting AVG processing time with MIN and MAX values.

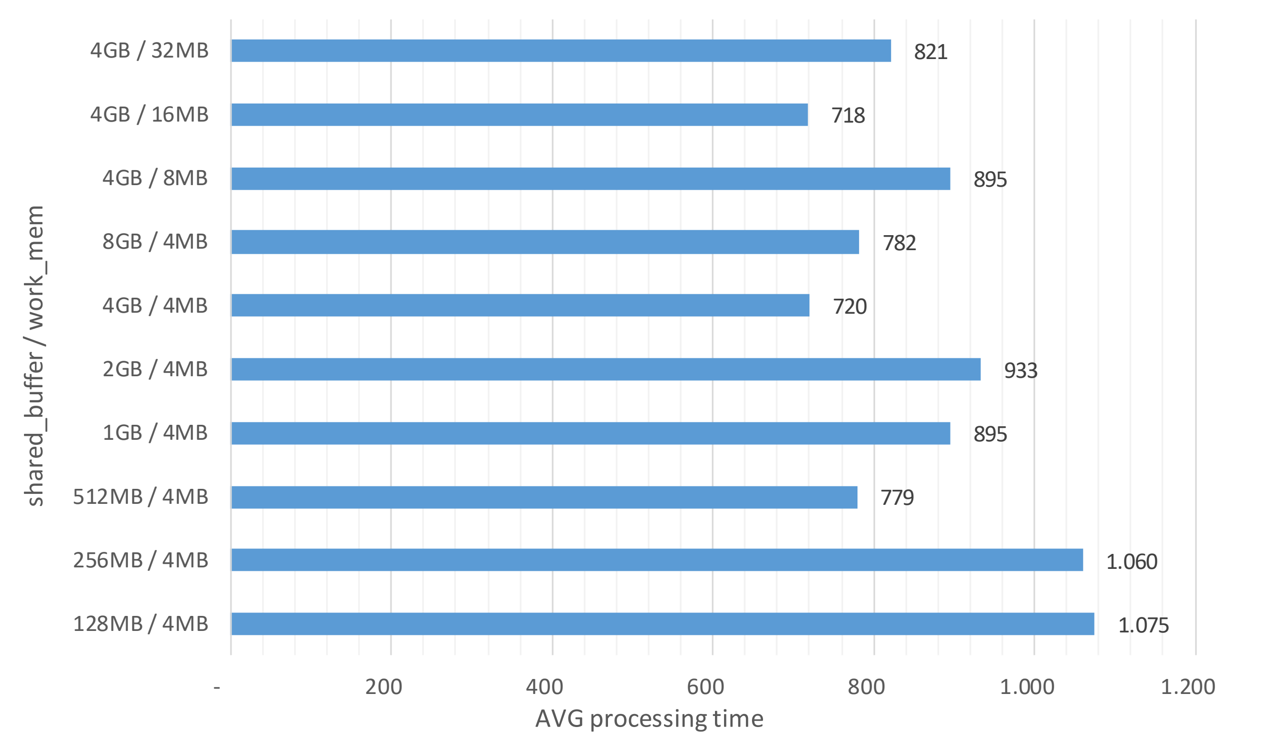


Figure 5: Graph is presenting the AVG processing time.

The best test results were obtained by configuring the parameters shared_buffers = 4GB and work_mem = 16MB. Compared to the default parameters values (shared_buffers = 128MB and work_mem = 4MB**)**, the average processing time of the datapusher plugin has decreased by about 6 minutes (see Table 2). The values of these parameters have been set as production.

1.2 ISI SynPop tool performance

Here we report the results of the computing time scalability testing of the synthetic population generation tool written by the health habits pilot. The tool is composed of different python modules containing the routines and classes needed to generate and spatially arrange the agents, the households where they live and the schools/workplaces they attend/work. The SynPop creation process comprehends two main parts:

- *generation*: create the entities (agents, households and workplaces), generate their features (e.g., for agents set their sex, age, income etc.) and assign them to a municipality in the geographic area of interest;
- *clustering*: once all the locations (households, schools and workplaces) are crated scatter them in space proportionally to the population density and group these entities in hierarchical groups of given sizes to create local clusters of people interacting at different levels. For example, here we group all the people living in one municipality in districts of ~5000 people that are sub-divided in local communities of ~800 people further split in neighbourhoods (household clusters) of ~100 people.

The creation process relies on o a geographic database and on pre-processed Eurostat data that requires additional computing time and code overhead that has to be performed once for all, so we did not include it in this computing time estimate.

Since this tool does not require HPC machines but rather large node memory, large disk capacity and a large number of threads working with shared memory, tests were performed on a virtual machine provided by PSNC with 32GB of memory, 32 computing cores and 1TB disk space.

The population has been created for three different regions in the north-west of Italy so as to test different population sizes:

- ITC1: Piedmont region, i.e., the whole region (population ~4'380'000 people);
- ITC11: Turin, the largest province of the region (population ~2'270'000 people);
- ITC13ITC14: two small provinces (Biella and Verbania) with a total population of ~337'000 people).

For each selected region we generated the 10, 25, 50 75 and 100 percent of the population. Each use case is reported in Figures 1 and 2 as *(region code, population percentage)*, so that, for example, *(ITC1, 0.50)* refers to the creation of 50% of the entire Piedmont population.

We observe that the code overhead to set up the creation process and to perform the clustering procedure in each municipality deeply affects the computing time per agent in the creation process. Nevertheless, we recorded a constant decrease of computing time per agent in large populations so that the code can easily scale up to very large populations. However, this result calls for an overall improvement of the code parallelization, especially in the clustering part that can be performed at once on several municipalities by different threads instead of the current serial implementation.

In the final stage of the project we are tending to concentrate on testing and improving performance of implemented by the project community tools like: Chalmers Synthetic Population application for based on HH and GG micro samples, Network Reconstruction tool, simulation application MoTMo. Moreover extra attention will be paid to testing existing solutions (libraries) improving application reliability (e.g. checkpointing). It will result as guideline for application developers interested in improving reliability.

- Data Analytics0).
- To provide remote and immersive visualisation (Chapter 4).
- To provide DSLs for assembling GSS simulations (Chapter 5).
- To develop validated numerical methods for GSS simulations (Chapter 6).
- To develop a clear concept and support services for the hardware / software co-design of future needs coming from the users' communities (Chapter 7).

This report has been a living document and the release at project month 6 was Deliverable 3.2. The second release in M21 was Deliverable 3.3 and the third release in M33 is Deliverable 3.4 (this deliverable). The first deliverable (Deliverable 3.1) was about the state-of-the-art: methods, tools and mechanisms (MTMs) available off-the-shelf at the start of the CoeGSS project. With Deliverable 3.2 we proposed new MTMs based on the "gap" between WP3 (research tasks T3.1–

T3.6) and WP4 (the pilots). And now, at month 31, we capture a final snapshot of the MTMs under development in CoeGSS in Deliverable 3.4.

In CoeGSS the High Performance Computing community (here represented by WP3) meets with the Global Systems Science community (represented by WP4). Deliverable 3.2 was a first step towards bridging the gap between the two communities and this deliverable captures the progress made in the first 33 months of the project. We go through the tasks of WP3 in the same order as in Deliverable 3.1, Deliverable 3.2 and Deliverable 3.3 and we end with a chapter providing examples about how the GSS pilots in WP4 have interacted with WP3.

Note that additional information about the software side is provided in three WP3 deliverables on "Documentation and software": Deliverables 3.5 (month 18), 3.6 (month 28), and 3.7 (month 36).

1.3 Executive summary

Each of the six following chapters specify new methods, tools and mechanisms (MTMs) from the point of view of the six tasks of WP3 in CoeGSS.

- Chapter 2 describes the work on enhanced reliability and scalability: performance tests of the CKAN platform and database parameter optimization.
- Chapter 0 describes the Dakota framework for advanced parametric analyses, design exploration, and model calibration with computational models, and how it connects to the models through the data format described in Chapter 7.
- Chapter 4 describes how the CoeGSS visualisation toolkit can combine local and remote processes to make a hybrid visualisation system.
- Chapter 5 describes MTMs for formalisation of policy relevant GSS concepts and their translation into computational elements. One computational element is a generic method for network reconstruction based on information theoretic similarity measures, used to construct synthetic friendship networks.
- Chapter 6 describes an implementation of interval-based methods for validated numerics in GSS simulations.
- Chapter 7 describes the recent progress on specification and support of a unified HPC compliant HDF5 file structure for CoeGSS tools.
- Finally, Chapter 8 deals with utilization of the above work: how it is, and can be, used by the WP4 pilots, followed by a summary in Chapter 9.

2 Enhanced Reliability and Scalability

The Deliverable 3.3 described the workflow and architecture, and capabilities of the CKAN platform. In this chapter, we focus on the comparison of platform performance for single and HA configurations. We also performed comparison tests for various PostgreSQL database configuration parameters. In Section 2.2 we provided performance tests of the synthetic population generation tool implemented for Health Habits use case. Tests are performed against different parameters like: creation of different synthetic population, splitting in the agents and locations, clustering procedure.

2.1 Performance of data management system

The data management system is of at most importance for every data centric GSS system. Extensive usage of data requires high performance solutions that is why we paid a lot of attention for appropriate tuning system parameters. In Section 2.1.1 we tested overhead related with implementation of High Availability (HA) infrastructure, while in Section 2.1.2 we concentrated on PostgreSQL database itself.

2.1.1 CKAN performance for single and HA configurations

The Comprehensive Knowledge Archive Network (CKAN) implementation used for data management is a powerful system that provides access to data – by providing tools to streamline publishing, sharing, finding, and using data. CKAN targets data publishers (national and regional governments, companies, and organizations), which desire making their data open and available.

The chapter describes the conversion time of input CSV (Comma-Separated Values) files from a MIDAS (MIDAS data source 2018) source to the database in a single node of the CKAN platform and a CKAN instance in high availability (HA) configuration.

In the CKAN HA configuration the data is replicated to a backup node. In single mode the data is saved only in local database instance.

After a successful CSV file upload, the CKAN datapusher plugin converts input data (comma-separated values) to a database representation – one input CSV file – one table in PostgreSQL database. We made 10 repetitions for each individual input files.

Table 1: Conversion time of CSV input files. MIN – minimum processing time, MAX – maximum processing time, AVG – average processing time.

	Single mode processing			HA mode processing		
CSV LINES	MIN [s]	MAX [s]	**AVG [s]**	MIN [s]	MAX [s]	**AVG [s]**
84	1	1	**1**	1	1	**1**
167553	267	287	**277**	275	296	**284**
640261	1042	1073	**1058**	1055	1089	**1070**
765022	1370	1427	**1401**	1398	1468	**1434**
1336855	3028	3085	**3057**	3091	3169	**3136**

1751555	3669	3756	**3710**	3747	3853	**3799**

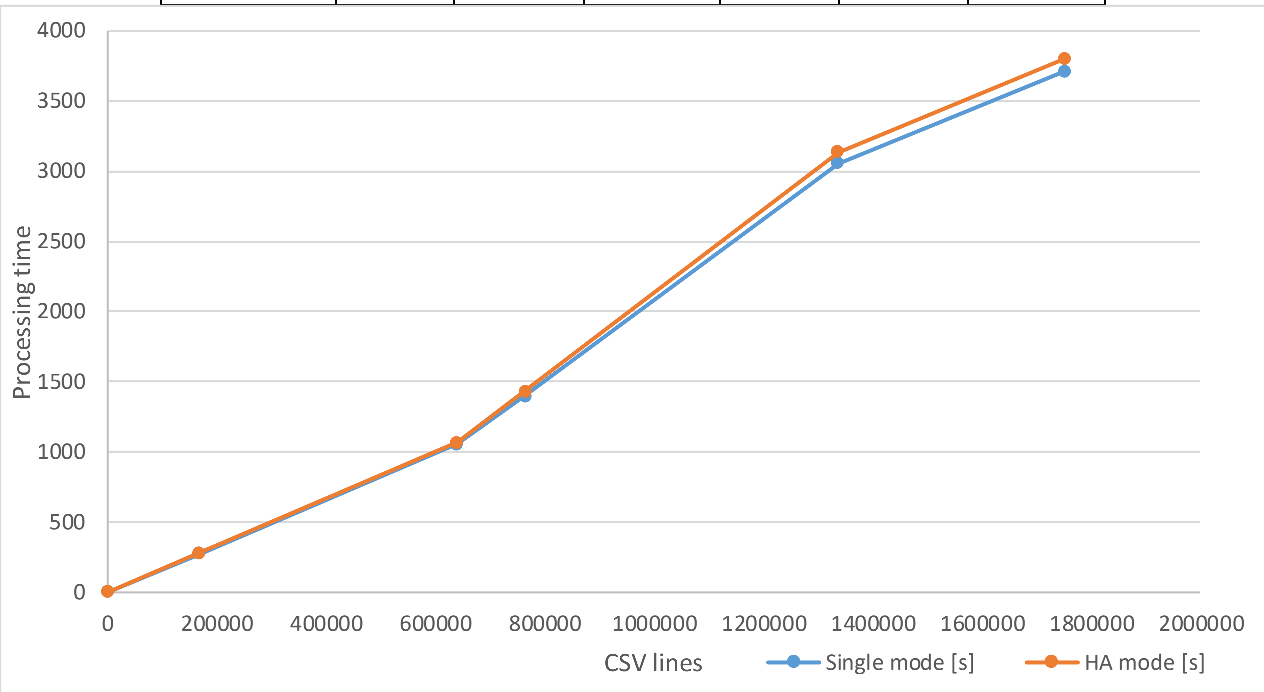


Figure 1: Line graph is presenting the AVG conversion time for single and HA mode.

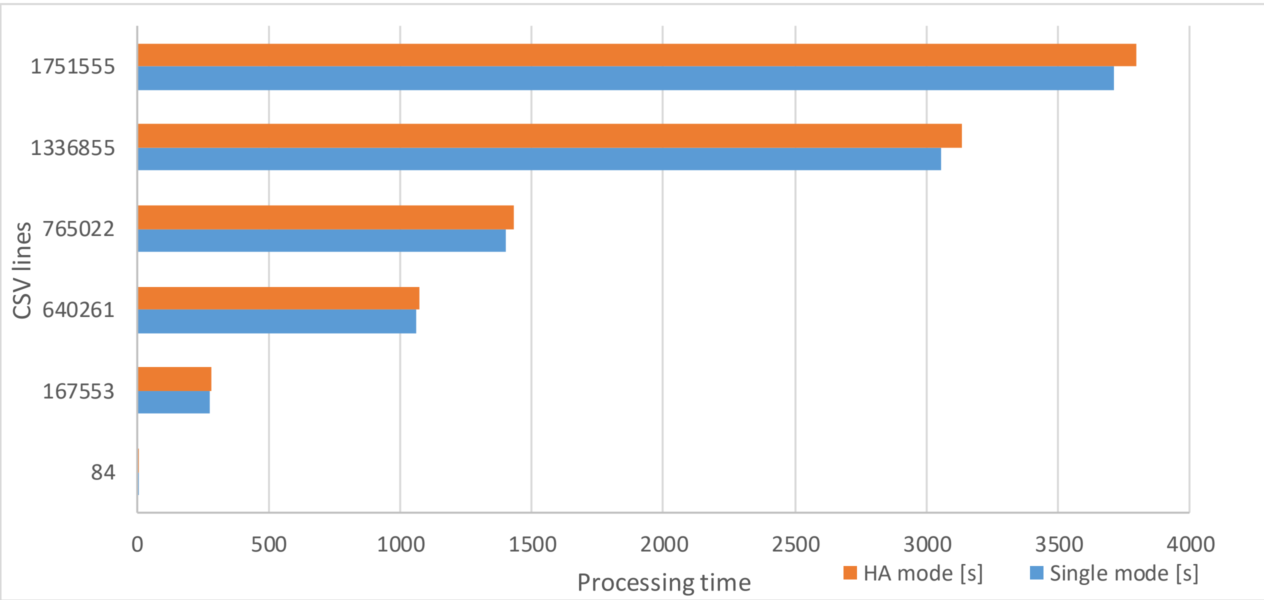


Figure 2: Bar graph is presenting the AVG conversion time for single and HA mode.

The conversion process in HA mode takes 2% more than processing in a single node instance. The difference in time increases with the number of lines in the input files.

2.1.2 Optimization of the PostgreSQL database parameters

In the CKAN HA configuration the database records are replicated from instance N1 to instance N2.

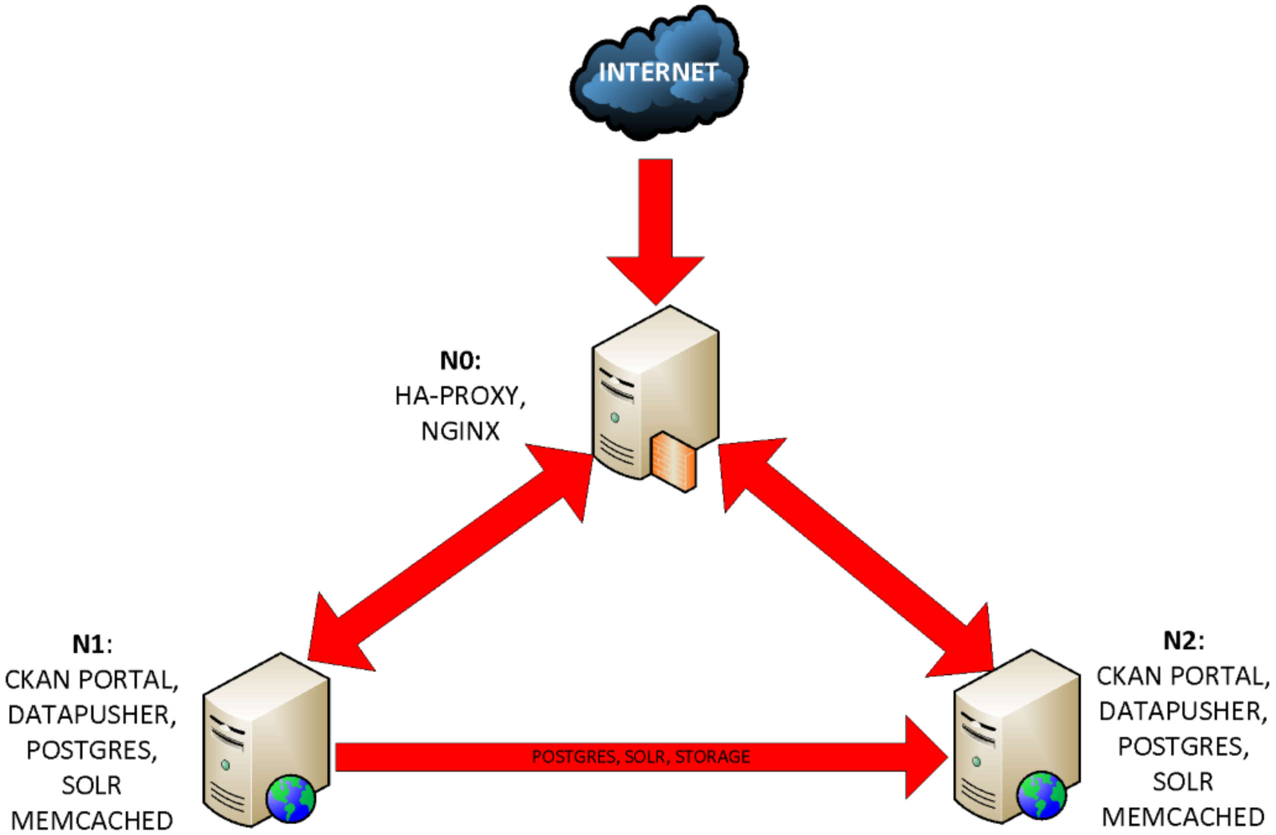


Figure 3: The CKAN HA configuration.

In the HA CKAN configuration we use Bi-Directional Replication for PostgreSQL (Postgres-BDR, or BDR), which is the open source multi-master replication system for PostgreSQL to reach full production status. BDR is specifically designed for geographically distributed clusters. It uses highly efficient asynchronous logical replication and supports anything from 2 to more than 48 nodes in a distributed database.

The PostgreSQL performance parameters are described on its wiki page (PostgreSQL Wiki - Tuning Your PostgreSQL Server n.d.). We analysed all PostgreSQL parameters and selected two of them to make performance tests:

- work_mem - amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files,
- shared_buffers - amount of memory the database server uses for shared memory buffers.

The other parameters in our environment turned out to be set as recommended.

The test input file was *household_sardegna.csv* from MIDAS (MIDAS data source 2018) which contains 640261 CSV records. We made 10 repetitions for individual parameter values.

Table 2: Test results of processing time of the optimization database parameters. MIN – minimum processing time, MAX – maximum processing time, AVG – average processing time.

shared_buffers / work_mem	MIN [s]	MAX [s]	AVG [s]
128MB / 4MB	924	1298	1 075
256MB / 4MB	1007	1114	1 060
512MB / 4MB	772	785	779
1GB / 4MB	725	1008	895
2GB / 4MB	708	1101	933
4GB / 4MB	711	720	720
8GB / 4MB	754	816	782
4GB / 8MB	714	988	895
4GB / 16MB	704	738	718
4GB / 32MB	712	1021	821

Some of the results differ significantly from the average value. The reason may be the load on the OpenStack environment on which the CKAN is run in the HA configuration.

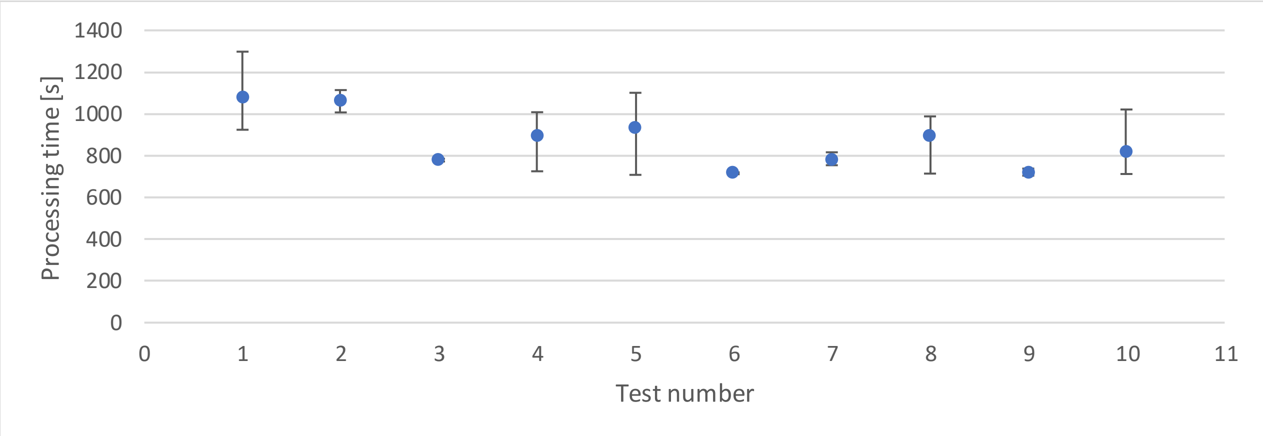


Figure 4: Graph is presenting AVG processing time with MIN and MAX values.

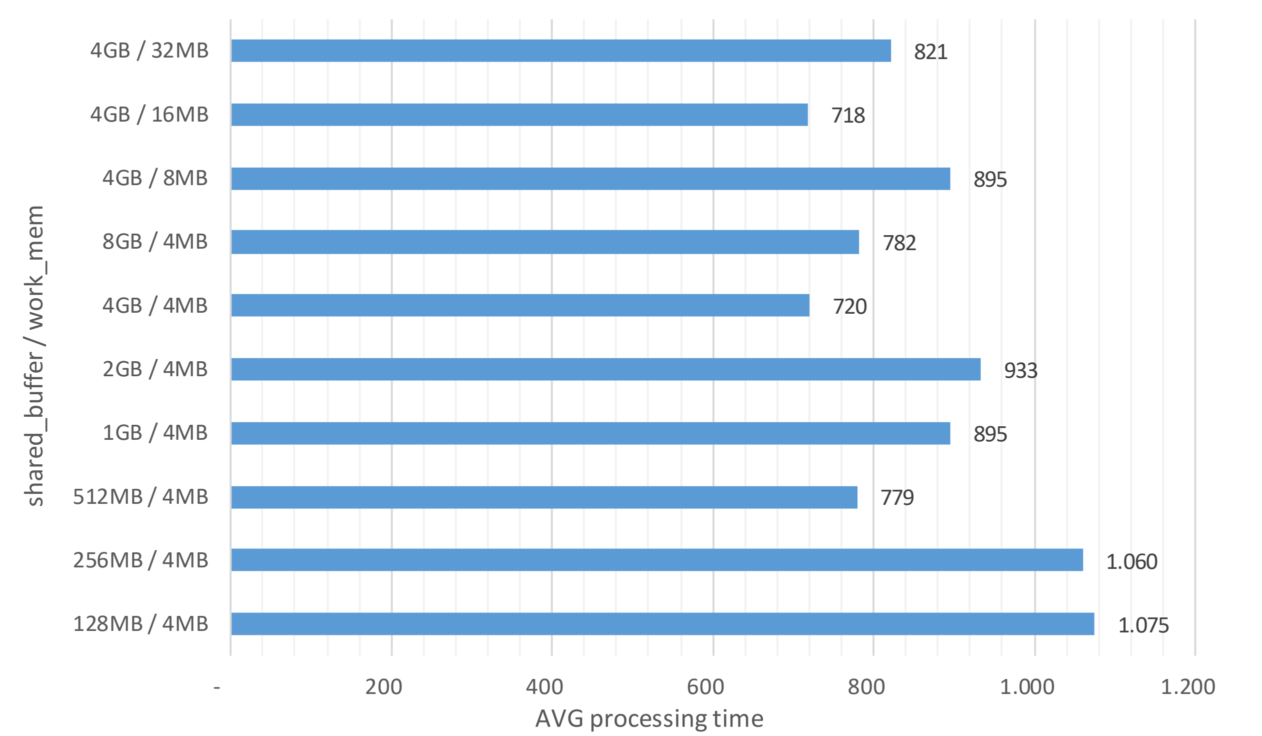


Figure 5: Graph is presenting the AVG processing time.

The best test results were obtained by configuring the parameters shared_buffers = 4GB and work_mem = 16MB. Compared to the default parameters values (shared_buffers = 128MB and work_mem = 4MB**)**, the average processing time of the datapusher plugin has decreased by about 6 minutes (see Table 2). The values of these parameters have been set as production.

2.2 ISI SynPop tool performance

Here we report the results of the computing time scalability testing of the synthetic population generation tool written by the health habits pilot. The tool is composed of different python modules containing the routines and classes needed to generate and spatially arrange the agents, the households where they live and the schools/workplaces they attend/work. The SynPop creation process comprehends two main parts:

- *generation*: create the entities (agents, households and workplaces), generate their features (e.g., for agents set their sex, age, income etc.) and assign them to a municipality in the geographic area of interest;
- *clustering*: once all the locations (households, schools and workplaces) are crated scatter them in space proportionally to the population density and group these entities in hierarchical groups of given sizes to create local clusters of people interacting at different levels. For example, here we group all the people living in one municipality in districts of ~5000 people that are sub-divided in local communities of ~800 people further split in neighbourhoods (household clusters) of ~100 people.

The creation process relies on o a geographic database and on pre-processed Eurostat data that requires additional computing time and code overhead that has to be performed once for all, so we did not include it in this computing time estimate.

Since this tool does not require HPC machines but rather large node memory, large disk capacity and a large number of threads working with shared memory, tests were performed on a virtual machine provided by PSNC with 32GB of memory, 32 computing cores and 1TB disk space.

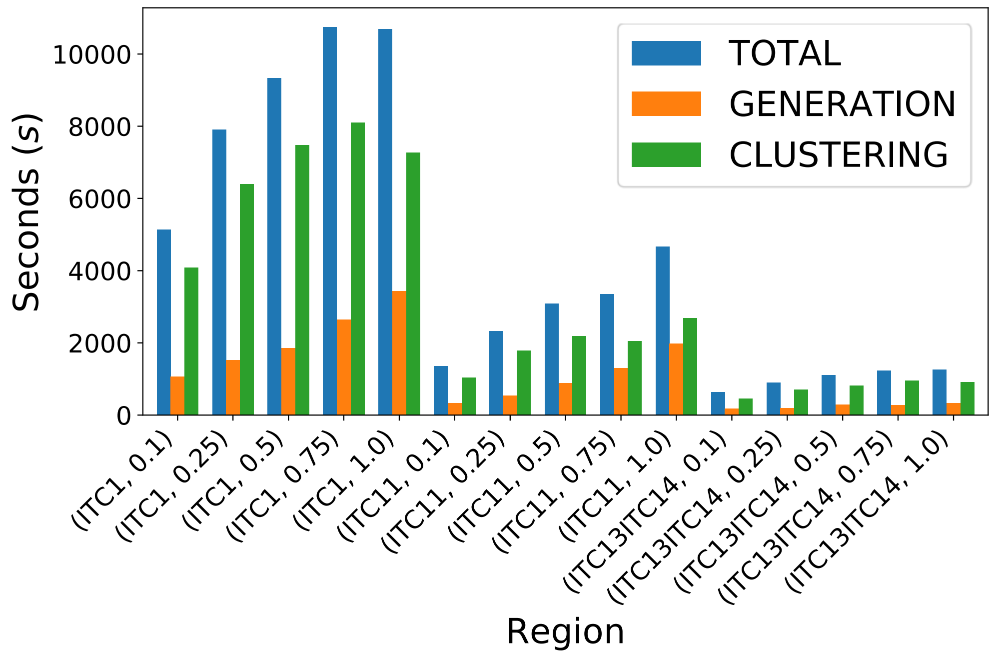


Figure 6: The total time spent in the creation of different synthetic population (blue bars). The time is split in the agents and locations table generation (orange bars) and in the local clustering procedure (green bars).

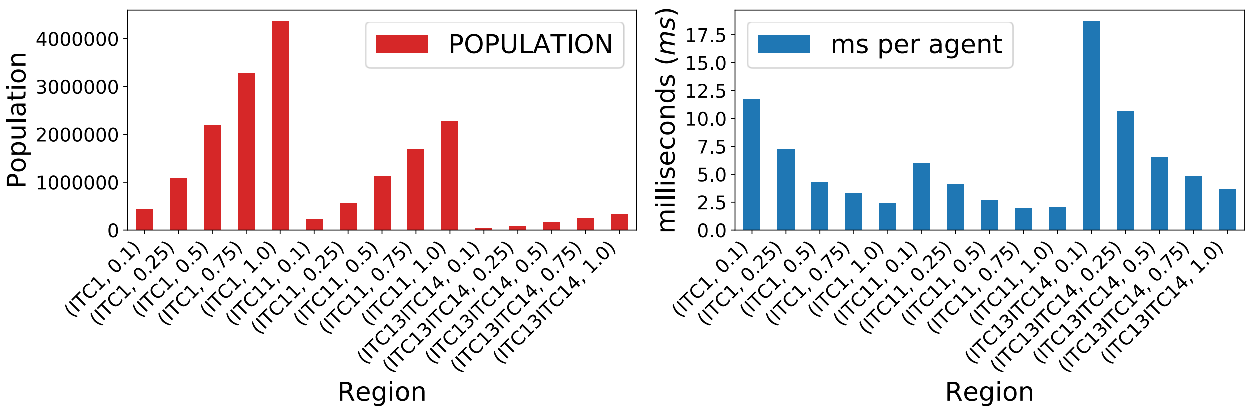


Figure 7: The population (number of agents) to be created (left plot) and the computing time spent per agent (right plot) for all the simulated scenarios.

The population has been created for three different regions in the north-west of Italy so as to test different population sizes:

- ITC1: Piedmont region, i.e., the whole region (population ~4'380'000 people);
- ITC11: Turin, the largest province of the region (population ~2'270'000 people);

- ITC13ITC14: two small provinces (Biella and Verbania) with a total population of ~337'000 people).

For each selected region we generated the 10, 25, 50 75 and 100 percent of the population. Each use case is reported in Figures 1 and 2 as *(region code, population percentage)*, so that, for example, *(ITC1, 0.50)* refers to the creation of 50% of the entire Piedmont population.

We observe that the code overhead to set up the creation process and to perform the clustering procedure in each municipality deeply affects the computing time per agent in the creation process. Nevertheless, we recorded a constant decrease of computing time per agent in large populations so that the code can easily scale up to very large populations. However, this result calls for an overall improvement of the code parallelization, especially in the clustering part that can be performed at once on several municipalities by different threads instead of the current serial implementation.

In the final stage of the project we are tending to concentrate on testing and improving performance of implemented by the project community tools like: Chalmers Synthetic Population application for based on HH and GG micro samples, Network Reconstruction tool, simulation application MoTMo. Moreover extra attention will be paid to testing existing solutions (libraries) improving application reliability (e.g. checkpointing). It will result as guideline for application developers interested in improving reliability.

3 Data Analytics

Deliverable 3.3 and Deliverable 3.5 addressed the processing and information management technologies for ABM simulations. In the second project year, as the ABM modelling stack became more mature, the requirements of the pilots for methods supporting the execution of parameter studies and optimization tasks on HPC systems arose. After detailed discussions in this direction in the second project year (see Deliverable 4.3), it was decided to evaluate the capabilities of Dakota – a framework for *"advanced parametric analyses, design exploration, model calibration, risk analysis, and quantification of margins and uncertainty with computational models."* (Adams, 2017). Dakota is open source under GNU LGPL[2].

3.1 High-level overview of Dakota

Dakota is a framework that be used either as a C++ library, giving tight integration with the simulation code, or used externally with no update of the simulation code. Instead, pre- and post-processing is used to communicate from Dakota to the standalone execution of the simulation, and back to Dakota again. This intermediary software, that Dakota sees, is called a driver. See Figure 8.

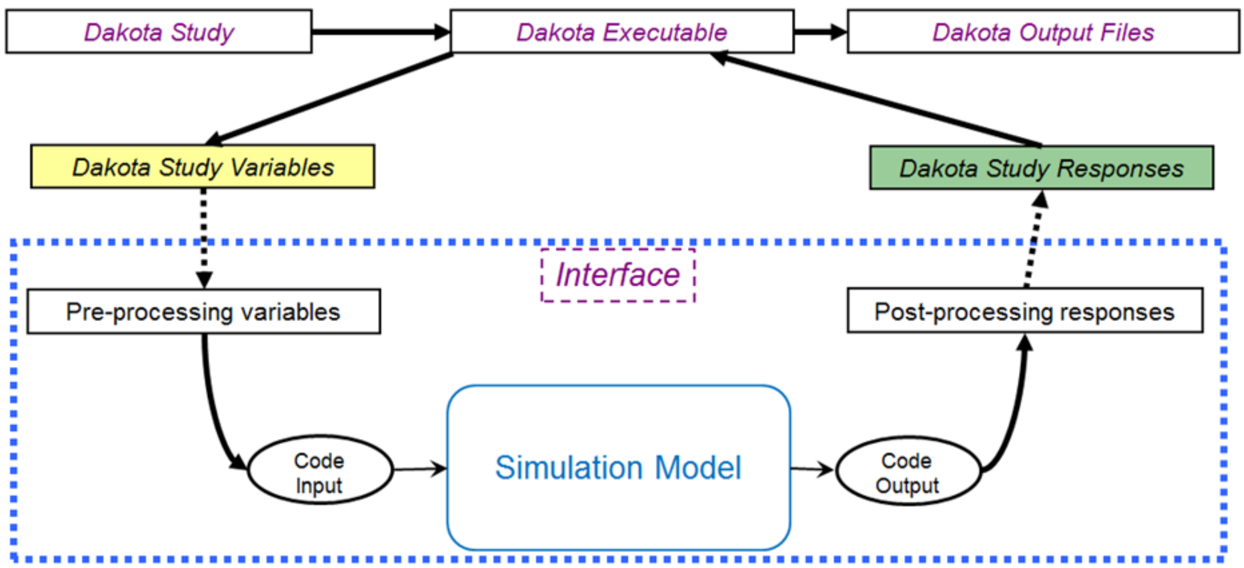


Figure 8: Dakota flowchart for running external simulation models. (Adams, 2017)

In Figure 8, the boxes have the following meaning:

- The *Dakota study* block represents a text file containing a description of the iterative method Dakota will use, the model used by the iterative method, what variables the simulation models uses, how the simulation model is interfaced, and how the simulation model responds.
- The *Dakota Executable* is the program the end user starts.

[2] https://www.gnu.org/copyleft/lesser.html

- The *Dakota Study Variables* block represents the set of variables the simulation should be executed with for an iteration.
- The *Pre-processing variables* block represents transforming Dakota's file format into a format that the simulation model understands (which could be command-line arguments).
- The *Simulation model* block is the program that implements the simulation, e.g. a Python or C program.
- The *Post-processing responses* block transforms the model output to a form that matches what was described in the Dakota study file.
- And finally, the *Dakota Output Files* represents the output that Dakota generates.

Since D3.3 a common file format for CoeGSS ABM output has been developed and a library for reading/writing output files have been created, see Chapter 7. We plan to use this format with Dakota in order to simplify and speedup the post-processing step.

For a given model/driver, running another study encompasses providing a new study file and executing Dakota. The Dakota execution for a model will be made accessible through a Cloudify blueprint, see D5.12.

In the rest of this chapter, we describe the Dakota package in detail and demonstrate its usage for adaptive parameter space exploration by means of a basic graph-based ABM model executing the social diffusion algorithm described by Bonabeau (Bonabeau, 2002).

3.2 Basic usage of Dakota

To get an idea of how to use Dakota it is definitely advised to work through the Examples section 2.3 of the Dakota user's manual (Adams, 2017). The execution of the examples on HPC systems is straightforward besides the fact, that the default setup of the input files provided with the examples contain the "`graphics`" keyword within the "`environment`" specification. This keyword triggers Dakota to open a graphical window, which displays, e.g., the convergence process of the executed method. The "`graphics`" keyword has to be commented out from the "`environment`" specification before the execution of the examples on any HPC system without X-forwarding. Especially interesting are the two examples given in section 2.3.5 of the Dakota user's manual, which show how to use Dakota to drive external simulation codes whose functionality is not visible to Dakota (so called black box codes).

Dakota is installed in both HPC centers – HLRS and PSNC. The following subsections discuss specific aspects of these installations.

3.2.1 Installation on HLRS

On the HLRS HPC-system HazelHen, we installed Dakota version 6.7 from sources available from the Dakota Homepage[3]. This installation is accessible via the module environment. To make the framework usable issue the command:

[3] https://dakota.sandia.gov/downloads

```
$ module load tools/dakota
```

Since Dakota was compiled from source using the GNU compiler suite, as cross compilers targeting HazelHen's Intel Haswell compute node architecture, and since it requires the TRILINOS solver package and the BOOST libraries for correct functionality, it is also necessary to load additional modules by issuing the following commands

```
$ module load PrgEnv-gnu
$ module load cray-tilinos
$ module load tools/boost/1.66.0
```

The tutorial files which are referenced in Dakota user's manual (Adams, 2017) can be found in the installation directory of HazelHen under /opt/hlrs/tools/dakota/6.7.0/examples.

3.2.2 Installation on PNSC

As the Operating System of the PSNC HPC-system Eagle is binary compatible with Red Hat Enterpise Linux 6[4], the pre-build binaries from the Dakota Homepage[5] can be used. After adding the 'dakota/bin' folder to the PATH and LD_LIBRARY_PATH environment variables, it's only necessary to load the LAPACK[6] module, and everything is set up.

Nevertheless, a Dakota module will be added in the next weeks, which will be built from source. This build will enable the Python direct interface, which is not part of the pre-build binaries.

3.3 Serial usage example

At this point we give a short overview about the steps which have been taken to transform the original implementation of the social diffusion problem into one that can be coupled to Dakota via the two filesets `params.in.<no>` and `results.out.<no>` and a so called driver script.

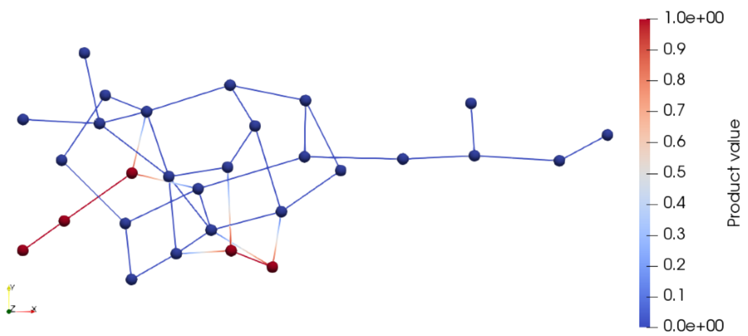


Figure 9: Erdős–Rényi random network.

As a demo problem the Erdős–Rényi random network shown in Figure 9 is considered. The posed problem is to find a pair of parameters d and Θ determining the diffusion speed so that after the

[4] https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux
[5] https://dakota.sandia.gov/downloads
[6] http://www.netlib.org/lapack/

execution of a fixed number of diffusion steps, the mean product value in the network is as close to 0.8 as possible.

To couple the original executable to Dakota it was at first necessary to add three, from the implementation point of view, rather straight forward capabilities to the simulation code.

1. **The determination of the value of the objective function.** In this simple test and demonstration example we have chosen the objective function to be:
$$f(\Theta, n) = \left|0.8 - \frac{1}{N}\sum_{k=1}^{N} \hat{V}_k\right|$$
Its evaluation was directly implemented into the simulation code even though it would have been possible to do this evaluation in an external post-processing step. Since the usage of an external post-processing procedure would require writing at least the last step's result data to disk one should in every case consider implementing the evaluation of the objective function directly into the simulation code.
2. **Writing the value of the objective function to a file** named `results.out.<no>` which is read by Dakota.
3. **Taking a unique job id as input parameter.** This capability is necessary in case of parallel execution to avoid having different instances of the simulation writing the value of the objective function to the same result file.

Since Dakota generates for each needed execution of the simulation code coupled to it a file named `params.in.<no>`, which contains the set of parameters which have to be evaluated, it is in almost every case necessary to wrap the actual call to the application by a so called driver script. The functionality the driver script has to provide in general is:

1. Taking the filename of the `params.in.<no>` file as its single command line parameter. The base-name `params.in`, can be modified within the input section of Dakota's input file. Dakota will append the number of the respective execution of the `simulation .<no>` to the base-name.
2. Reading the parameters determined by Dakota, in the given case n and Θ, from the `params.in.<no>` file and pass them to the actual simulation application. Since the simulation code used in this demonstration example takes all its parameters from the command line the conversion of the two parameters can be done with two `grep` commands within the driver script (see Figure 10).
3. An additional post-processing step after the execution of the simulation code can be the calculation of the objective function's value from the simulation results. As already stated above this step should be avoided if possible to reduce the load of the filesystems.

The implementation of the complete workflow is shown in Figure 10 where boxes in green represent the two participating binary executables of Dakota and the social diffusion problem simulation code, blue boxes represent files, the orange box represents the driver script and the grey box variables within the driver script, which are directly passed as command line parameters to the `graph_abm` executable.

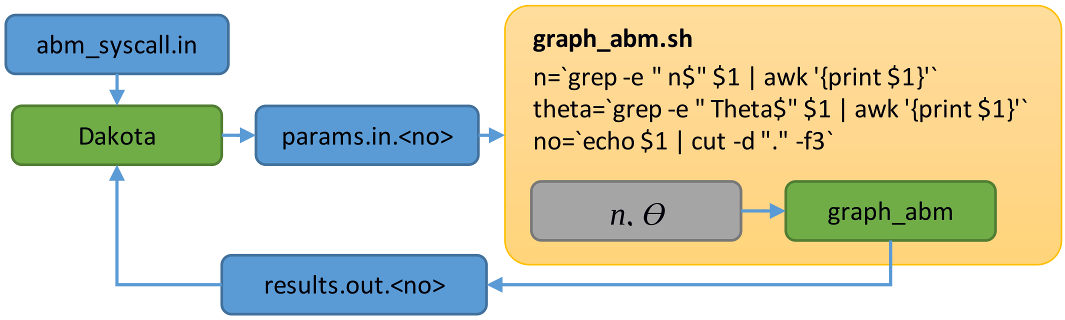
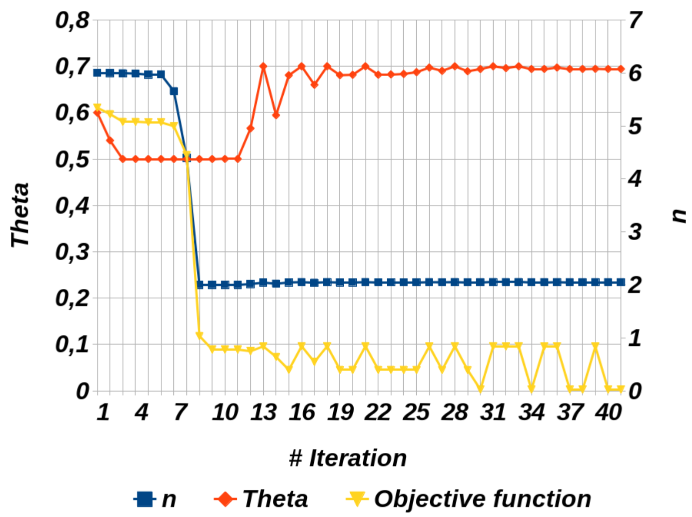


Figure 10: Coupling the demonstration application to Dakota.

The content of the Dakota input file used to steer the example problem is given in Figure 11.

Once Dakota manages the convergence of the posed problem, in the given case the minimization of the objective function, the output files can be reviewed. It contains the convergence process, the optimal set of parameters as well as other information about the solution process. Figure 11 presents the convergence graphs of the parameter optimization. One can see that the optimization method needed 41 iterations to find an optimal solution. These iterations included 71 evaluations of the objective function i.e. solutions of the social diffusion problem.




```
# Dakota Input File: abm_syscall.in
environment
  tabular_graphics_data
  tabular_graphics_file = 'abm_syscall.dat'
method
  max_iterations = 100
  convergence_tolerance = 1e-6
  conmin_frcg
model
  single
variables
  continuous_design = 2
    initial_point      6.0       0.6
    lower_bounds       2.0       0.5
    upper_bounds      10.0       0.7
    descriptors        'n'       'Theta'
interface
  analysis_driver = 'graph_abm.sh'
    file_tag
    file_save
    fork
    evaluation_scheduling peer dynamic
    parameters_file = 'params.in'
    results_file    = 'results.out'
responses
  objective_functions = 1
  numerical_gradients
    method_source dakota
    interval_type forward
    fd_gradient_step_size = 1.e-4
  no_hessians
```

Figure 11: Convergence of the test problem and Dakota input

3.4 Testing Dakota’s parallel capabilities on HazelHen

In order to speed up the optimization process the next step was to test whether the parallel processing capabilities of Dakota are applicable on the HazelHen HPC system. Out of the different parallel execution approaches that are implemented in Dakota and that are described in detail in chapter 17 of the Dakota user’s manual, we tested the single-level parallelism via the message passing interface (MPI). The corresponding example which was taken as the base line for the presented test case can be found on the HazelHen system under `/opt/hlrs/tools/dakota/6.7.0/examples/parallelism/Case1-MassivelySerial`. To enable the parallel execution of Dakota the fork together with the evaluation scheduling peer dynamic option was used in the interface section of the Dakota input file as presented in Figure 11.

To do a first evaluation of the parallel execution with a computationally more expensive problem, another Erdős–Rényi random network with 17177 nodes and 20000 edges was generated and the same optimization problem like with the small network was executed. In serial mode the total execution time for 4 iterations with 8 evaluations of the objective function, i.e. 8 executions of the social diffusion problem, was 43.7 sec. This time was reduced to 34.2 sec by running Dakota in parallel with two processes. With only two optimization parameters and the selected optimization method it was not feasible to use more processes.

The next step will be to test the parallel capabilities of Dakota with the pilot's use cases and see how the efficient execution of these problems can be handled.

3.5 Conclusions

Since the initial evaluation of Dakota showed that its capabilities satisfy all requirements of the pilots and the initial installation and usage tests of the framework have been carried out on the HLRS HPC-system, the plans to the end of the project are targeted towards a more efficient execution model of the framework on the HPC-systems at HLRS and PSNC and towards an increased usability for the pilots and with that for the whole GSS community.

To increase the efficiency when using the framework, it is planned to more closely evaluate Dakota's direct interface. By using the direct interface to reach a closer integration of Dakota and the targeted ABM simulation frameworks the communication between Dakota and the simulation application is carried out via the main memory and no longer via the file system. Additionally, the usage of the direct interface will allow for the integrated MPI execution of Dakota and the simulation application which could also significantly increase efficiency. To improve the usability of Dakota, the potential of its graphical user interface (GUI) with respect to the targeted pilot use cases will be evaluated. This evaluation should lead to the decision whether it is more reasonable and efficient to replicate the needed functionalities of the GUI in the CoeGSS portal or to pass the user from the portal onwards to the targeted HPC-system and directly use the GUI there for the setup and process steering of e.g. parameter studies or optimization runs.

4 Remote and Immersive Visualisation Systems

Within the project, the focus of the Visualization Task is to develop and to provide remote and immersive visualization services to consortium partners as well as to CoeGSS users. These services are continuously being integrated into the CoeGSS portal regarding the defined workflow as described in Deliverable 3.3 Section 2.2 (Figure 2). These services provide access to HPC as well as to sophisticated visualization resources integrated in a seamless manner in order to create “Immersive Analytics Environments” for huge statistical and multidimensional datasets.

Deliverable 3.1 reports on evaluation of software tools for remote and immersive visualisation. A software system was selected that was expected to meet the requirements recorded at the time of the report. The focus of Deliverable 3.2 was to report which user requirements regarding interfaces, systems and tools were collected and specified. Deliverable 3.3 reports on the requirements specified by users, in particular by the pilots. Implemented modules and plug-ins, which have been developed to meet the requirements of Pilot 1 (Health Habits) and Pilot 2 (Green Growth), are briefly introduced.

The aim of the past months was to complete the specification with regard to the requirements of access options for remote and immersive visualization. This section briefly describes the possibilities of accessing the user data and the corresponding visualization options.

4.1 The CoeGSS Visualisation Toolbox

As specified in deliverable D3.3 Section 2.2, based on the CoeGSS system architecture, the CoeGSS system workflow describes the data flow through the most relevant modules or processing steps (Figure 12).

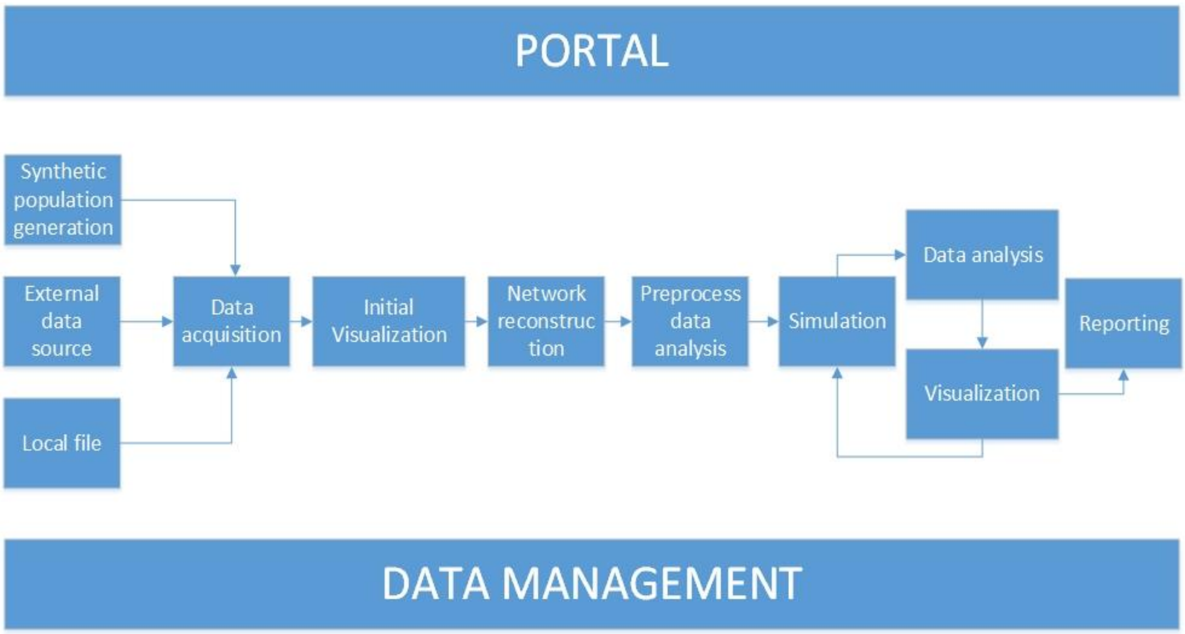


Figure 12: CoeGSS System Workflow.

Within the CoeGSS system workflow, the advanced visualisation is used to explore and investigate results from simulation runs as well as output from data analytics interactively. Due to the iterative process as well as the complexity and size of the data sets being visualised, it might become difficult to visualize these kind of data sets, usually located on HPC resources, with standard PCs or laptops used locally by the user.

For that reason, the CoeGSS portal offers multiple possibilities to access COVISE[7]/OpenCOVER[8], a software for advanced interactive scientific visualisation, which implements various modules with focus on reading, processing and visualisation data within the CoeGSS workflow (see also Deliverable 3.3). The user may run the software locally, in a remote session or by using a hybrid session, integrating local and remote resources depending on effort and resources needed to process and visualize given data sets.

To run a ***local session***, the software COVISE and OpenCOVER have to be installed on the local computer system as introduced in deliverable D3.6. The advantages of this solution are that the visualisation session can run on a local machine stand-alone without the need of internet access nor access privileges to any HPC resources. This is a typical configuration used by CAVE setups or similar visualisation installations for instance but also is useful to investigate the data on local workstations with proper processing and graphics performance. Necessarily, datasets or subsets have to be downloaded via file transfer to the local machine. Reading, processing and rendering the data sets must be done by the local machine.

For a ***remote session*** the user can invoke a Virtual Network Computing session (VNC) as introduced in deliverable D3.5. The VNC session makes use of dedicated login, pre- and post-processing nodes, which are part of the HPC infrastructure within the project. Available systems as well as hardware specifications on Hazel Hen at HLRS for instance are described in deliverable D3.5 chapter 4.3.2. An advantage of this option is, that processing as well as visualisation is running on the pre- and post-processing nodes within an HPC infrastructure, offering high processing and graphic performance and are also equipped with a considerable amount of memory. On the user side, only the access data and a VNC client are required, some of which are also freely available. On the other hand, good network bandwidth with low-latency is required.

A ***hybrid session*** can be invoked by running a distributed COVISE session partly on local computers and partly on remote systems. This feature is one of the outstanding possibilities of the software environment COVISE and has been developed further for many years and is used in industry and research. For example, a remote system with I/O and storage capabilities, which is used to read and process the data, and a local system with appropriate rendering capabilities, which is used to render the data interactively to the user's screen, can be interconnected within COVISE. The following paragraphs will give an example on how to setup and run such a hybrid session.

Setting up a distributed COVISE session requires access to all involved machines. The COVISE session is controlled by the user from the so called MapEditor on the local workstation (Figure 13).

---

7 https://www.hlrs.de/covise/

8 https://www.hlrs.de/solutions-services/service-portfolio/visualization/covise/opencover/

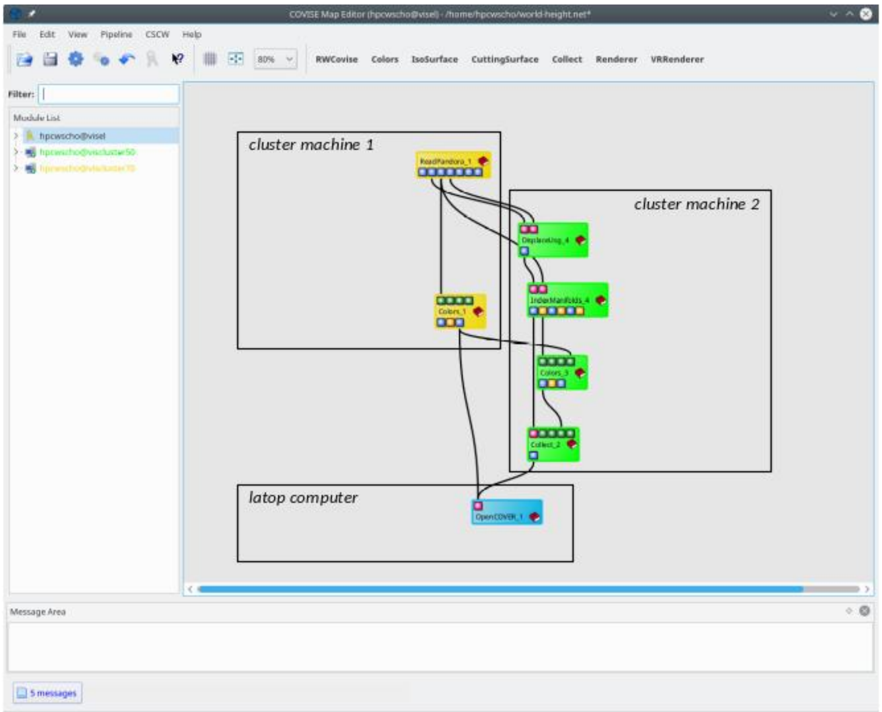


Figure 13: Setting up a distributed COVISE session.

In this example, three computers have been used to read, process and visualise a data set, which can no longer be visualized on the affected laptop stand-alone due to lack of memory. Cluster machine 1 (yellow), which in this example has a high network bandwidth, a high I/O performance and direct access to the data, is used to read in the HDF5 data set and only performs small processing tasks. The data is read into COVISE objects, which are then transferred as necessary to the computers involved via shared memory. Transferring COVISE shared memory objects is optimized and usually much faster than transferring and re-reading the raw data. Cluster machine 2 (green) has a considerable amount of memory as well as processing performance and has the task of processing the read data objects and preparing the data for visualization. In this example, the laptop computer (blue) waits for the data to be read, processed and being transferred to the laptop memory. The data so far is prepared accordingly for visualization, so that the laptop computer only receives the necessary data objects and only has to perform the rendering of the data on the local screen.

Figure 14 illustrates the principles used in the example, which run in the background for distributed working in COVISE.

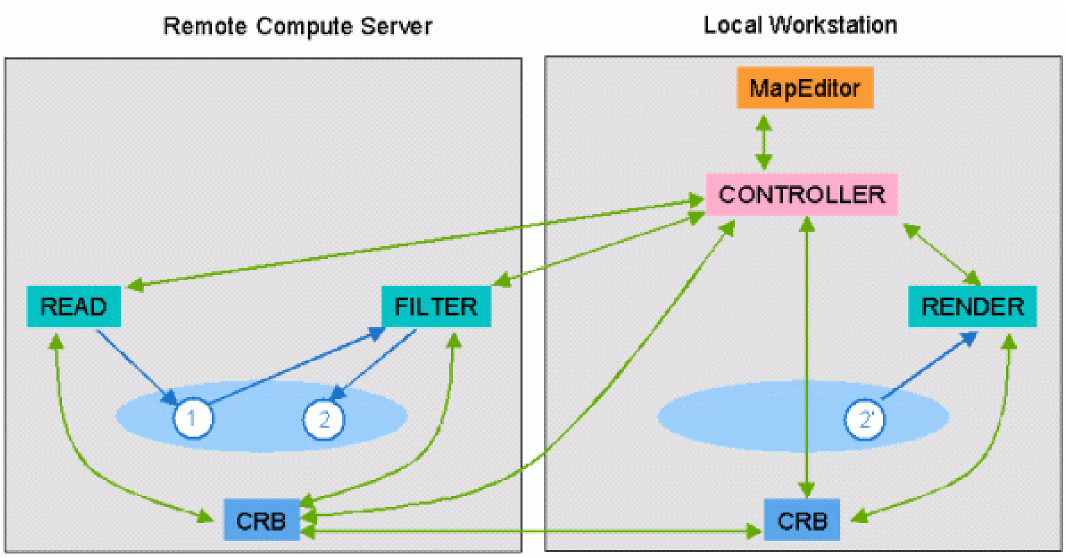


Figure 14: Remote and local visualization in a hybrid session

Simplified, the application consists of three modules: a module which reads the data (READ) into the memory, a module which extracts a special feature (FILTER) and a module, that displays the extracted data (RENDER). As the filter module consumes much CPU time and memory, it will be started on a remote compute server, as well as the Reader because the data to be read is located on the remote machine in this example. Just to illustrate the point, the remote compute could also be a pre- and post-processing node within a HPC system for instance. The first process started by COVISE is the Controller which in turn starts the user interface process MapEditor (Figure 13) and the data management process the COVISE Request Broker (CRB). As soon as another host is included in the session, a CRB is started on that computer. The read module and the filter module are started on the remote computer system and the renderer on the local workstation. The green arrows between the processes Controller, MapEditor, CRB and the modules indicate TCP sockets, the blue arrows indicate shared memory access.

When the module net-file is executed by the MapEditor controlled by the user, the Controller sends a start message to the remote read module. The read module reads in the data file and creates a COVISE data object (1) in shared memory and after processing tells the controller that the module has finished. The controller informs the filter module on the remote computer to start. The filter module asks its data management process (CRB) for the data object (1). The filter module now reads that data object, computes something and puts the data object (2) into shared memory. It then tells the controller that it has finished. The controller informs the renderer module to start. The renderer asks the CRB for object (2) and as this object is not available on the local workstations the CRB transfers it from the compute server into the shared memory of the local workstation (2'). Now the renderer can access this object and display the data.

As illustrated the advantage of a hybrid session is, that storage and processing capabilities for these potentially huge datasets for visualisation are still provided by HPC resources, while visualisation can be run locally on a graphics workstation or a compute cluster when using a Powerwall for example. This option enables high-quality rendering depending on local rendering capabilities, but still requires good bandwidth.

4.2 Conclusion

The three different scenarios, how to access remote and immersive visualization, are described in this chapter. All three options were tested with data sets coming from Pilot 1 (Health Habits) and Pilot 2 (Green Growth), taking full advantage of the corresponding environments. The necessary data interfaces regarding the connection to the data base and the CoeGSS web portal were identified and specified. The next step is to integrate these access options into the CoeGSS portal. Results of the implementation are expected to be reported in Deliverable 3.7.

5 Domain Specific Languages (DSLs)

Earlier deliverables D3.2 and D3.3 identified gaps in the functionality provided by state-of-the-art tools that are used for the development of Synthetic Information Systems (SISs), and proposed DSL and network reconstruction tools for addressing the gaps. This deliverable presents an evolved design, which is a result of further requirements that have been identified by the pilot projects.

We start by describing four DSLs for generating synthetic information systems in the context of GSS simulations. These are:

- **DSLH**: DSL for determining the structure of the simulation needed to answer a high-level question
- **DSLD**: DSL for data description and use
- **DSLP**: DSL for generating the synthetic population
- **DSLA**: DSL for generating an agent-based model

These DSLs correspond to the various stages of building an SIS for GSS.

An important aspect of GSS models is that the agents are related by multiple networks of relationships: geographic proximity (neighbourhood), but also social and economical connections (friendship, followers on social media, business relationships, etc.). Data about the location of households has been part of census collections since the beginning, and is available from the usual sources, e.g., Eurostat. Social data, on the other hand is much more volatile, and, while the advent of social media has made it more available, its collection is difficult and can lead to skewing results towards certain segments of the population. An alternative is to develop indicators of the influence an agent can have on another's behaviour, and to use the available data to estimate these indicators. Tools to achieve this "network reconstruction" have also been developed within Task 3.4, using the results of **DSLD** and **DSLP** and supporting the construction of ABMs with **DSLA**. These tools are presented in Section 5.5.

5.1 DSLH: DSL for determining the structure of the simulation needed to answer a high-level question

GSS aims to develop systems, theories, languages and tools for computer-aided policy making with potentially global implications. The aim of **DSLH** is to bridge the gap between the GSS problem description, given by the users in application-specific terms, and the low-level software layers comprising the SIS.

For example, GSS practitioners might want to assess

- the avoidability of a temperature increase of more than 2°C with respect to pre-industrial times
- the vulnerability of certain social groups to lung cancer
- the reachability of a state in which green cars are in the majority

DSLH formalises the high-level concepts of "avoidability", "vulnerability", "reachability", associating to each of these the type of simulations needed to assess it.

The DSL has the following structure:

- front-end: high-level "avoidability", "vulnerability", "reachability"
- back-end: lists of items needed to assemble the simulation, e.g.
    - the SIS that can be used to determine possible trajectories
    - a *harm* function, that measures damages, losses, etc. along a trajectory
    - a *vulnerability measure* that fulfils the monotonicity condition

DSLH is described in the publications (Ionescu, 2016), (Botta, Jansson, Ionescu, Christiansen, & Brady, Sequential Decision Problems, Dependent Types and Generic Solutions, 2017), and (Botta, Jansson, & Ionescu, Contributions to a Computational Theory of Policy Advice and Avoidability, 2017).

The software described in these publications is implemented[9] in the dependently-typed programming language Idris (Brady, 2017). A related implementation[10] in the Agda programming language[11]. A collection of application independent Idris libraries that grew out of the SeqDecProbs framework is IdrisLibs[12].

This DSL will not be further developed in CoeGSS, but will be used to guide the implementation of lower level software systems.

A first example application is outlined in the recent paper (Botta, Jansson, & Ionescu, The Impact of Uncertainty on Optimal Emission Policies, 2018).

5.2 DSLD: DSL for data description and use

The first step towards building an SIS consists in obtaining and processing data. The data describes (some of) the attributes of (some of) the agents that make up the synthetic population. This data is, in general, in "raw" form, containing redundant or irrelevant information. For example, the General Household Survey, 2006 from the UK Office for National Statistics (Office for National Statistics. Social and Vital Statistics Division 2009) contains a data file with 1541 columns, many of which are not relevant for most applications, and some containing data that is derived from other columns. **DSLD** will allow the users to describe the data required for the agents, describe the raw data, and manipulate the data at a high-level, in terms of agent attributes rather than, e.g., table-columns.

DSLD does *not* manipulate the data directly. Rather, it generates code that can be used to prepare and test the data in the CoeGSS HPC environments.

[9] https://github.com/nicolabotta/SeqDecProbs
[10] https://github.com/patrikja/SeqDecProb_Agda
[11] http://wiki.portal.chalmers.se/agda
[12] https://gitlab.pik-potsdam.de/botta/IdrisLibs

The DSL has the following structure:

- **Front-end** The user specifies the characteristics and their types (possible values) and relations between them (e.g., if age is < 10 years old, then education level is not University)
- **Back-end** The result is a generated C code for preparing and testing the data

The DSL is architected as a quoted DSL (QDSL) in Haskell, based on the work presented in (Najd, et al. 2016). Like embedded DSLs (EDSLs), QDSLs allow for borrowing certain aspects of the host language to use in the guest language, such as operator or conditional statements, which allows for fast prototyping.

Unlike EDSLs, QDSL programs are not expressed directly using expressions of the host language, but rather by using quotation, which provides a clear separation between the host language and the DSL. In addition to that, the QDSL infrastructure provides a simple, but powerful optimisation framework, which allows for using high-level constructs in the front-end language, that will then be eliminated during the compilation process, yielding efficient code. For example, all constructions related to sum types, which do not exist in the C programming language, can be eliminated.

To demonstrate how high-level features can be eliminated from the target code, let's consider this definition of a mapping of numeric values contained in a column of a data file into descriptive labels.

```
eduMapQ :: Mapping
eduMapQ =
  [[6, 8, 9]     |-> "NoData",
   [1]           |-> "HigherEd",
   [2]           |-> "OtherEd",
   [3]           |-> "NoEd"]
  `noDefCase` ()
```

The column describes the education level of an individual. If a row contains number 6, 8 or 9 in the column, this means that no data on the education level was collected. Values 1, 2 and 3 denote different education levels. Finally, any other value is illegal, which is specified using the `noDefCase` combinator.

Having defined the mapping, we can define a property that will operate on the labels given to the values in the column.

```
prop1 :: Maybe String -> Bool
prop1 (Just "NoData")   = False
prop1 (Just "HigherEd") = True
prop1 (Just "OtherEd")  = True
prop1 (Just "NoEd")     = True
prop1 Nothing           = False
```

The property checks whether there are no illegal values in the column by returning `False` for the argument `Nothing`, which denotes an illegal value. Furthermore, the property also checks whether there are no persons for which the data is missing, by returning `False` for the argument `Just "NoData"`.

Using our framework, the mapping and the property can be combined and compiled down to the following low-level C function:

```
unsigned int prop1(unsigned int x1)
{
    int x2;
    int x3;
    int x4;
    int x5;
    int x6;
    unsigned int v4;
    int x7;
    unsigned int v3;
    int x8;
    unsigned int v2;
    int x9;
    unsigned int v1;

    x2 = x1 == 9U;
    x3 = x1 == 8U;
    x4 = x1 == 6U;
    x5 = x3 || x4;
    x6 = x2 || x5;
    if (x6) {
        v4 = 0U;
    } else {
        x7 = x1 == 1U;
        if (x7) {
            v3 = 1U;
        } else {
            x8 = x1 == 2U;
            if (x8) {
                v2 = 1U;
            } else {
                x9 = x1 == 3U;
                if (x9) {
                    v1 = 1U;
                } else {
                    v1 = 0U;
                }
                v2 = v1;
            }
```

```
                v3 = v2;
            }
            v4 = v3;
        }
        return v4;
}
```

The function is difficult to read by a human, but can be compiled using a modern C compiler to yield efficient code. Please note that both the string labels and the `Maybe` datatype have been eliminated from the resulting code–the only values used are machine integers.

Thus, we were able to use high-level code on the programmer's side, and at the same time generate efficient low-level code suitable for high-performance compilation.

As mentioned earlier, currently the DSL is a quoted DSL embedded in Haskell. Ultimately, concrete surface syntax might be created for DSLD to make it easier for non-expert programmers to use it, but the underlying optimisation infrastructure provided by the QDSL approach is likely to be retained.

The current plans for the development are as follows:

- Demo implementation will be ready in mid-July
- Tests with Health Habits will happen two weeks after that, with Green Growth four weeks after

5.3 DSLP: DSL for generating the synthetic population

The data available will define, in most cases, only a small number of the agents necessary for the SIS. The next step is therefore to build a synthetic population from which all agents can be defined. This step requires the results of **DSLD**, namely the data description in terms of high-level agent attributes, together with a micro-sample conforming to this description, and statistical information about the population, usually in the form of marginal distributions of attribute values. **DSLP** will then generate *HPC-ready* code for extending the micro-sample to a synthetic population.

The DSL has the following structure:

- **Front-end** The user specifies the input data description using DSLD, and the procedure for generating the synthetic population using a set of primitive operations, like IPF or sampling
- **Back-end** The result is a generation procedure implemented in C

Currently, the following computational kernels are available:

- Efficient IPF kernel based on the PBLAS[13] API
- Haskell implementation created for validation purposes

Similarly to **DSLD**, **DSLP** is implemented as a Haskell-based QDSL.

[13] http://www.netlib.org/utk/papers/scalapack/node9.html (Office for National Statistics. Social and Vital Statistics Division 2009)

These are not currently integrated with other developments in the work package.

The current plans for the development are as follows:

- Preliminary integration and demo implementation until end of June
- Integration with the Portal until end of August
- A version based on interval analysis until end of September

5.4 DSLA: DSL for generating an agent-based model

This DSL constructs the final component of an SIS: the agent-based model. **DSLA** provides the structure for implementing the description and dynamics of the agents, together with those of the network of relationships/interactions between them, and of the environment in which they interact. **DSLA** uses the data description of **DSLD** and the synthetic populations built by **DSLP**.

The DSL has the following structure:

- **Front-end** The user provides characteristics and actions/dynamics of agents, relationship and interaction networks, characteristics and dynamics of spatial layer, description of computational environment
- **Back-end** The result is code for ABM initialisation and parallel simulation, including partitioning on the basis of the spatial network information

DSLA will be described in detail in D3.8. Elements of **DSLA** are a direct result of the *Green Growth* pilot and support the current implementation of MOTMO.

The following developments are planned:

- Complete DSLA and make a first version of source code available on a public repository (e.g., GitHub) by mid-July
- Improve the interface between DSLD, DSLP, and the tool for similarity networks on the formalisation of high-level policy concepts.
- Make the final version of DSLA source code available by end September
- Document DSLA by end September

5.5 Network reconstruction component

In the previous Deliverable 3.3 it was proposed to study the applicability of entropy-based methods for the reconstruction of social network from partial information. During our analysis, we realized that the entropy-based approach was not feasible for the problem at hand. Indeed, data about actual friendship are hard to find and in most of the case are inferred from other data (Renaud Lambiotte 2008, Barthélemy 2011), so it is even difficult to state which is the partial information we have access to. Moreover, since agents are equipped with an array of data of different nature (categorical data, numerical values, geographical coordinates…) it is even hard to decide which is the correct information determining the interactions and how it is related to the social influence network.

We then decided to move to a different framework and infer the structure of the network of social influences from all data that define the agents, or otherwise stated, it is the synthetic population itself that produce the related influence network.

For most of the social simulations, the main idea is that friendship relations influence the behaviours of single agents. In the present approach, we assume that similar agents influence each other, thus the more attributes they share, the higher the tendency to have an impact on the other habits. Anyway, agents should have even the possibility to physically interact. In other words, the distance should damp the effects of sharing attribute: even if two agents share most of their attributes, it is unlikely that they may influence each other if they do not have the possibility to meet.

In summary, in the new framework, the problem is twofold: on the one hand, we assume that the influence among agents is driven by how much of their attributes they share; on the other hand, the contribution is reduced by the distance among them.

Nevertheless, there is another non-trivial issue. The agents of CoeGSS pilots come from the CoeGSS generation of a synthetic population and synthetic agents are defined by different attributes that can be of different types: integers, floats or categorical data. Consequently, the similarity network needs to handle data of different kind and return a similarity value.

In the literature, different proposals for similarity networks can be found, mostly basing their definition on the target of their analysis and thus lacking a clear generalisation; moreover, most of the studies focus on just one data type, disregarding the possibility of heterogeneous datasets, i.e. the possibility of handling different kinds of data at the same time. In the influence network tool, we follow the definition of Lin (Lin, 1998) since it has several positive features: it is general and unbiased, being based on Information Theory, and permits to analyse different data types. Moreover, it permits to consider possible dependencies among agent attributes, which is the usual case for real data. In concrete, if **A** and **B** are the two vectors of attributes defining a couple of agents, Lin similarity weighs the information carried by the commonalities of **A** and **B** (i.e. the attributes that **A** and **B** share) over the information carried singularly by **A** and **B**:

$$sim(\mathbf{A},\mathbf{B}) = \frac{2\log P((\mathbf{common}(\mathbf{A},\mathbf{B}))}{\log P(\mathbf{A}) + \log P(\mathbf{B})}$$

The formula compares the self-information (Cover & Thomas, 2006) of the single agents with their commonalities one. If their commonalities describe both A and B, the similarity is 1, otherwise the greatest value are provided if their commonalities are infrequent, i.e. if the description of the attributes shared by both A and B is rare. In order to infer influence relations, we dampen the similarity contribution with a function of the distance, mimicking the geographical dependence of the spatial distribution of phone calls and other media (Renaud Lambiotte 2008, Barthélemy 2011). In our tool, both exponential and power law dampenings are examined. In formula, this reads as:

$$a_{\mathbf{AB}} = f(\mathbf{d_{AB}}) \cdot sim(\mathbf{A},\mathbf{B}),$$

Where $\mathbf{d_{AB}}$ is the geographical distance between **A** and **B** and sim is the abovementioned Lin similarity.

It is standard that similarity measures range from 0 to 1; even after the introduction of the geographical dampening factor, the interval for the final network weight is unchanged. Thus, in the resulting final network, each of the weights falls in the interval [0,1], where 0 means no influence and 1 maximum influence.

In the construction of the synthetic population, some attributes are naturally dependent: for instance, a child (i.e. a low age agent) cannot have a high level of education or a high income. This forces us to consider all possible dependencies: for instance, the probability P(**A**) in the similarity definition is the frequency of agents showing all the attributes of **A** at the same time. Otherwise stated, considering the frequency of each of the attributes on its own would have result in disregarding possible dependencies in the attribute definition. A similar approach is used for commonalities: we have to calculate the frequency of agents whose attributes are between those defining **A** and the ones defining **B**.

If the number of agents is quite high, this procedure results in a high number of calculation. Nevertheless, calculation times can be substantially reduced by defining a training set of a small fraction of the original population in order to calculate the similarities. The most effective training set dimensions, in terms of both accuracy and calculation times, are currently under study.

We observed that the generated networks show a strong hierarchical structure, organised in dense clusters: if we select links over a certain threshold, we observed clusters that are contained in the analogous for the network cut at a lower threshold. The interesting point is that this feature is typical for social networks and it is correctly captured by our similarity network tool.

In the current realisation, the input data is a synthetic population (created in CoeGSS) as a HDF5 file. Synthetic population data need to be pre-processed to be suitable for the influence network calculation. The user may decide to disregard some of the attributes of the synthetic population tool, considering the information they carry as unnecessary or even misleading for the simulation. Since the similarity network approach is completely general, this selection is going to be part of the pre-processing of the network reconstruction tool: once data are suitably selected, the algorithm will be fed by the resulting dataset and nothing changes.

As already mentioned above, Lin similarity is able to handle different kind of data. Nevertheless, different kinds of data are considered differently: the commonality of two agents for a ordinal value attribute is defined as the set of all entries between the value of the two agents, while for categorical data is considered as the whole set (since there in not any general way to order categorical values). Thus, in the preprocessing an extra vector describing the category of the attribute is added.

The code is implemented in Python, making use of mpi4py[14] and h5py[15] packages, i.e. Python wrappers respectively for MPI (Message Passing Interface) and the (parallelised) management of HDF5 files. The algorithm has just two parameters: the length scale at which the damping factor is equal to ½ and a categorical entry controlling the damping function (exponential or power law, at the moment).

Taken a couple of nodes, the algorithm first determines if the two nodes are close enough to provide a non-negligible contribution, that is if the geographical distance scaling function is greater than a given threshold (we set the threshold for negligible contribution to 10^{-6}). Secondly, in order to discount the effect of possible dependencies in the attributes distribution, we select, in the training set, the number of agents sharing the attributes that **A** and **B** share. As it is shown in Figure 15, the following plot, making use of a training set of the 10% of the original population reduce dramatically the calculation times. In the Figure, the cyan line represents the calculation time for a training set of the 10% of the original population, whereas the dashed blue one for the complete system.

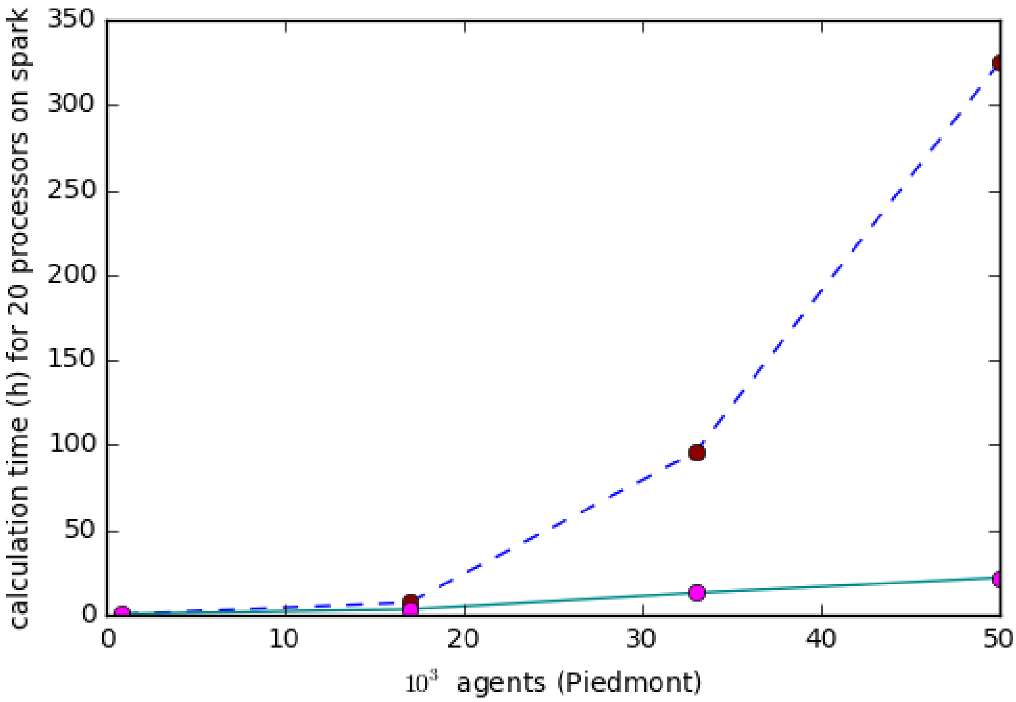


Figure 15: Calculation time for network reconstruction.

The output of the similarity network algorithm is saved in a different HDF5 file. First, an original HDF5 group is created. Then an HDF5 dataset contains the global similarity, i.e. the one obtained from the weighted contribution of all similarities per attribute.

Due to the nature of the influence network and its peculiar behaviour, the complexity of the algorithm goes as $O(N^3)$, where N is the number of nodes. An $O(N^2)$ comes from calculating a

[14] http://mpi4py.readthedocs.io/en/stable/index.html
[15] http://docs.h5py.org/en/latest/index.html

contribution for every couple of nodes, the remaining O(N) since, in each single calculation, due to possible dependencies of attributes, we have to search for all agents sharing the same characteristics. Thus, the resulting algorithm complexity is $O(N^3)$. The following plot represents calculation times for some subsets of the synthetic population of Piedmont. Nevertheless, the geographical threshold and the training set reduce the calculation times substantially.

In the next months we are going to consider the most effective dimension of the training set for the accuracy and the calculation times.

5.6 Conclusions

We have described the current status in the development of DSLs and network reconstruction tools needed for the generation of GSS-specific synthetic information systems.

The developments reported here follow those carried out in the CoeGSS pilots. For example, DSLA is the result of attempts to implement multi-agent models first using special purpose code, then existing frameworks such as Pandora. Finally, the lessons learned in these attempts have led to the implementation of a CoeGSS framework for multi-agent modelling, tailored to the specific properties of GSS models (e.g., multiple networks of relationship between agents).

The most important remaining task is the integration of the tools presented here within the CoeGSS portal.

Task 3.4 is a natural meeting point of the GSS and HPC communities, a fact underscored by the animated discussions carried out within the CoeGSS Plenary meeting in May 2018. The outcome of these discussions has heavily influenced our presentation here, we gratefully acknowledge this contribution.

6 Representing Uncertainty in Modelling and Computation

The main goal of task 3.2 is to ensure the validity and correctness of the software components developed for GSS simulations. As we declared in D3.2 within Task 3.2 we are concentrating on the implementation of interval-based methods to validate algorithms developed for CoeGSS and perform tests with it to estimate its precision.

Interval analysis has three uses:

1. to prevent round-off errors in computations involving floating-point numbers. This is the original motivation for its development. In CoeGSS, this is very important because we do not have the possibility to test our models empirically, therefore need alternative ways to ensure correctness of implementation. Otherwise, it will be difficult to tell whether the results we see come from our assumptions, or from our errors.
2. to account for uncertainty in the data. In many cases, input data is expressed in terms of intervals, and choosing a point value in these intervals is likely to lead to incorrect results. Moreover, interval methods can show whether and to what extent such uncertainties propagate in the model (e.g., whether the intervals, and thus the uncertainty associated with the values, increase rapidly).
3. as a source of new numerical methods, in particular for root finding, parameter fitting, and global optimisation. For example, a global optimisation method will result in a list of intervals, which is guaranteed to contain all the optima. These methods require a lot more computational power than the traditional algorithms, but can at least be used in order to test the results of the latter.

As a direct CoeGSS application, we are currently testing the IPF implementation of DSLP (see Section 5.3 in this deliverable) by implementing an interval analysis based method.

6.1 Method

Interval analysis is an approach to perform numerical computations with simultaneous calculation of error bounds. It was introduced by Moore in 1966 (Moore, 1966) and has been a field of active research since then. The reasons why we chose this method are the following:

1. Finite digits floating point arithmetic is not reliable. For example, it is easy to prove that
$$x_0 = 1, x_1 = \frac{1}{3}, \ldots, x_{n+1} = \frac{13}{3}\, x_n - \frac{4}{3} x_{n-1}$$
converges to 0 for n going to infinity. With finite digits, however, calculation implies that it approaches infinity.
2. Some of the most important tools in function analysis are related to intervals (Taylors theorem).
3. Intervals are best suitable to represent data from measurements including errors.
4. In control theory parameters are more natural to be represented by intervals instead of exact values (see Chapter 5).

Although there are developing libraries in C and Fortran for interval arithmetic, the use of a functional programming language like Haskell (Rump, et al. 2016) has an advantage above imperative programming languages. As Haskell is closer to the mathematical specification of an algorithm in the design phase and due to its referential transparency, it makes reasoning about its correctness easier.

6.2 State of Work

6.2.1 Implementation of Interval data type

The first step towards a library for interval arithmetic was the introduction of a data type Interval. An interval is represented by two Doubles, which allows all kinds of closed intervals and, due to the existence and standards for handling of constants for $-\infty$ and ∞ in the IEEE 754-2008 Standard for Floating-Point Arithmetic (Deming & Stephan, 1940), also of intervals of type $(-\infty, a]$, $[a, \infty)$, and $(-\infty, \infty)$. The empty interval will be represented by $[NaN, NaN]$ which allows to use the exception handling of floating point data types ensured by IEEE 754-2008 Standard for arithmetic of intervals.

To characterise intervals there was implemented a set of properties, including the predicates to perform the division in the right way: positive, negative, mixed and zero.

To ensure that the exact value of a function is always contained in the result of its evaluation with intervals it is important to make sure that the rounding mode is set appropriately, i.e. the downwards mode is used when calculating lower bounds and upwards mode is used when calculating the upper bounds of the intervals.

To avoid a frequent change of rounding modes (which is highly inefficient) we implemented the arithmetic operations with a fixed rounding mode using the method of (Rump, et al. 2016).

6.2.2 Interval arithmetic

The axioms for interval arithmetic introduced in (Moore, 1966) require for the interval extension **f** of any real valued function f with n real variables, that given intervals $I_1, .. I_n$ that

$$\forall x_1, \dots, x_n \in I_1, \dots I_n \text{ holds } f(x_1, \dots, x_n) \in \mathbf{f}\ (I_1, \dots I_n)$$

The challenge here is to find implementations for **f,** where the resulting intervals are as tight as possible.

We firstly implemented two versions (with rounding up and down) for every floating point operation: $(\odot_d, \odot_u)\ with \odot \in \{+, -, *, \div\}$ and built upon this we implemented addition, subtraction and multiplication on intervals such that for $\circ \in \{+, -, *\}$ we have:

$$I_1 \circ I_2 = \{x \circ y | x \in I_1,\ y \in I_2\}$$

In the implementation we followed the standards of the IEEE P-1788 working group interval arithmetic. The exception handling of IEEE754 allows to handle the cases where one or both of the operands are the empty or a half-open interval efficiently (i.e. without additional case analysis).

For the division the situation is more complicated. As long as the divisor does not contain zero, division is defined by

$$[a_1, a_2] \div [b_1, b_2] = [c_1, c_2] \quad where$$

$c_1 = \min\{(a_1 \div_d b_1), (a_1 \div_d b_2)(a_2 \div_d b_1)(a_2 \div_d b_2)\}$ and
$c_2 = \max\{(a_1 \div_u b_1), (a_1 \div_u b_2)(a_2 \div_u b_1)(a_2 \div_u b_2)\}$.

Unfortunately, the result of a division by a mixed interval is not always an interval, but to in some cases a pair of intervals. Table 3 shows all six cases for the definition of interval division for the cases when $[b_1, b_2]$ contains zero.

Table 3: List of cases for division by intervals containing zero

case	A = $[a_1, a_2]$	B = $[b_1, b_2]$	B′	A÷ B'	A $\div B$
1	$0 \in A$	$0 \in B$			$(-\infty, \infty)$
2	$0 \notin A$	$[0,0]$			$\emptyset$
3	$a_2 < 0$	$b_1 < b_2 = 0$	$[b_1, -\varepsilon]$	$\left[\frac{a_2}{b_1}, \frac{a_1}{-\varepsilon}\right]$	$\left[\frac{a_2}{b_1}, \infty\right)$
4	$a_2 < 0$	$b_1 < 0 < b_2$	$[b_1, -\varepsilon] \cup [\varepsilon, b_2]$	$\left[\frac{a_2}{b_1}, \frac{a_1}{-\varepsilon}\right] \cup \left[\frac{a_1}{\varepsilon}, \frac{a_2}{b_2}\right]$	$\left(-\infty, \frac{a_2}{b_2}\right] \cup \left[\frac{a_2}{b_1}, \infty\right)$
5	$a_2 < 0$	$0 = b_1 < b_2$	$[\varepsilon, b_2]$	$\left[\frac{a_1}{\varepsilon}, \frac{a_2}{b_2}\right]$	$\left(-\infty, \frac{a_2}{b_2}\right]$
6	$0 < a_1$	$b_1 < b_2 = 0$	$[b_1, -\varepsilon]$	$\left[\frac{a_2}{-\varepsilon}, \frac{a_1}{b_1}\right]$	$\left(-\infty, \frac{a_1}{b_1}\right]$
7	$0 < a_1$	$b_1 < 0 < b_2$	$[b_1, -\varepsilon] \cup [\varepsilon, b_2]$	$\left[\frac{a_2}{-\varepsilon}, \frac{a_1}{b_1}\right] \cup \left[\frac{a_1}{b_2}, \frac{a_2}{\varepsilon}\right]$	$\left(-\infty, \frac{a_1}{b_1}\right] \cup \left[\frac{a_1}{b_2}, \infty\right)$
8	$0 < a_1$	$0 = b_1 < b_2$	$[\varepsilon, b_2]$	$\left[\frac{a_1}{b_2}, \frac{a_2}{\varepsilon}\right]$	$\left[\frac{a_1}{b_2}, \infty\right)$

With the implemented arithmetic functions and suitable implementations for abs, negate and signum we made intervals an instance of the Haskell Type Num (this is the type class for numerical data types, that is usually used for numerical functions). All implementations are contained in the module IntervalArithmetic.

6.2.3 Interval lists

Since the set of intervals is not closed under the arithmetic operations (division may yield a pair of intervals instead of an interval) we defined a new data type *IntervalList* that has lists of intervals as elements. For these type we implemented the four basic arithmetic operations (*module IntervalLists*) , such that they fulfil an analogue of Moore's axiom.

The data type *IntervalList* is closed under all arithmetic operations. Working with this data type instead of the interval type has the additional advantage to provide a more flexible setting for optimisation algorithms, as we now can define the outcome of a refinement operation (e.g. for zero finding of a function) to be of data type *IntervalList.*

The data type *IntervalList* gives rise to partial order: Let A, B be lists of intervals, then we can say A is smaller than B w.r.t. this ordering if every element x that is covered by some interval from the list A is also covered by some interval from B. Given a set of lists of intervals we call the minimum in this ordering a *clean list of intervals*.

To reduce computational costs when calculating with lists of intervals we want to represent results by shortest lists with tightest intervals. It turns out that this intuition is exactly covered by the requirement to have a clean list of intervals as output. To achieve this we implemented an operation- *cleanList*- for finding this minimum. It is contained in the *cleanList module.*

It could be proven that *cleanList* has nice algebraic properties, i.e. it is a monad. A publication with the details about the *IntervalList* calculus and the *cleanList* monad is in preparation.

6.2.4 Validation of Iterative Proportional Fitting (IPF)

The first algorithm we wanted to evaluate with interval analysis is the IPF algorithm that is used to generate synthetic populations see Deliverable 3.3, page 36.

We implemented a generic version of the two-dimensional version IPF that can handle floats, doubles or intervals as inputs.

The IPF algorithm is a procedure to find a least square adjustment of a frequency table for given marginals. Its convergence proof was given in (Deming & Stephan, 1940).

The input is a frequency table (micro sample) i.e. a matrix with r rows and c columns and two vectors of length r and c of real numbers that represent the marginal for the rows and the columns. In an iterative procedure the rows and columns are adapted in turn such that after several steps the sum of the columns is close enough to the corresponding entry in the marginal vector.

The convergence proof in (Deming & Stephan, 1940) is only valid for real numbers. Thus, the first question was to find indicators for convergence for finite digit numbers. The second question we wanted to answer was which number of iterations is necessary to reach a certain accuracy. The third question was to find out how accuracy depends on the size of inputs, the dimension of the matrix and the number of iterations.

To find the answer to these questions we implemented an interval version of IPF. The inputs are marginals $(mcol)$ and $(mrow)$ that are vectors of Doubles and a matrix (representing the microsample) with singleton intervals $[l_{i,j}^{(0)}, u_{i,j}^{(0)}]$ with $l_{i,j}^{(0)} = u_{i,j}^{(0)}$ as entries. Each iteration step adapts these entries to intervals that contain with certainty the "real-valued" entry that would have been computed in a pen and paper calculation.

To evaluate the algorithm we performed calculations with several test inputs. We created for the dimensions 2, 5, 10 for r and c twenty random test samples and observed the results after several steps of iterations.

6.2.5 Results

The first observation we made was that the algorithm showed a similar behaviour on all test examples. After six or seven iterations the relative errors of the column sums dropped below 10^{-6}. As a measure for the accuracy we used relative bounded errors for every column defined as

$$le_j^{(k)} = 1 - \frac{1}{mcol_j}\sum_{i=1}^{r} l_{i,j}^{(k)} \text{ and } ue_j^{(k)} = 1 - \frac{1}{mcol_j}\sum_{i=1}^{r} u_{i,j}^{(k)} ,$$

where $l_{i,j}^{(k)}$ and $u_{i,j}^{(k)}$ are the lower and upper bound of the interval valued matrix entries at position (i,j) after the k-th iteration of IPF and $mcol_j$ is the j-th entry in the column marginal.

For one test sample with dimension with ten rows and columns the behaviour of the relative bounded error is depicted in the graphics below. For every column j we used a particular color to show le_j and ue_j. The scale of the y-axis is logarithmic.

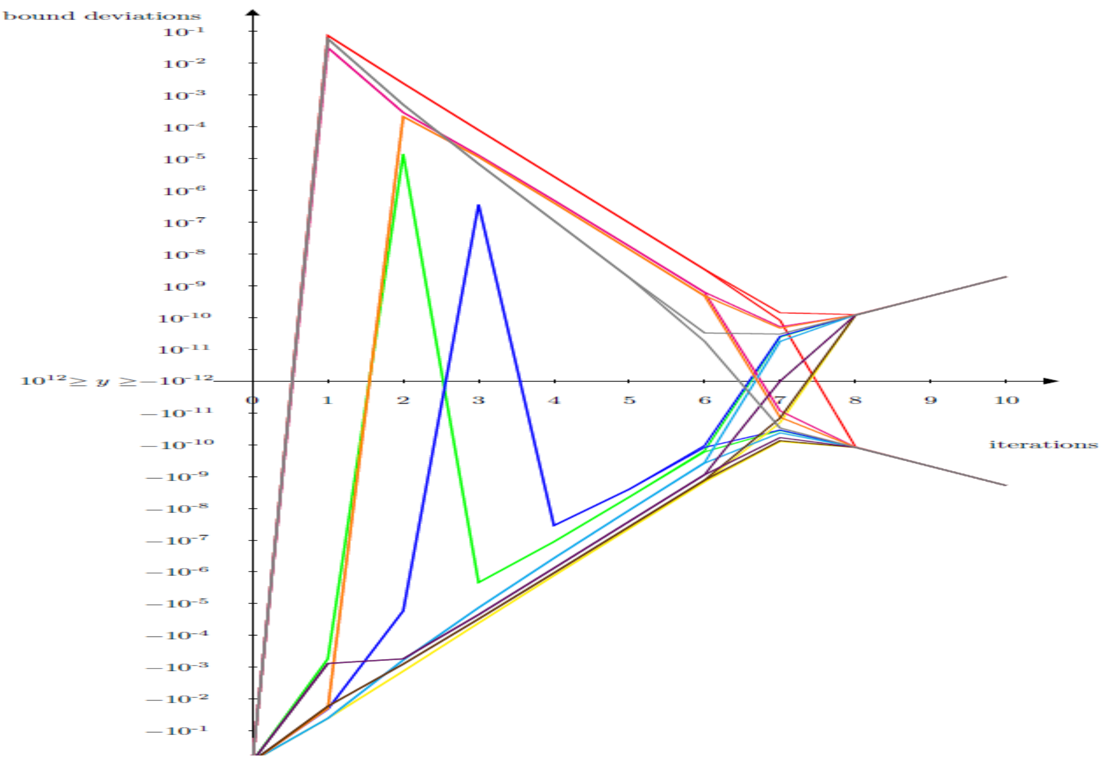


Figure 16: Relative bound deviations dependent of iteration number.

At the start for all columns j the values $le_j^{(0)}$and $ue_j^{(0)}$ coincide and are negative as the entries in the matrix are usually much smaller than the marginal (because a micro sample consists of fewer entities). For every column j with a number of iterations the sum of lower and upper bounds of entries in the matrix converge to $mcol_j$. At some point due to rounding errors the widths of the intervals $[l_{i,j}^{(k)} , u_{i,j}^{(k)}]$ increase and consequently $le_j^{(k)}$ and $ue_j^{(k)}$ diverge, i.e. move into opposite directions, such that two lines with the same colour are visible. We don't know yet if this effect can be avoided by another implementation of the error calculation or IPF procedure (remember that calculation with finite digit number is not associative!).

6.3 Conclusion

The next months we will finish the analysis of the n-dimensional IPF algorithm, which will help the pilots to assess the uncertainty of the methods used to create synthetic populations for agents with n distinct properties that is due to numerical errors. In a similar manner we will also validate the network reconstruction algorithm.

Beside the validations (and with it implementations of interval versions) of algorithms used by the pilots as a result of task 3.2 there will be a Haskell library of basic mathematical functions like square root, exp, log and polynomials that can be used by others who want to validate their algorithms in a similar way we did. Built on the Idris implementation of the divide and conquer algorithm scheme described in D3.3 this library will also contain an interval version of the Newton method for finding zeros.

7 Hardware and software co-design

Besides delivering different individual GSS software packages for fulfilling steps in the CoeGSS workflow, our project should take care of composing a single solid toolchain from those diverse packages. As an important part of this task, we must define an efficient data exchange format and implement it for major tools in the CoeGSS workflow. In Deliverable 4.2, the authors made initial steps in this direction by specifying requirements and guidelines to the HDF5 file structure for data exchange between CoeGSS tools. In this chapter, we present recent progress on specification and support of a unified HPC compliant HDF5 file structure for CoeGSS tools – one of the key subtasks in the frame of our co-design activities.

7.1 Structure of the HDF5 files for CoeGSS tools

HDF5 is a data model, library, and file format widely used by the scientific and engineering communities for storing and managing large amounts of data (HDF Group, 2018). It is designed for flexible high performant I/O. HDF5 is portable and extensible and, thus, easily sharable. It supports heterogeneous data and allows to keep metadata with data in the same file. HDF5 seamlessly integrates with application written on all major programming languages of HPC and GSS communities by official interfacing with C/C++ and Fortran APIs as well as by providing third-party bindings to other languages (including Python and R). All these reasons motivated us to choose HDF5 as a default file format for data exchange between CoeGSS tools.

7.1.1 Data structure

Figure 17 illustrates structure of the HDF5 files which we designed for data exchange between CoeGSS tools.

The top level **"/"-group** of the HDF5 file is used as a container for synthetic (or real) datasets, user-defined types, and global attributes that convey general information. If file is a product of ABMS simulation, the set of root attributes should include the following items which represent some common properties of the simulation process:

- `numSteps` – It defines number of simulation steps. In case of applications that intend to prepare simulation input (build synthetic populations and synthetic networks), its value is usually equal to 1.
- `serializerResolution` – It defines number of steps between serialization call during simulation. Its value is a positive integer.
- `size` – It defines raster dimensions. This attribute holds two integers of type `H5T_NATIVE_HSIZE` for x and y dimensions.

Besides the attributes, "/"-group contains the following data components:

- step groups – These groups hold data about agents, social networks, and rasters on each particular simulation step. File has $numSteps/serializerResolution$ step groups. Each step group is named by the serial number of simulation step it represents (starting from 1).

- `types` group – This group contains datatype definitions (of type `H5T_COMPOUND`) for agents stored in the given HDF5 file. In case of simulation output files, we assume the agent types do not change during the whole multi-step simulation process, thus we can store all definitions of agent types in `types` group at the top level "/"-group. The datatypes are committed, which makes them reusable in different datasets of the file. It helps to avoid duplicated definitions of the agent types. Datatype definitions inside `types` group have unique names.

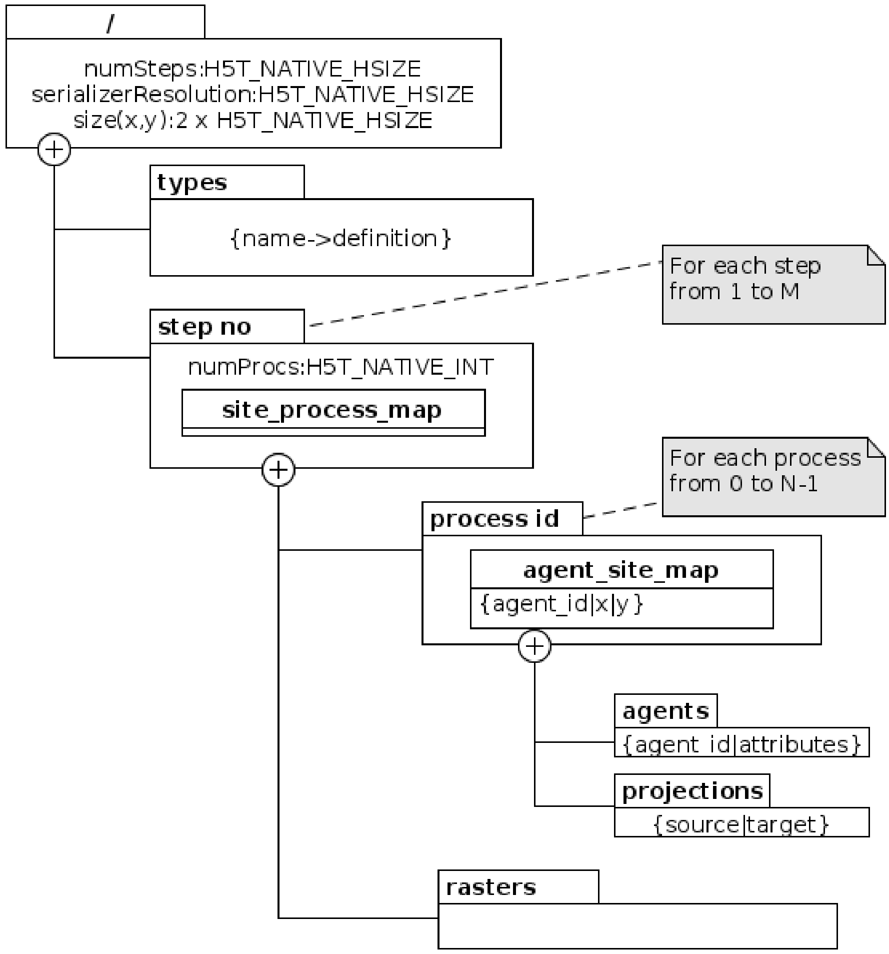


Figure 17: Structure of HDF5 files for CoeGSS tools

Each **step group** stores the following data components:

- process groups –These groups hold data for each specific process. They are named by MPI process IDs used in the simulation.
- `rasters` group – This group contains datasets for all rasters defined in the framework by the modeller. Each dataset is a table with N rows, where N is a number of raster points populated by agents. Usually, $N \ll X \times Y$, where X and Y denote dimensions of the raster (the maximum latitude and longitude of the sites). Types of raster values determine the type of items in `rasters` group datasets.
- `site_process_map` dataset – This dataset contains information about assignment of sites to processes. It is a N-dimensional table of type `H5T_NATIVE_INT` where each cell

corresponds to the site location (row number and column number as *x* and *y* coordinates) and contains ID of the process containing it.

In addition, step groups contain attribute `numProcs` of type `H5T_NATIVE_INT`. This attribute defines the number of processes used in the simulation.

Each **process group** stores the following entries:

- `agents` group – It contains datasets (of type `H5T_COMPOUND`) for all agents stored in the given HDF5 file. Each dataset is a compound table with column names corresponding to the agent attributes (fields). The rows contain information about agents. It means that we have just one dataset per agent type.
- `projections` group – It contains datasets (of type `H5T_COMPOUND`) for the social graphs. Those datasets hold edge lists in tabular format with 2 columns named `source` (of type `H5T_NATIVE_ULONG`) and `target` (of type `H5T_NATIVE_ULONG`) corresponding to source and target agent IDs of the link in social graph.
- `agent_site_map` dataset – This dataset contains information about assignment of agents to sites. It is a table with columns: `id` (of type `H5T_NATIVE_ULONG`), `x` (of type `H5T_NATIVE_INT`), and `y` (of type `H5T_NATIVE_INT`). Column `id` corresponds to the agent ID, whereas `x` and `y` corresponds to spatial coordinates of the site.

7.1.2 Implementation of the HDF5 extension

In order to facilitate support of the above mentioned HDF5 file structure by CoeGSS tools, we designed and implemented HDF5 extension library for C++ users. This extension provides simple API for reading and writing different objects of the data model – agents, rasters, projections (social networks), simulation setup, etc – from/to HDF5 files of the given structure.

Our HDF5 extension has been implemented as a C++ library and contains the following components:

- **Agent reader** – This class reads agent attributes from the step group with the given number. By default, it loads data for the last stored simulation step. If this operation succeeds a proper set of agents is created. Otherwise, if the given HDF5 has a wrong structure or data is not found – C++ exception is thrown. Created agents are properly located in the simulation environment by reading and setting their location coordinates stored in agent-site map dataset found in the HDF5 file. Also, all relations between agents are read and properly restored.
- **Site reader** – This component reads raster datasets and site-process map dataset using the latter to properly allocate created sites. Namely, it creates the sites in the MPI processes with the IDs defined by site-process map.
- **HDF5 extension reader** – It is a joint class containing both (agent and site) reader components. This component allows to read the whole data about synthetic population, synthetic network, and rasters from the given well formed HDF5 file at once.

- **Agent writer** – It is a class responsible for writing information about agents into a well formed HDF5 file. This component stores a proper structure for further simulation steps and/or postprocessing. Apart from storing agent attributes (`agents` groups), relationships between agents (`projections` groups), and data about agent location (`agent_site_map` dataset), it creates a table with unique HDF5 datatypes corresponding to all agent types (`types` group). Once the given agent datatype is found in this table, it is treated as existing without overwriting.
- **Site writer** – This class is responsible for writing information about sites. It stores site attributes (`rasters` group) and mapping of sites on MPI processes (`site_process_map` dataset).
- **HDF5 extension writer** – It is a joint class containing both (agent and site) writer components which serves for writing the whole data about synthetic population, synthetic network, and rasters to the given well formed HDF5 file at once.

In order to reach higher I/O performance, we equipped our HDF5 extension with support of different integrated HDF5-features for time and storage space optimizations. In particular, the writing components of the HDF5 extension allow to compress written data as chunked and compressed datasets inside the given HDF5 file by specifying adequate values for compression level and data chunk size. Since data chunks and compression levels are handled by the native HDF5 C library, they are completely transparent for the reading components of the HDF5 extension.

For the testing and benchmarking purposes, we implemented a HDF5 plugin for the Amos ABMS framework based on our HDF5 extension library. Our tests helped us to reveal a data compression bug in the native HDF5 C library. More precisely, the tests show that native HDF5 library fails to store datasets if the number of items to store is not divisible by the chunk size. This bug is confirmed by HDF group and must be fixed in the next releases of the HDF5 C library (HDF Group, n.d.).

7.2 Conclusions

In the next months, we plan to finalize work on the HDF5 extension library. Besides focusing on thorough tests and benchmarks for the HDF5 extension, we intend to look at algorithmic ways to improve I/O performance. In particular, we plan to analyse formats for storing evolution of the synthetic population that require less physical space than naïve serialization of the whole population (Tatara, Collier, Ozik, & Macal, 2017).

Apart from work related to efficient data storing, in the frame of the co-design task, we are going to convert findings from benchmarking GSS applications (some of them are summarized in the deliverables D5.7 and upcoming D5.8) into practical recommendations for hardware providers that might help to build clusters targeting GSS users.

Last but not least, we also plan to concentrate our efforts on improving performance of the network reconstruction application, since it is the most computationally expensive non-embarrassingly parallel part of the CoeGSS workflow we see so far.

8 Pilots' utilization

This chapter shows how the WP3 work is, and can be, utilized by the CoeGSS pilot projects.

8.1 Health Habits pilot

The health habits pilot heavily profited from the synergetic interaction with the WP3 partners. The common work mainly encompassed three areas of key importance: the synthetic information systems (specifically the synthetic population generation), data management/analytics, and, the development and parallelization of the ABM simulation code.

SIS and synthetic population - The pilot developed a general procedure to generate synthetic populations with realistic traits and structural properties in the European area, using aggregated data from Eurostat and national census data. The generation procedure has been split in well separated functional steps so that it can be easily implemented in the project's portal.

The WP3 partners helped in the selection of the output file format and structure as well as in the investigation of performance improvements and bottlenecks individuation in the code. In particular, the population is composed by agents (single entities encoding the traits of a person in the population) that are then assigned to a household (a group of people living together) and, if the agent is a student or a worker, they are assigned to a school or workplace. These location entities are created so as to recreate the size distribution of real schools and workplaces. All the information is reported in three tables contained in an *hdf5* file. The tables contain the agents, the households, and the workplace information, respectively. The agent table is linked to the household and workplace tables by using the household and workplace unique identifier codes (integers) specifying the id of the household the agent is living in or their workplace/school.

Another important improvement deployed in the synthetic population structure is its hierarchical structure. Agent, households and workplaces are grouped in an arbitrary number of levels representing the administrative and local organization of the area under investigation. For example, in a population there may be $L=5$ levels of this hierarchy, starting from the coarser one (countries) passing through regions, provinces and municipalities and then adding a level to split people living in the same municipality in districts of comparable size. In this way, each branch of the hierarchy has a comparable number of people so that the population can be easily distributed amongst multiple computational nodes. This hierarchical representation of the population is encoded in a tuple containing the identifying code of each one of the L levels of the hierarchy. The generation procedure of these levels requires some computationally demanding steps that result in a long generation time (e.g. a population of 14 million people in the north-west of Italy takes about 6 hours of computing time). That is why the WP3 partners working on the synthetic population generation are inspecting the code to find the parts that can be optimized to improve the scalability.

With respect to the current development work, the WP3 partners working on SIS are assisting the pilot to further increase the level of accuracy of the generated population and to improve its match with empirical data. This will be done by implementing a hierarchical iterative proportional fitting

(HIPF) procedure that, starting from micro-samples reporting the age structure and socio-economic indicators of households in different area of Europe, will allow to recreate joint distributions of agents' traits while constraining the marginal distribution of the real population properties. For example, this procedure will allow to improve both the in-house age structure and the parent-children age difference distribution. Indeed, these quantities are currently approximated by using reasonable but arbitrary assumptions on the household composition and the HIPF procedure will allow us to derive this structure from the real data of the micro-sample.

Data management and analytics – Regarding the data management part the pilot improved the input and output data structures to be passed to and retrieved from the simulations. The input improvements have already been described in the previous section on the synthetic population. Regarding the data management, the pilot has implemented in its pipeline the usage of the CKAN repository to share the data with other partners and store it before using it to run simulations. Moreover, the pilot completely re-designed the simulations' output file structure to make it produce a file which is compliant with the commonly agreed output structure, thus enabling HPDA pipelines on the output. Specifically, the output file is now organized in a hierarchical structure where the first level has a value for each processor that performed a part of the simulation, so that a fast aggregation of the results can be obtained using, for example, Spark routines. The pilot also provided some examples of such HPDA pipelines using Spark, showing the time and performance improvements with respect to the traditional serial data analysis pipelines. Moreover, it also started a discussion with WP3 partners to move part of the offline data analysis from the post-processing part to a streaming, run-time analysis of the output data. This improvement shall be carried out so as to save I/O calls, and especially writing calls, that are time consuming when performed on the diskless installations of the HPC systems.

Code parallelization – The work done on the simulation code has been twofold: on one hand the pilot analysed different implementation of parallel ABM code found in the literature and distilled from them the features and design properties that were needed for the pilot use case. On the other hand, it developed from scratch and in close collaboration with the HLRS group the serial prototype of the code, identified the parts of the code to be optimized and improved the overall code performances. Then, starting from this prototype example, a co-design of the parallelization scheme of the serial code has been carried out with the WP3 partners. Specifically, by leveraging on both the hierarchical representation of the synthetic population and the explicit representation of the different contexts in which agents can interact (household, workplace, local community etc.) the message passing scheme to keep the different computing nodes synchronized has been determined.

The preliminary results of the parallelization showed an overall improvement of the elapsed computing time and the need to reduce the I/O operations so as to maximize the parallelization benefit. Last but not least, particular attention will be devoted to make the system entities (agents, households and workplaces) less memory consuming (currently we use about 500MB to represent one million agents in memory) so as to allow for the simulation of even larger populations in the future.

8.2 Green Growth pilot

The green growth pilot developed two different SIS, a preliminary “green car diffusion (GCD)” model, which is implemented using the HPC-ABM framework Pandora (Rubio-Campillo, 2014), and the agent based SIS “Mobility Transformation Model (MoTMo), which is using a HPC-ABM framework that was developed as part of CoeGSS based on the results from discussions and studies described in Deliverable 4.2 Chapter 7.

Thereby the development of the models benefits from the work of WP3 in various ways:

Data analytics - While GCD was calibrated manually (see Deliverable 4.5, chapter 3.2.3 for details), MoTMo will use Dakota for the model calibration and sensitivity analysis. Therefore the 'dakota.interfacing' module was integrated into MoTMo. At the current state we have successfully run small sampling studies on the Eagle cluster for testing the integration. Next, we will make a global sensitivity analysis via a Variance-based Decomposition (see chapter 4.6.2 of the Dakota manual) to identify variables that can be removed as design parameters as they do not have a strong influence on the simulation outcomes.

The calibration itself will try to fit data from 2010 - 2016 for the number of electric cars, number of combustions cars and the number of km people used public transport for each federal state of Germany. As it's difficult to define a single error measure because of the different nature of the data to be fit, we plan do to a categorical criteria calibration, where the categories are intervals around the data to be fit (see also (Thiele, Kurth, & Grimm, 2014)).

The corresponding Dakota study will be defined as multiple optimization using the weighting factor approach (see chapter 6.3.1 of the Dakota manual).

Visualization - As the GCD model is based on Pandora, and a COVISE plug-in has been developed that can read the output of a Pandora simulation (see D3.3, section 5.2.1.1), it is possible to visualize the results via a remote session in 3D. Figure 18 shows the spatial distribution of battery electric vehicles in Europe for the year 2025. See Deliverable 4.4 for more details.

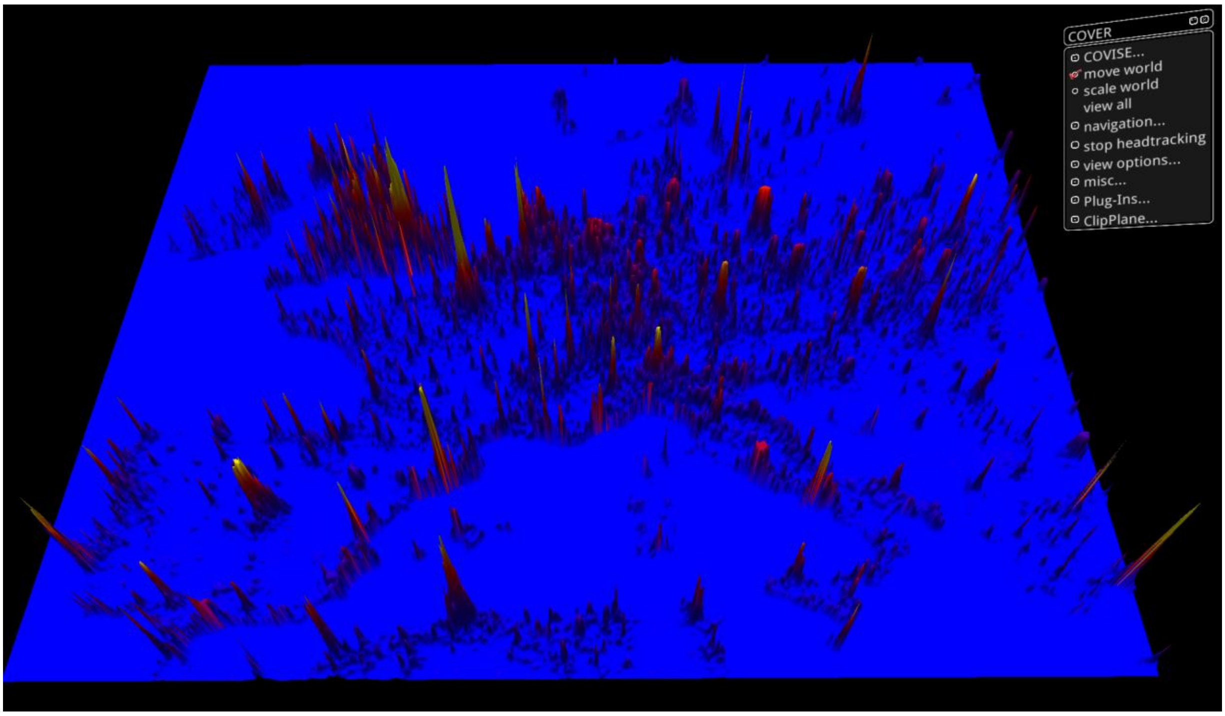


Figure 18: COVISE 3D visualisation.

Network reconstruction - MoTMo will change the to integrate the Network reconstruction code developed by IMT to generation the social network. This will allow a more information-theoretic founded approach to generate similarity networks. Within the process, it will be tested how the different generation methods will impact the model results.

Synthetic population - MoTMo will also use the synthetic population that is generated by the ISI SynPop tool specifically for the German use case. Due to the agreed structure, a subset of the population has been already incorporated for testing.

8.3 Global Urbanisation pilot

SIS and synthetic population - Synthetic populations will prove important in allowing the simulation of realistic populations while protecting private data. They will also allow us to test various replications of a given population. Finally, they open the possibility of varying precise characteristics of the population to account for demographic change or simulate different evolution hypotheses.

This pilot has based its simulations on Paris open data describing population characteristics such as income, residential location, real estate pricing (see Figure 19), commuting flows, car ownership, but also real estate pricing or car pollution.

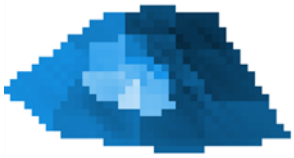


Figure 19: Average real estate prices in the different Paris districts.

Data analytics - Pre-processing data facilitates preparing model simulation. Indeed, finding ad-hoc fine scaled data is often challenging, when it exists. It then requires cleansing, processing, formatting to be made fit for simulation. Data might further require (dis)aggregation and performing calculations. For this pilot we have for instance interpolated initial district data onto a finer grid (see Figure 20).

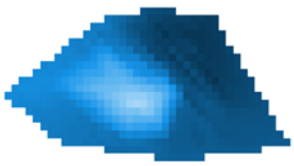


Figure 20: Interpolating to refine data spatial granularity.

Analytics over one simulation can concern different phases of a GSS model life cycle: data pre-processing, model calibration and validation, validated model scenario evaluation. It can involve various kind of operations. For instance, observing results at different scales, to seek the emergence of regular patterns. It can allow to study the joint evolution of various observable to better understand how they might be linked. It can associate different kinds of results to calculate further indicators, based upon one or a few observables, such as global costs, or spatial heterogeneity.

Stochastic models (as can be found for global challenges) request a set of replications, the results of which need to be statistically summarized (mean, standard deviation, confidence interval, …)

More generally, because of their complexity, such models integrate a level of uncertainty in their outcomes, calling for analytics to characterize them in a clearer way.

Completing the model simulation, it allows to calculate analytics on the results, which widen the understanding of the model behaviour at different scales and following different points of view, evaluating key parameters and indicators of interest. Post-processing results allows also to define maps of key observables (pollution level, price, …) or indicators (resilience) in the parameter space over different sets of simulations. We have for instance calculated the evolution over time of the spatial heterogeneity of various indicators to complete basic statistics, in the following figure (Figure 21) for the proportion of commuters preferring public transport over taking their car, and for different versions of our model with increasing complexity. It allows here to reveal how spatial heterogeneity of green commuters increases around time steps 7-9, due to heterogeneous public transport offer before decreasing again, as demand leads the transport offer to expand and even out.

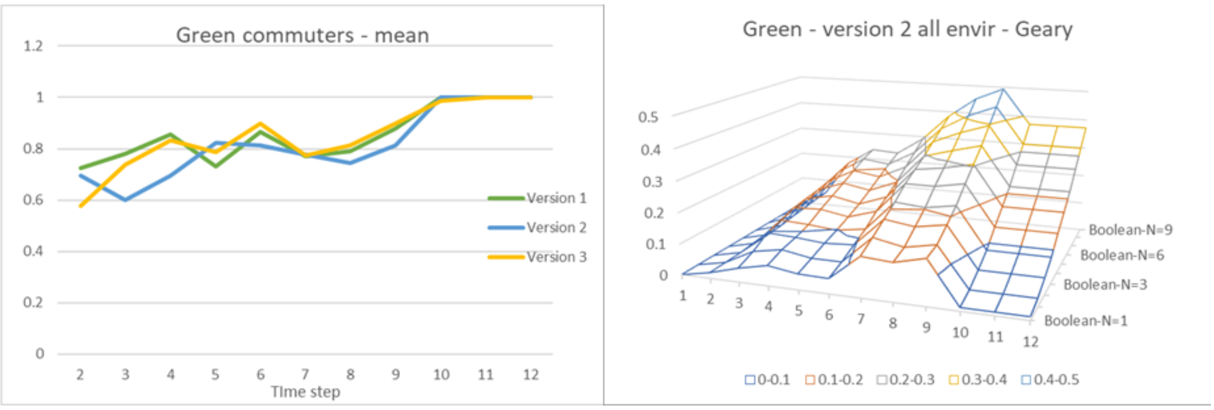


Figure 21: Completing basic statistics on the proportion of green commuters with the evolution of their spatial heterogeneity (following Geary's C).

Observing the results of not only one but a few simulations in the parameter space can allow to fill many purposes. Sensitivity analysis will put into light how one or a few observable(s) (such as pollution or cost) depend(s) on one or a few parameters (such as the ecological awareness of commuters, for instance). In the following figure (Figure 22) we study how to decrease pollution in increasing citizen ecological awareness and public transport adaptability to travellers' demand.

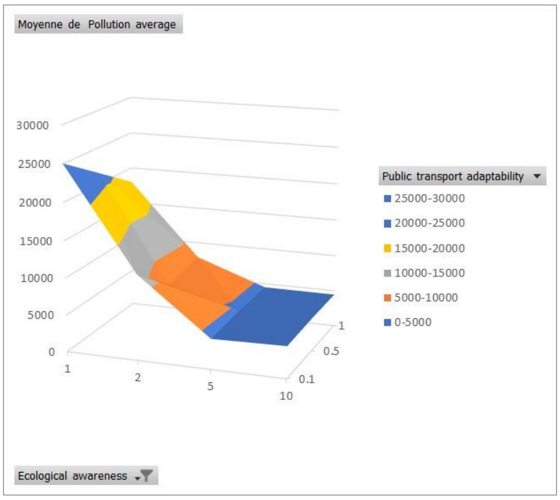


Figure 22: Pollution average depending on ecological awareness and public transport adaptability.

It can allow to define also key parameter values leading to fundamental changes in the evolution of the model (minimal level of incentives leading commuters to predominantly move over to public transports for instance). It can allow to define maps with areas of risk or resiliency for given key performance indicators, and therefore answer high level questions. For instance, in the following graph (Figure 23) appear different levels of pollution depending on levels of ecological awareness and the adaptability of public transport offer to demand.

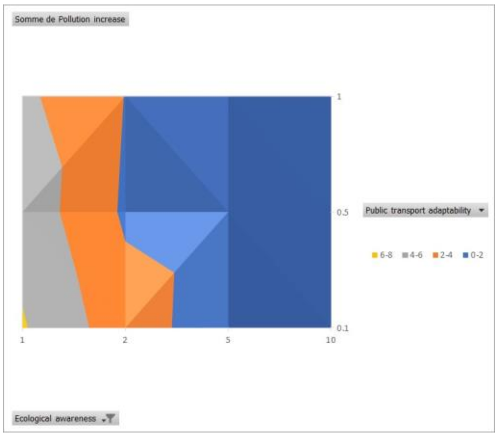


Figure 23: Average levels of pollution depending on ecological awareness and public transport adaptability.

Code parallelization – We tested the scalability (defined as the gain in time when increasing the number of cores to a fixed number) of a simple Cosmo model, when varying different parameters, such as the complexity and time requested of a single operation (not to be parallelized), the number of loops, the number of agents and the intensity of their interactions. Simulation times went from a few seconds to over nine hours. If the two first parameters did not influence scalability much, both the number of agents and the intensity of their interactions seemed more significant (see Figure 24).

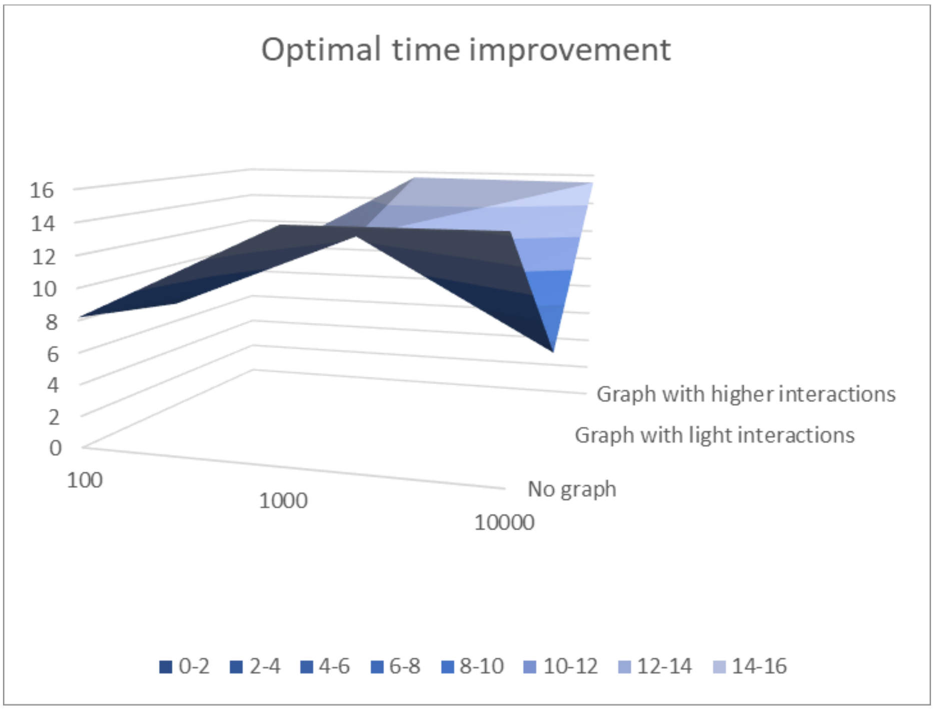


Figure 24: Optimal time improvement (ratio of simulation time between 1 and maximum number of cores) , following the number of agents (varying from 100 to 10000) and the

intensity of their interactions (none, light interactions, interactions requested non-negligible calculation time).

Good news is that scalability seems to increase with difficulty, i.e. high number of agents with interactions requesting calculation time. However, the simulations with light (yet overall) interactions show lower scalability, particularly with the highest number of agents. We hypothesize that the additional time they request (shown in the following graph) is difficult to parallelize.

9 Summary

This deliverable describes parts of the tools and methods used and developed within the CoeGSS project. Together with the other WP3 deliverables (3.1 to 3.8) it documents our journey from the high-level objectives of the pilots (acting as "CoeGSS-internal stakeholders") to new methods, tools and mechanisms (MTMs) for high performance simulations of global systems.

The journey has not been easy, and it is not over yet: even if the CoeGSS project is reaching its end, the work on expanding the reach of HPC to cover the expanding area of Global Systems Science has just begun. Two concrete signs towards creating an expanding market is the HLRS agreement with Baden-Württemberg to set up a centre of competence for global systems and that 7 CoeGSS partners + 12 non-CoeGSS partners have joined forces in a consortium application on "Exascale, Data, and Global Evolutions (EDGE)" (currently under evaluation for the call H2020-INFRAEDI-2018-1).

Concrete technical results of WP3 are described in the different chapters, but some of the most important outcomes are more interdisciplinary: non-technical, cultural changes in the groups of researchers and stakeholders. These outcomes are described in other deliverables, but one part needs to be stressed here: the development of a shared understanding, a common language, to express the key ideas of the stakeholders in a way that is precise enough to be translated into computer code, yet concise enough to be discussed with domain experts.

References

Adams, Brian M. et al. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.7 User's Manual.* Sandia Labs, 2017.

Allwood, J.M., V. Bosetti, N.K. Dubash, L. Gómez-Echeverri, and C. von Stechow. *Allwood, J.M., V. Bosetti, N.K. Dubash, L. Gómez-Echeverri, a"Glossary." In Climate Change 2014: Mitigation of Climate Change. Contribution of Working Group III to the Fifth Assessment Report of the Intergovernmental Panel on Clim.* New York, NY, USA: Cambridge University Press, 2014.

Avila, Lisa S., et al. "The VTK user's guide." 2010.

Bader, David A., and Kamesh Madduri. "SNAP, Small-world Network Analysis and Partitioning: An open-source parallel graph framework for the exploration of large-scale networks." *IPDPS.* IEEE, 2008. 1-12.

Barthélemy, Marc. "Spatial networks." *Physics Reports* 499, no. 1-3 (2011): 1 - 101.

Bauer, N., L. Baumstark, M. Haller, M. Leimbach, G. Luderer, M. Lueken, R. Pietzcker, et al. "REMIND: The Equations." 2011. https://www.pik-potsdam.de/research/sustainable-solutions/models/remind/remind-equations.pdf.

Bollobas, B. *Random Graphs.* Edited by W. Fulton, A. Katok, F. Kirwan, P. Sarnak, B. Simon and B. Totaro. Cambridge University Press, 2001.

Bonabeau, Eric. "Agent-Based Modeling: Methods and Techniques for Simulating Human Systems." *Proceedings of the National Academy of Sciences of the United States of America* (National Academy of Sciences) 99 (2002): 7280-7287.

Boriah, Shyam, Varun Chandola, and Vipin Kumar. "Similarity measures for categorical data: A comparative evaluation." *In Proceedings of the eighth SIAM International Conference on Data Mining.* 243-254.

Botta, N., A. Mandel, M. Hofmann, S. Schupp, and C. Ionescu. "Mathematical Specification of an Agent-Based Model of Exchange." *Proceedings of the AISB Convention 2013, "Do-Form: Enabling Domain Experts to Use Formalized Reasoning" Symposium.* 2013.

Botta, Nicola, Patrik Jansson, and Cezar Ionescu. "Contributions to a Computational Theory of Policy Advice and Avoidability." *Journal of Functional Programming*, 2017.

Botta, Nicola, Patrik Jansson, and Cezar Ionescu. "The Impact of Uncertainty on Optimal Emission Policies." *Earth System Dynamics*, 2018.

Botta, Nicola, Patrik Jansson, Cezar Ionescu, David R Christiansen, and Edwin Brady. "Sequential Decision Problems, Dependent Types and Generic Solutions." *Logical Methods in Computer Science 13*, 2017.

Brady, Edwin. *Type-Driven Development with Idris.* Manning Publications Company, 2017.

Chevalier, C., and F. Pellegrini. “PT-Scotch: A Tool for Efficient Parallel Graph Ordering.” *Parallel Comput.* (Elsevier Science Publishers B. V.) 34 (2008): 318-331.

Cimini, Giulio, et al. “Reconstructing Topological Properties of Complex Networks Using the Fitness Model.” In *Social Informatics: SocInfo 2014 International Workshops, Barcelona, Spain, November 11, 2014, Revised Selected Papers*, edited by Luca Maria Aiello and Daniel McFarland, 323-333. Cham: Springer International Publishing, 2015.

Cimini, Giulio, Tiziano Squartini, Andrea Gabrielli, and Diego Garlaschelli. “Estimating topological properties of weighted networks from limited information.” *Phys. Rev. E* (American Physical Society) 92, no. 4 (Oct 2015): 040802.

Cover, Thomas M., and Joy A. Thomas. *Elements of Information Theory.* New York, NY, USA.: Wiley-Interscience, 2006.

Deming, W.E., and F.F. Stephan. “On a Least Squares Adjustment of a Sampled Frequency Table When the Expected MarginalTotals are Known.” *The Annals of Mathematical Statistics, Vol. 11, No. 4*, 1940: 427-444.

Finus, Michael, Ekko van Ierland, and Rob Dellink. “Stability of Climate Coalitions in a Cartel Formation Game.” *FEEM Working Paper No. 61*, 2003.

Gallagher, S., L. Richardson, S. L. Ventura, and W. F. Eddy. “SPEW: Synthetic Populations and Ecosystems of the World.” *ArXiv e-prints*, #jan# 2017.

Gintis, H. “The Dynamics of General Equilibrium.” *Economic Journal*, 2007.

Gintis, H. “The Emergence of a Price System from Decentralized Bilateral Exchange.” *B. E. Journal of Theoretical Economics*, 2006.

Gonzalez, Joseph E., Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. “PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs.” *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12).* Hollywood, CA: USENIX, 2012. 17-30.

Greenemeier, Larry. *When Will Computers Have Common Sense? Ask Facebook.* 2016. https://www.scientificamerican.com/article/when-will-computers-have-common-sense-ask-facebook/#.

Gregor, Douglas, and Andrew Lumsdaine. “The Parallel BGL: A Generic Library for Distributed Graph Computations.” *Parallel Object-Oriented Scientific Computing (POOSC).* 2005.

HDF Group. *HDF Forum: Compression and chunk size issue.* https://forum.hdfgroup.org/t/compression-and-chunk-size-issue/4381.

—. *Introduction to HDF5.* 2018. https://support.hdfgroup.org/HDF5/doc/H5.intro.html.

Heitzig, Jobst. “Bottom-up Strategic Linking of Carbon Markets: Which Climate Coalitions Would Farsighted Players Form?” *SSRN Environmental Economics EJournal*, 2012.

Helm, Carsten. “International Emissions Trading with Endogenous Allowance Choices.” *Journal of Public Economics 87: 2737–47*, 2003.

Ionescu, Cezar. “Vulnerability Modelling with Functional Programming and Dependent Types.” *Mathematical Structures in Computer Science 26 (1)*, 2016: 114-128.

Karypis, G., and K. Schloegel. “ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 4.0.” 2013, 32.

Lee, Sophia Seung-yoon. “Fuzzy-Set Method in Comparative Social Policy: A Critical Introduction and Review of the Applications of the Fuzzy-Set Method.” *Quality & Quantity 47 (4)*, 2013.

Leskovec, Jure, and Rok Sosic. “SNAP: A General Purpose Network Analysis and Graph Mining Library.” *CoRR* abs/1606.07550 (2016).

Lin, Dekang. “An Information-Theoretic Definition of Similarity.” *In Proceedings of the 15th International Conference on Machine Learning.* San Francisco, CA, USA: Morgan Kaufmann, 1998. 296–304.

Mandel, A., S., W. Fürst, F. Meissner Lass, and C Jaeger. “Lagom generiC: an agent-based model of growing economies.” *ECF Working Paper 1*, 2009.

Mazzarisi, Piero, and Fabrizio Lillo. “Methods for Reconstructing Interbank Networks from Limited Information: A Comparison.” In *Econophysics and Sociophysics: Recent Progress and Future Directions*, edited by Frédéric Abergel, et al., 201-215. Cham: Springer International Publishing, 2017.

Metz, Cade. *Google’s Dueling Neural Networks Spar to Get Smarter, No Humans Required.* 2017. https://www.wired.com/2017/04/googles-dueling-neural-networks-spar-get-smarter-no-humans-required/?imm_mid=0f0e59&cmp=em-data-na-na-newsltr_ai_20170417.

MIDAS data source. 2018. http://data.olympus.psc.edu/.

Moore, Ramon E. “Interval analysis. .” *Prentice-Hall, Englewood Cliffs, N.J*, 1966.

Najd, Shayan, Sam Lindley, Josef Svenningsson, and Philip Wadler. “Everything old is new again: Quoted Domain Specific Languages.” *Partial Evaluation and Program Manipulation 2016.* St. Petersburg: ACM, 2016. 25-36.

Office for National Statistics. Social and Vital Statistics Division. “General Household Survey, 2006, 3rd Edition.” *2009.* UK Data Service, 2009.

PostgreSQL Wiki - Tuning Your PostgreSQL Server. https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server.

Raven, Peter, Rosina Bierbaum, and John Holdren. ""Confronting Climate Change: Avoiding the Unmanageable and Managing the Unavoidable." UN-Sigma Xi Climate Change Report." 2007.

Renaud Lambiotte, Vincent D. Blondel, Cristobald de Kerchove, Etienne Huens, Christophe Prieur, Zbigniew Smoreda, Paul Van Dooren. "Geographical dispersal of mobile communication networks." *Physica A: Statistical Mechanics and its Applications*, 2008: 5317 - 5325.

Research Domain III, PIK. *ReMIND-R.* 2013. http://www.pik-potsdam.de/research/sustainable-solutions/models/remind.

Rump, S., T. Ogita, Y Morikura, and S. . Oishi. "Interval arithmetic with fixed rounding mode." *Nonlinear Theory and Its Applications, IEICE 754* 7, no. 3 (2016): 362-373.

Saracco, Fabio, Riccardo Di Clemente, Andrea Gabrielli, and Tiziano Squartini. "Detecting early signs of the 2007--2008 crisis in the world trade." *Scientific Reports* (Nature Publishing Group) 6 (2016).

Schellnhuber, Hans Joachim. "Discourse: Earth System Analysis - the Scope of the Challenge." In *Earth System Analysis: Integrating Science for Sustainability*, by Hans Joachim Schellnhuber and Volker Wenzel, 3–195. Berlin/Heidelberg: Springer, 1998.

Schloegel, Kirk, George Karypis, and Vipin Kumar. "Sourcebook of Parallel Computing." Chap. Graph Partitioning for High-performance Scientific Simulations in *Sourcebook of Parallel Computing*, edited by Jack Dongarra, et al., 491-541. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

Smith, Douglas R. "The Design of Divide and Conquer Algorithms." *Sci. Comput. Program.* (Elsevier North-Holland, Inc.) 5 (#feb# 1985): 37-58.

Squartini, T., I. van Lelyveld, and D. Garlaschelli. "Early-warning signals of topological collapse in interbank networks." *Scientific Reports* 3 (#nov# 2013): 3357.

Tatara, Eric, Nicholson Collier, Jonathan Ozik, and Charles Macal. "Endogenous Social Networks from Large-Scale Agent-Based Models ." *Proceedings of IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* 2017.

team, CoeGSS project. *D4.5: – Second Status Report of the Pilots.* CoeGSS, 2017.

Thiele, Jan C., Winfried Kurth, and Volker Grimm. " Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and 'R'." *Journal of Artificial Societies and Social Simulation*, 2014.