

Multilevel Prediction

Generated by Doxygen 1.8.6

Mon Nov 9 2015 19:16:58

Contents

1	Main Page	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	ChainExperiment Class Reference	9
5.1.1	Member Data Documentation	9
5.1.1.1	contrarianMax	9
5.1.1.2	contrarianMin	10
5.1.1.3	contrarianStep	10
5.1.1.4	delayMax	10
5.1.1.5	delayMin	10
5.1.1.6	delayStep	10
5.1.1.7	id	10
5.1.1.8	postMeasurements	10
5.1.1.9	preMeasurements	10
5.1.1.10	threshold	10
5.1.1.11	timeMax	10
5.1.1.12	timeMin	10
5.1.1.13	timeStep	10
5.1.1.14	update	11
5.2	ChainVoterGraph Class Reference	11
5.3	CompleteVoterGraph Class Reference	11
5.3.1	Detailed Description	12
5.3.2	Constructor & Destructor Documentation	12
5.3.2.1	CompleteVoterGraph	12

5.4	DataPointStruct Struct Reference	12
5.4.1	Detailed Description	12
5.5	EmptyVoterMeasurement Class Reference	12
5.5.1	Detailed Description	13
5.5.2	Constructor & Destructor Documentation	13
5.5.2.1	EmptyVoterMeasurement	13
5.6	MacroVoterMeasurement Class Reference	13
5.6.1	Detailed Description	14
5.6.2	Constructor & Destructor Documentation	14
5.6.2.1	MacroVoterMeasurement	14
5.7	MarkovDataSet Class Reference	14
5.8	MarkovProcess Class Reference	14
5.8.1	Detailed Description	16
5.8.2	Constructor & Destructor Documentation	16
5.8.2.1	MarkovProcess	16
5.8.3	Member Function Documentation	16
5.8.3.1	computeStationaryDistribution	16
5.8.3.2	computeTrajectory	16
5.8.3.3	setDistribution	17
5.8.3.4	setTransition	17
5.8.3.5	setTransition	17
5.8.4	Member Data Documentation	17
5.8.4.1	distribution	17
5.8.4.2	distributions	17
5.8.4.3	lastDelay	17
5.8.4.4	lastTime	17
5.8.4.5	size	17
5.8.4.6	transition	18
5.8.4.7	transitions	18
5.9	MarkovTrajectory Class Reference	18
5.10	MicroVoterMeasurement Class Reference	18
5.10.1	Detailed Description	19
5.10.2	Constructor & Destructor Documentation	19
5.10.2.1	MicroVoterMeasurement	19
5.11	OrderedPartition Class Reference	19
5.12	Part Class Reference	19
5.12.1	Detailed Description	20
5.12.2	Constructor & Destructor Documentation	20
5.12.2.1	Part	20
5.13	Partition Class Reference	20

5.13.1 Detailed Description	21
5.14 Timer Class Reference	21
5.14.1 Detailed Description	21
5.15 TwoCommunitiesExperiment Class Reference	21
5.15.1 Detailed Description	22
5.15.2 Constructor & Destructor Documentation	23
5.15.2.1 TwoCommunitiesExperiment	23
5.15.3 Member Data Documentation	24
5.15.3.1 compactModel	24
5.15.3.2 contrarian1Max	24
5.15.3.3 contrarian1Min	24
5.15.3.4 contrarian1Step	24
5.15.3.5 contrarian2Max	24
5.15.3.6 contrarian2Min	24
5.15.3.7 contrarian2Step	24
5.15.3.8 delayMax	24
5.15.3.9 delayMin	25
5.15.3.10 delayStep	25
5.15.3.11 equalContrarian	25
5.15.3.12 equalInterRate	25
5.15.3.13 equalIntraRate	25
5.15.3.14 equalSize	25
5.15.3.15 id	25
5.15.3.16 id_number	25
5.15.3.17 interR1Max	25
5.15.3.18 interR1Min	25
5.15.3.19 interR1Step	26
5.15.3.20 interR2Max	26
5.15.3.21 interR2Min	26
5.15.3.22 interR2Step	26
5.15.3.23 intraR1Max	26
5.15.3.24 intraR1Min	26
5.15.3.25 intraR1Step	26
5.15.3.26 intraR2Max	26
5.15.3.27 intraR2Min	26
5.15.3.28 intraR2Step	26
5.15.3.29 oppositeInterRate	26
5.15.3.30 postMeasurements	26
5.15.3.31 preMeasurements	27
5.15.3.32 size1Max	27

5.15.3.33 size1Min	27
5.15.3.34 size1Step	27
5.15.3.35 size2Max	27
5.15.3.36 size2Min	27
5.15.3.37 size2Step	27
5.15.3.38 threshold	27
5.15.3.39 timeMax	27
5.15.3.40 timeMin	27
5.15.3.41 timeStep	27
5.15.3.42 update	27
5.16 TwoCommunitiesVoterGraph Class Reference	28
5.16.1 Detailed Description	28
5.16.2 Constructor & Destructor Documentation	29
5.16.2.1 TwoCommunitiesVoterGraph	29
5.16.3 Member Function Documentation	29
5.16.3.1 getCompactMarkovPartition	29
5.16.3.2 getCompactMarkovPartition	29
5.16.3.3 getCompactMarkovProcess	29
5.16.4 Member Data Documentation	30
5.16.4.1 community1	30
5.16.4.2 community2	30
5.16.4.3 contrarian1	30
5.16.4.4 contrarian2	30
5.16.4.5 interRate1	30
5.16.4.6 interRate2	30
5.16.4.7 intraRate1	30
5.16.4.8 intraRate2	30
5.16.4.9 size1	30
5.16.4.10 size2	30
5.17 VoterBinning Class Reference	31
5.18 VoterDataSet Class Reference	31
5.19 VoterEdge Class Reference	32
5.19.1 Detailed Description	32
5.19.2 Constructor & Destructor Documentation	32
5.19.2.1 VoterEdge	32
5.19.3 Member Data Documentation	32
5.19.3.1 node1	32
5.19.3.2 node2	32
5.19.3.3 weight	32
5.20 VoterGraph Class Reference	33

5.20.1 Detailed Description	34
5.20.2 Constructor & Destructor Documentation	34
5.20.2.1 VoterGraph	34
5.20.3 Member Function Documentation	34
5.20.3.1 addEdge	34
5.20.3.2 addNode	34
5.20.3.3 getMarkovPartition	34
5.20.3.4 getMarkovPartition	35
5.20.3.5 getMarkovProcess	35
5.20.4 Member Data Documentation	35
5.20.4.1 edgeNumber	35
5.20.4.2 edgeSet	35
5.20.4.3 edgeWeight	35
5.20.4.4 nodeMap	35
5.20.4.5 nodeNumber	35
5.20.4.6 nodeSet	35
5.20.4.7 nodeWeight	36
5.20.4.8 process	36
5.20.4.9 updateProcess	36
5.21 VoterMeasurement Class Reference	36
5.21.1 Detailed Description	37
5.21.2 Constructor & Destructor Documentation	37
5.21.2.1 VoterMeasurement	37
5.21.3 Member Function Documentation	37
5.21.3.1 addProbe	37
5.21.3.2 print	37
5.21.4 Member Data Documentation	37
5.21.4.1 graph	37
5.21.4.2 metricMap	37
5.21.4.3 partition	37
5.21.4.4 probeMap	37
5.21.4.5 probeNumber	38
5.21.4.6 type	38
5.22 VoterMeasurementState Class Reference	38
5.23 VoterMeasurementTrajectory Class Reference	38
5.24 VoterNode Class Reference	39
5.24.1 Detailed Description	39
5.24.2 Constructor & Destructor Documentation	39
5.24.2.1 VoterNode	39
5.24.3 Member Data Documentation	39

5.24.3.1	contrarian	39
5.24.3.2	id	40
5.24.3.3	inEdgeNumber	40
5.24.3.4	inEdgeSet	40
5.24.3.5	inEdgeWeight	40
5.24.3.6	outEdgeNumber	40
5.24.3.7	outEdgeSet	40
5.24.3.8	outEdgeWeight	40
5.24.3.9	weight	40
5.25	VoterProbe Class Reference	40
5.25.1	Detailed Description	41
5.25.2	Constructor & Destructor Documentation	41
5.25.2.1	VoterProbe	41
5.25.3	Member Function Documentation	41
5.25.3.1	addNode	41
5.25.3.2	addNodes	41
5.25.3.3	print	41
5.25.3.4	setNodeSet	42
5.25.4	Member Data Documentation	42
5.25.4.1	graph	42
5.25.4.2	nodeNumber	42
5.25.4.3	nodeSet	42
5.26	VoterState Class Reference	42
5.27	VoterTrajectory Class Reference	43
6	File Documentation	45
6.1	/home/lamarche/programming/multilevel_prediction/src/csv_tools.hpp File Reference	45
6.1.1	Detailed Description	46
6.1.2	Function Documentation	46
6.1.2.1	deleteCSV	46
6.1.3	Variable Documentation	46
6.1.3.1	VERBOSE	46
6.2	/home/lamarche/programming/multilevel_prediction/src/main.hpp File Reference	46
6.2.1	Detailed Description	47
6.3	/home/lamarche/programming/multilevel_prediction/src/markov_process.hpp File Reference	47
6.3.1	Detailed Description	47
6.4	/home/lamarche/programming/multilevel_prediction/src/partition.hpp File Reference	47
6.4.1	Detailed Description	48
6.5	/home/lamarche/programming/multilevel_prediction/src/timer.hpp File Reference	48
6.5.1	Detailed Description	49

6.6	/home/lamarche/programming/multilevel_prediction/src/voter_experiment.hpp File Reference . . .	49
6.6.1	Detailed Description	50
6.6.2	Function Documentation	50
6.6.2.1	twoCommunitiesExperiment	50
6.7	/home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp File Reference	50
6.7.1	Detailed Description	52
6.7.2	Enumeration Type Documentation	52
6.7.2.1	MeasurementType	52
6.7.2.2	UpdateProcess	52
6.7.2.3	VoterMetric	53
	Index	54

Chapter 1

Main Page

This program is a toolbox to compute optimal predictors of Markov chains, and in particular multilevel agent-based systems, by using the information bottleneck method. For details regarding the formal grounds of this method, please refer to:

Robin Lamarche-Perrin, Sven Banisch, and Eckehard Olbrich. The Information Bottleneck Method for Optimal Prediction of Multilevel Agent-based Systems. Technical Report MIS-Preprint 55/2015, Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany, 2015.

<http://www.mis.mpg.de/publications/preprints/2015/prepr2015-55.html>

Copyright © 2015 Robin Lamarche-Perrin (Robin.Lamarche-Perrin@lip6.fr)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ChainExperiment	9
DataPointStruct	12
MarkovDataSet	14
MarkovProcess	14
MarkovTrajectory	18
OrderedPartition	19
Part	19
Partition	20
Timer	21
TwoCommunitiesExperiment	21
VoterBinning	31
VoterDataSet	31
VoterEdge	32
VoterGraph	33
ChainVoterGraph	11
CompleteVoterGraph	11
TwoCommunitiesVoterGraph	28
VoterMeasurement	36
EmptyVoterMeasurement	12
MacroVoterMeasurement	13
MicroVoterMeasurement	18
VoterMeasurementState	38
VoterMeasurementTrajectory	38
VoterNode	39
VoterProbe	40
VoterState	42
VoterTrajectory	43

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ChainExperiment	9
ChainVoterGraph	11
CompleteVoterGraph	
An interaction graph with edges between each pair of nodes (in both direction, with equal weight for each edge)	11
DataPointStruct	
The computation time and memory consumption associated to an experiment of a given size	12
EmptyVoterMeasurement	
A measurement without any probe (no observation)	12
MacroVoterMeasurement	
A measurement consisting in one probe observing all nodes of the interaction graph	13
MarkovDataSet	14
MarkovProcess	
A finite Markov chain described by a discrete state space, an initial distribution, and a transition kernel	14
MarkovTrajectory	18
MicroVoterMeasurement	
A measurement consisting in one probe for each node of the interaction graph	18
OrderedPartition	19
Part	
A part of a partition (i.e., a set of individuals)	19
Partition	
A partition (i.e., a set of disjoint and covering parts)	20
Timer	
A timer mesuring computation times and memory consumption during the program execution	21
TwoCommunitiesExperiment	
Programming a prediction experiment on a two-communities Voter Model	21
TwoCommunitiesVoterGraph	
An interaction graph consisting in two communities of nodes (complete graph within each community, complete interaction between the two communities, possibly with different weights)	28
VoterBinning	31
VoterDataSet	31
VoterEdge	
An edge of the interaction graph	32
VoterGraph	
The interaction graph describing a Voter Model	33
VoterMeasurement	
A measurement to observe the Voter Model according set of probes	36

VoterMeasurementState	38
VoterMeasurementTrajectory	38
VoterNode A node of the interaction graph	39
VoterProbe A probe to observe the Voter Model according to a subset of nodes	40
VoterState	42
VoterTrajectory	43

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

/home/lamarche/programming/multilevel_prediction/src/ coconut.hpp	??
/home/lamarche/programming/multilevel_prediction/src/ csv_tools.hpp	
Tools to handle input and output CSV files	45
/home/lamarche/programming/multilevel_prediction/src/ landuse.hpp	??
/home/lamarche/programming/multilevel_prediction/src/ main.hpp	
Main program to run tests and experiments (see for example class VoterExperiment)	46
/home/lamarche/programming/multilevel_prediction/src/ markov_process.hpp	
Class to build a finite Markov chain	47
/home/lamarche/programming/multilevel_prediction/src/ partition.hpp	
Classes to build partitions of the state space of Markov chains	47
/home/lamarche/programming/multilevel_prediction/src/ timer.hpp	
Tools to mesure computation times and memory consumption during the program execution . .	48
/home/lamarche/programming/multilevel_prediction/src/ voter_experiment.hpp	
Tools to run prediction experiments on Voter Models	49
/home/lamarche/programming/multilevel_prediction/src/ voter_graph.hpp	
Classes to build an interaction graph (nodes and edges) describing a Voter Model and some observation tools (probes and measurements)	50

Chapter 5

Class Documentation

5.1 ChainExperiment Class Reference

Public Member Functions

- **ChainExperiment** (int size, double contrarian, bool ring, double time, double delay, SpecMeasurementSet *preMeasurements, SpecMeasurementSet *postMeasurements)

Public Attributes

- int id
- UpdateProcess update
- double threshold
- bool withAggregation
- bool ring
- int sizeMin
- int sizeMax
- int sizeStep
- double contrarianMin
- double contrarianMax
- double contrarianStep
- int timeMin
- int timeMax
- int timeStep
- int delayMin
- int delayMax
- int delayStep
- SpecMeasurementSet * preMeasurements
- SpecMeasurementSet * postMeasurements

Static Public Attributes

- static int id_number = 0

5.1.1 Member Data Documentation

5.1.1.1 double ChainExperiment::contrarianMax

The contrarian rate of nodes within community 1 in the last experiment

5.1.1.2 double ChainExperiment::contrarianMin

The contrarian rate of nodes within community 1 in the first experiment

5.1.1.3 double ChainExperiment::contrarianStep

The contrarian rate of nodes within community 1 between two consecutive experiments

5.1.1.4 int ChainExperiment::delayMax

The delay before post-measurement in the last experiment

5.1.1.5 int ChainExperiment::delayMin

The delay before post-measurement in the first experiment

5.1.1.6 int ChainExperiment::delayStep

The delay before post-measurement between two consecutive experiments

5.1.1.7 int ChainExperiment::id

A unique experiment number

5.1.1.8 SpecMeasurementSet* ChainExperiment::postMeasurements

A set of post-measurement to be predicted

5.1.1.9 SpecMeasurementSet* ChainExperiment::preMeasurements

A set of pre-measurement for prediction

5.1.1.10 double ChainExperiment::threshold

The precision threshold to compute the stationary distribution of the Markov chain (see computeStationary-Distribution method in [MarkovProcess](#) class)

5.1.1.11 int ChainExperiment::timeMax

The time of pre-measurement (-1 for stationary distribution) in the last experiment

5.1.1.12 int ChainExperiment::timeMin

The time of pre-measurement (-1 for stationary distribution) in the first experiment

5.1.1.13 int ChainExperiment::timeStep

The time of pre-measurement (-1 for stationary distribution) between two consecutive experiments

5.1.1.14 UpdateProcess ChainExperiment::update

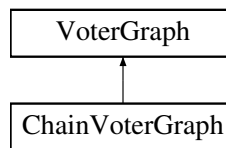
The update process of the built Voter Model

The documentation for this class was generated from the following files:

- [/home/lamarche/programming/multilevel_prediction/src/voter_experiment.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/voter_experiment.cpp](#)

5.2 ChainVoterGraph Class Reference

Inheritance diagram for ChainVoterGraph:



Public Member Functions

- **ChainVoterGraph** (int size, double contrarian=0, bool ring=false, int update=[UPDATE_EDGES](#))

Public Attributes

- int **size**
- double **contrarian**
- [VoterNode](#) ** **nodeArray**
- bool **ring**

The documentation for this class was generated from the following files:

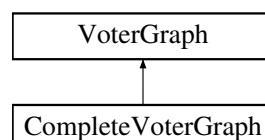
- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp](#)

5.3 CompleteVoterGraph Class Reference

An interaction graph with edges between each pair of nodes (in both direction, with equal weight for each edge)

```
#include <voter_graph.hpp>
```

Inheritance diagram for CompleteVoterGraph:



Public Member Functions

- [CompleteVoterGraph](#) (int size, int update=[UPDATE_EDGES](#), double contrarian=0)
Constructor.

Additional Inherited Members

5.3.1 Detailed Description

An interaction graph with edges between each pair of nodes (in both direction, with equal weight for each edge)

5.3.2 Constructor & Destructor Documentation

5.3.2.1 CompleteVoterGraph::CompleteVoterGraph (int *size*, int *update* = UPDATE_EDGES, double *contrarian* = 0)

Constructor.

Parameters

<i>size</i>	: Size of the graph
<i>update</i>	: How the system evolves at each simulation step (UPDATE_NODES or UPDATE_EDGES)
<i>contrarian</i>	: The contrarian rate of each node

The documentation for this class was generated from the following files:

- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp](#)

5.4 DataPointStruct Struct Reference

The computation time and memory consumption associated to an experiment of a given size.

```
#include <timer.hpp>
```

Public Attributes

- int **size**
- float **time**
- int **memory**

5.4.1 Detailed Description

The computation time and memory consumption associated to an experiment of a given size.

The documentation for this struct was generated from the following file:

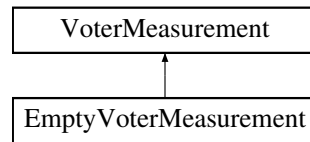
- [/home/lamarche/programming/multilevel_prediction/src/timer.hpp](#)

5.5 EmptyVoterMeasurement Class Reference

A measurement without any probe (no observation)

```
#include <voter_graph.hpp>
```

Inheritance diagram for EmptyVoterMeasurement:



Public Member Functions

- [EmptyVoterMeasurement](#) ([VoterGraph](#) *graph)

Constructor.

Additional Inherited Members

5.5.1 Detailed Description

A measurement without any probe (no observation)

5.5.2 Constructor & Destructor Documentation

5.5.2.1 EmptyVoterMeasurement::EmptyVoterMeasurement ([VoterGraph](#) * graph)

Constructor.

Parameters

graph	: The interaction graph to be observed
-----------------------	--

The documentation for this class was generated from the following files:

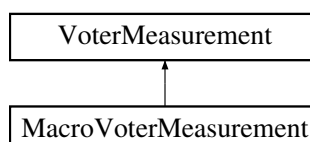
- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.cpp](#)

5.6 MacroVoterMeasurement Class Reference

A measurement consisting in one probe observing all nodes of the interaction graph.

```
#include <voter_graph.hpp>
```

Inheritance diagram for MacroVoterMeasurement:



Public Member Functions

- [MacroVoterMeasurement](#) ([VoterGraph](#) *graph, std::set< [VoterMetric](#) > metrics)

Constructor.

Additional Inherited Members

5.6.1 Detailed Description

A measurement consisting in one probe observing all nodes of the interaction graph.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 MacroVoterMeasurement::MacroVoterMeasurement (VoterGraph * *graph*, std::set< VoterMetric > *metrics*)

Constructor.

Parameters

<i>graph</i>	: The interaction graph to be observed
<i>metrics</i>	: The set of metrics associated to the macro-probe

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp
- /home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp

5.7 MarkovDataSet Class Reference

Public Member Functions

- **MarkovDataSet** ([MarkovProcess](#) *process, int size, int time, int length)
- double **computeScore** ([Partition](#) *preP, [Partition](#) *postP, int delay, int trainingLength)

Public Attributes

- [MarkovProcess](#) * **process**
- int **size**
- int **time**
- int **length**
- [MarkovTrajectory](#) ** **trajectories**

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/markov_process.hpp
- /home/lamarche/programming/multilevel_prediction/src/markov_process.cpp

5.8 MarkovProcess Class Reference

A finite Markov chain described by a discrete state space, an initial distribution, and a transition kernel.

```
#include <markov_process.hpp>
```

Public Member Functions

- [MarkovProcess](#) (int **size**)

Constructor.

- [~MarkovProcess](#) ()
Destructor.
- void [print](#) ()
Print the Markov chain structure and details.
- void [setDistribution](#) (double *array)
Set the initial distribution.
- void [setTransition](#) (double *array)
Set the transition kernel.
- void [setTransition](#) (int i, double *array)
Set one line of the transition kernel.
- double * [getDistribution](#) (int time)
Get the state distribution at a given time (-1 for the stationary distribution)
- double * [getTransition](#) (int delay)
Get the transition kernel for a given number of simulation steps (delay)
- void [computeStationaryDistribution](#) (double threshold)
Compute the stationary distribution of the Markov chain by iterating the transition kernel.
- [MarkovTrajectory](#) * [computeTrajectory](#) (int time, int length)
Compute a possible trajectory of the Markov chain.
- double [getProbability](#) (int individual, int currentTime)
Get the probability to be in a given state at a given time (-1 for the stationary distribution)
- double [getProbability](#) ([Part](#) *part, int currentTime)
Get the probability to be in a given subset of states at a given time (-1 for the stationary distribution)
- double [getNextProbability](#) (int nextIndividual, int currentIndividual, int delay)
Get the probability to be in a given state after a given delay knowing the current state.
- double [getNextProbability](#) ([Part](#) *nextPart, int currentIndividual, int delay)
Get the probability to be in a given subset of states after a given delay knowing the current state.
- double [getNextProbability](#) ([Part](#) *nextPart, [Part](#) *currentPart, int delay, int time)
Get the probability to be in a given subset of states after a given delay knowing the current subset of states at a given time (-1 for the stationary distribution)
- double [getEntropy](#) ([Partition](#) *partition, int currentTime)
Get the Shannon entropy of the state distribution at a given time (-1 for the stationary distribution) when lumped according to a given partition of the state space.
- double [getMutualInformation](#) ([Partition](#) *nextPartition, [Partition](#) *currentPartition, int delay, int time)
Get the mutual information between the state distribution at a given time (-1 for the stationary distribution) and the state distribution after a given delay, when both are lumped according to a given partition of the state space.
- double [getPartMutualInformation](#) ([Partition](#) *nextPartition, [Part](#) *currentPart, int delay, int time)
- double [getNextEntropy](#) ([Partition](#) *partition, bool micro, int delay, int time)
Get the Shannon entropy of the next state distribution knowing the current state distribution at a given time (-1 for the stationary distribution), when the next distribution is lumped according to a given partition of the state space, and when the current partition is also lumped by the same partition (when micro is false)
- double [getInformationFlow](#) ([Partition](#) *partition, int delay, int time)
Get the information flow at a given time (-1 for the stationary distribution) between the Markov chain lumped according to a given partition of the state space and the microscopic Markov chain.
- int * [getOptimalCut](#) (int microSize, double *macroEntropy, double *macroInformation, double beta)
- std::set< [OrderedPartition](#) * > * [getOptimalOrderedPartition](#) ([Partition](#) *nextPartition, [Partition](#) *currentPartition, int delay, int time, double threshold)

Public Attributes

- int [size](#)
- double * [distribution](#)
- std::vector< double * > * [distributions](#)
- int [lastTime](#)
- double * [transition](#)
- std::vector< double * > * [transitions](#)
- int [lastDelay](#)

5.8.1 Detailed Description

A finite Markov chain described by a discrete state space, an initial distribution, and a transition kernel.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 MarkovProcess::MarkovProcess (int *s*)

Constructor.

Parameters

<i>size</i>	: The size of the Markov chain state space
-------------	--

Author

Robin Lamarche-Perrin

Date

22/01/2015

5.8.3 Member Function Documentation

5.8.3.1 void MarkovProcess::computeStationaryDistribution (double *threshold*)

Compute the stationary distribution of the Markov chain by iterating the transition kernel.

Parameters

<i>threshold</i>	: Determines stationarity by giving the minimal difference between two probability values in two different but consecutive distributions
------------------	--

Warning

Will endlessly loop for periodic Markov chains

5.8.3.2 MarkovTrajectory * MarkovProcess::computeTrajectory (int *time*, int *length*)

Compute a possible trajectory of the Markov chain.

Parameters

<i>length</i>	: Length of the trajectory
---------------	----------------------------

5.8.3.3 void MarkovProcess::setDistribution (double * array)

Set the initial distribution.

Parameters

<i>array</i>	: An array of probabilities (summing to 1) with the size of the Markov chain state space
--------------	--

5.8.3.4 void MarkovProcess::setTransition (double * array)

Set the transition kernel.

Parameters

<i>array</i>	: A 2D-array of probabilities (summing to 1 within each row) with the square of the size of the Markov chain state space
--------------	--

5.8.3.5 void MarkovProcess::setTransition (int i, double * array)

Set one line of the transition kernel.

Parameters

<i>j</i>	: The index of the row to be set
<i>array</i>	: An array of probabilities (summing to 1) with the size of the Markov chain state space

5.8.4 Member Data Documentation

5.8.4.1 double* MarkovProcess::distribution

The initial probability distribution of the system state (time 0)

5.8.4.2 std::vector<double*>* MarkovProcess::distributions

A vector of probability distributions through time (from 0 to lastTime)

5.8.4.3 int MarkovProcess::lastDelay

The delay of the furthest computed transition kernel in the transitions vector

5.8.4.4 int MarkovProcess::lastTime

The time of the furthest computed probability distribution in the distributions vector

5.8.4.5 int MarkovProcess::size

The size of the Markov chain state space

5.8.4.6 `double* MarkovProcess::transition`

The transition kernel of the Markov chain (1 step)

5.8.4.7 `std::vector<double*>* MarkovProcess::transitions`

A vector of transition kernels for several steps (from 1 to lastDelay)

The documentation for this class was generated from the following files:

- [/home/lamarche/programming/multilevel_prediction/src/markov_process.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/markov_process.cpp](#)

5.9 MarkovTrajectory Class Reference

Public Member Functions

- **MarkovTrajectory** ([MarkovProcess](#) *process, int time, int length)
- void **print** (int binary=0)

Public Attributes

- [MarkovProcess](#) * **process**
- int **time**
- int **length**
- int * **states**

The documentation for this class was generated from the following files:

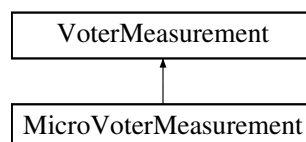
- [/home/lamarche/programming/multilevel_prediction/src/markov_process.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/markov_process.cpp](#)

5.10 MicroVoterMeasurement Class Reference

A measurement consisting in one probe for each node of the interaction graph.

```
#include <voter_graph.hpp>
```

Inheritance diagram for MicroVoterMeasurement:



Public Member Functions

- **MicroVoterMeasurement** ([VoterGraph](#) *graph, std::set< [VoterMetric](#) > metric)
Constructor.

Additional Inherited Members

5.10.1 Detailed Description

A measurement consisting in one probe for each node of the interaction graph.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 MicroVoterMeasurement::MicroVoterMeasurement (VoterGraph * *graph*, std::set< VoterMetric > *metric*)

Constructor.

Parameters

<i>graph</i>	: The interaction graph to be observed
<i>metric</i>	: The set of metric associated to the micro-probe

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.cpp](#)

5.11 OrderedPartition Class Reference

Public Member Functions

- **OrderedPartition** (int s, double p)
- void **print** ()

Public Attributes

- int **microSize**
- int * **optimalCut**
- double **param**
- double **beta**
- std::string **string**
- double **entropy**
- double **information**

The documentation for this class was generated from the following file:

- /home/lamarche/programming/multilevel_prediction/src/[partition.hpp](#)

5.12 Part Class Reference

A part of a partition (i.e., a set of individuals)

```
#include <partition.hpp>
```

Public Member Functions

- [Part](#) ()
- **Part** ([Part](#) *part)
- void **addIndividual** (int i, bool front=false, int value=-1)
- bool **contains** (int i)
- bool **equal** ([Part](#) *p)
- void **print** (bool endl=false)
- int **printSize** ()

Public Attributes

- int **id**
- int **size**
- int **value**
- std::list< int > * **individuals**

5.12.1 Detailed Description

A part of a partition (i.e., a set of individuals)

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Part::Part ()

Author

Robin Lamarche-Perrin

Date

22/01/2015

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[partition.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/partition.cpp

5.13 Partition Class Reference

A partition (i.e., a set of disjoint and covering parts)

```
#include <partition.hpp>
```

Public Member Functions

- **Partition** ([Partition](#) *partition)
- void **addPart** ([Part](#) *p, bool front=false)
- [Part](#) * **findPart** (int individual)
- [Part](#) * **getPartFromValue** (int value)
- bool **equal** ([Partition](#) *p)
- void **print** (bool endl=false)

Public Attributes

- int **size**
- std::list< [Part](#) * > * **parts**

5.13.1 Detailed Description

A partition (i.e., a set of disjoint and covering parts)

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[partition.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/partition.cpp

5.14 Timer Class Reference

A timer mesuring computation times and memory consumption during the program execution.

```
#include <timer.hpp>
```

Public Member Functions

- **Timer** (char *file=0)
- void **start** (int size, std::string text="")
- void **startTime** ()
- void **startMemory** ()
- void **stop** (std::string text="")
- void **stopTime** ()
- void **stopMemory** ()
- void **step** (std::string text="")

5.14.1 Detailed Description

A timer mesuring computation times and memory consumption during the program execution.

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[timer.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/timer.cpp

5.15 TwoCommunitiesExperiment Class Reference

Programming a prediction experiment on a two-communities Voter Model.

```
#include <voter_experiment.hpp>
```

Public Member Functions

- [TwoCommunitiesExperiment](#) (int size1, int size2, double intraR1, double intraR2, double interR1, double interR2, double contrarian1, double contrarian2, double time, double delay, SpecMeasurementSet *[preMeasurements](#), SpecMeasurementSet *[postMeasurements](#))
Constructor.
- [~TwoCommunitiesExperiment](#) ()
Destructor.

Public Attributes

- int `id`
- `UpdateProcess` `update`
- double `threshold`
- bool `compactModel`
- bool `withAggregation`
- bool `partDecomposition`
- int `size1Min`
- int `size1Max`
- int `size1Step`
- int `size2Min`
- int `size2Max`
- int `size2Step`
- bool `equalSize`
- double `intraR1Min`
- double `intraR1Max`
- double `intraR1Step`
- double `intraR2Min`
- double `intraR2Max`
- double `intraR2Step`
- double `interR1Min`
- double `interR1Max`
- double `interR1Step`
- double `interR2Min`
- double `interR2Max`
- double `interR2Step`
- bool `equalIntraRate`
- bool `equalInterRate`
- bool `oppositeInterRate`
- double `contrarian1Min`
- double `contrarian1Max`
- double `contrarian1Step`
- double `contrarian2Min`
- double `contrarian2Max`
- double `contrarian2Step`
- bool `equalContrarian`
- int `timeMin`
- int `timeMax`
- int `timeStep`
- int `delayMin`
- int `delayMax`
- int `delayStep`
- `SpecMeasurementSet` * `preMeasurements`
- `SpecMeasurementSet` * `postMeasurements`

Static Public Attributes

- static int `id_number` = 0

5.15.1 Detailed Description

Programming a prediction experiment on a two-communities Voter Model.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 TwoCommunitiesExperiment::TwoCommunitiesExperiment (int *size1*, int *size2*, double *intraR1*, double *intraR2*, double *interR1*, double *interR2*, double *contrarian1*, double *contrarian2*, double *time*, double *delay*, SpecMeasurementSet * *preMeasurements*, SpecMeasurementSet * *postMeasurements*)

Constructor.

Parameters

<i>size1</i>	: The size of community 1 in all experiments
<i>size2</i>	: The size of community 2 in all experiments
<i>intraR1</i>	: The weight of edges within community 1 in all experiments
<i>intraR2</i>	: The weight of edges within community 2 in all experiments
<i>interR1</i>	: The weight of edges from community 1 to community 2 in all experiments
<i>interR2</i>	: The weight of edges from community 2 to community 1 in all experiments
<i>contrarian1</i>	: The contrarian rate of nodes within community 1 in all experiments
<i>contrarian2</i>	: The contrarian rate of nodes within community 2 in all experiments
<i>time</i>	: The time of pre-measurement (-1 for stationary distribution) in all experiments
<i>delay</i>	: The delay before post-measurement in all experiments
<i>pre-Measurements</i>	: A set of pre-measurement for prediction
<i>post-Measurements</i>	: A set of post-measurement to be predicted

5.15.3 Member Data Documentation

5.15.3.1 bool TwoCommunitiesExperiment::compactModel

If true, the computed microscopic Markov chain is lumped according to the macro-state of community 1, the macro-state of community 2, and the state of the first agent in community 1

5.15.3.2 double TwoCommunitiesExperiment::contrarian1Max

The contrarian rate of nodes within community 1 in the last experiment

5.15.3.3 double TwoCommunitiesExperiment::contrarian1Min

The contrarian rate of nodes within community 1 in the first experiment

5.15.3.4 double TwoCommunitiesExperiment::contrarian1Step

The contrarian rate of nodes within community 1 between two consecutive experiments

5.15.3.5 double TwoCommunitiesExperiment::contrarian2Max

The contrarian rate of nodes within community 2 in the last experiment

5.15.3.6 double TwoCommunitiesExperiment::contrarian2Min

The contrarian rate of nodes within community 2 in the first experiment

5.15.3.7 double TwoCommunitiesExperiment::contrarian2Step

The contrarian rate of nodes within community 2 between two consecutive experiments

5.15.3.8 int TwoCommunitiesExperiment::delayMax

The delay before post-measurement in the last experiment

5.15.3.9 int TwoCommunitiesExperiment::delayMin

The delay before post-measurement in the first experiment

5.15.3.10 int TwoCommunitiesExperiment::delayStep

The delay before post-measurement between two consecutive experiments

5.15.3.11 bool TwoCommunitiesExperiment::equalContrarian

If true, the contrarian rate of nodes within community 2 is the same than the contrarian rate of nodes within community 1 in all experiments

5.15.3.12 bool TwoCommunitiesExperiment::equalInterRate

If true, the weight of edges from community 2 to community 1 is the same than the weight of edges from community 1 to community 2 in all experiments

5.15.3.13 bool TwoCommunitiesExperiment::equalIntraRate

If true, the weight of edges within community 2 is the same than the weight of edges within community 1 in all experiments

5.15.3.14 bool TwoCommunitiesExperiment::equalSize

If true, the size of community 2 is the same than the size of community 1 in all experiments

5.15.3.15 int TwoCommunitiesExperiment::id

A unique experiment number

5.15.3.16 int TwoCommunitiesExperiment::id_number = 0 [static]**Author**

Robin Lamarche-Perrin

Date

22/01/2015

5.15.3.17 double TwoCommunitiesExperiment::interR1Max

The weight of edges from community 1 to community 2 in the last experiment

5.15.3.18 double TwoCommunitiesExperiment::interR1Min

The weight of edges from community 1 to community 2 in the first experiment

5.15.3.19 double TwoCommunitiesExperiment::interR1Step

The weight of edges from community 1 to community 2 between two consecutive experiments

5.15.3.20 double TwoCommunitiesExperiment::interR2Max

The weight of edges from community 2 to community 1 in the last experiment

5.15.3.21 double TwoCommunitiesExperiment::interR2Min

The weight of edges from community 2 to community 1 in the first experiment

5.15.3.22 double TwoCommunitiesExperiment::interR2Step

The weight of edges from community 2 to community 1 between two consecutive experiments

5.15.3.23 double TwoCommunitiesExperiment::intraR1Max

The weight of edges within community 1 in the last experiment

5.15.3.24 double TwoCommunitiesExperiment::intraR1Min

The weight of edges within community 1 in the first experiment

5.15.3.25 double TwoCommunitiesExperiment::intraR1Step

The weight of edges within community 1 between two consecutive experiments

5.15.3.26 double TwoCommunitiesExperiment::intraR2Max

The weight of edges within community 2 in the last experiment

5.15.3.27 double TwoCommunitiesExperiment::intraR2Min

The weight of edges within community 2 in the first experiment

5.15.3.28 double TwoCommunitiesExperiment::intraR2Step

The weight of edges within community 2 between two consecutive experiments

5.15.3.29 bool TwoCommunitiesExperiment::oppositeInterRate

If true, the weight of edges from community 2 to community 1 is the opposite of the weight of edges from community 1 to community 2 in all experiments ($w_2 = 1 - w_1$)

5.15.3.30 SpecMeasurementSet* TwoCommunitiesExperiment::postMeasurements

A set of post-measurement to be predicted

5.15.3.31 SpecMeasurementSet* TwoCommunitiesExperiment::preMeasurements

A set of pre-measurement for prediction

5.15.3.32 int TwoCommunitiesExperiment::size1Max

The size of community 1 in the last experiment

5.15.3.33 int TwoCommunitiesExperiment::size1Min

The size of community 1 in the first experiment

5.15.3.34 int TwoCommunitiesExperiment::size1Step

The size of community 1 between two consecutive experiments

5.15.3.35 int TwoCommunitiesExperiment::size2Max

The size of community 2 in the last experiment

5.15.3.36 int TwoCommunitiesExperiment::size2Min

The size of community 2 in the first experiment

5.15.3.37 int TwoCommunitiesExperiment::size2Step

The size of community 2 between two consecutive experiments

5.15.3.38 double TwoCommunitiesExperiment::threshold

The precision threshold to compute the stationary distribution of the Markov chain (see computeStationary-Distribution method in [MarkovProcess](#) class)

5.15.3.39 int TwoCommunitiesExperiment::timeMax

The time of pre-measurement (-1 for stationary distribution) in the last experiment

5.15.3.40 int TwoCommunitiesExperiment::timeMin

The time of pre-measurement (-1 for stationary distribution) in the first experiment

5.15.3.41 int TwoCommunitiesExperiment::timeStep

The time of pre-measurement (-1 for stationary distribution) between two consecutive experiments

5.15.3.42 UpdateProcess TwoCommunitiesExperiment::update

The update process of the built Voter Model

The documentation for this class was generated from the following files:

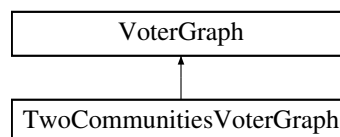
- [/home/lamarche/programming/multilevel_prediction/src/voter_experiment.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/voter_experiment.cpp](#)

5.16 TwoCommunitiesVoterGraph Class Reference

An interaction graph consisting in two communities of nodes (complete graph within each community, complete interaction between the two communities, possibly with different weights)

```
#include <voter_graph.hpp>
```

Inheritance diagram for TwoCommunitiesVoterGraph:



Public Member Functions

- [TwoCommunitiesVoterGraph](#) (int [size1](#), int [size2](#), double [intraRate1](#), double [intraRate2](#), double [interRate1](#), double [interRate2](#), double [contrarian1](#), double [contrarian2](#), int [update=UPDATE_EDGES](#))
Constructor.
- [~TwoCommunitiesVoterGraph](#) ()
Destructor.
- [MarkovProcess](#) * [getCompactMarkovProcess](#) ()
Build the Markov chain associated to the graph, lumped according to the macro-state of community 1, the macro-state of community 2, and the state of the first agent in community 1.
- [Partition](#) * [getCompactMarkovPartition](#) ([VoterProbe](#) *probe, [VoterMetric](#) metric)
Build the partition of the lumped Markov chain state space (see [getCompactMarkovProcess](#)) associated to a probe with a given metric (e.g., [METRIC_MACRO_STATE](#) or [METRIC_ACTIVE_EDGES](#))
- [Partition](#) * [getCompactMarkovPartition](#) ([VoterMeasurement](#) *measurement)
Build the partition of the lumped Markov chain state space (see [getCompactMarkovProcess](#)) associated to a measurement (i.e., a set of probes)

Public Attributes

- int [size1](#)
- int [size2](#)
- double [intraRate1](#)
- double [intraRate2](#)
- double [interRate1](#)
- double [interRate2](#)
- double [contrarian1](#)
- double [contrarian2](#)
- std::set< [VoterNode](#) * > * [community1](#)
- std::set< [VoterNode](#) * > * [community2](#)

5.16.1 Detailed Description

An interaction graph consisting in two communities of nodes (complete graph within each community, complete interaction between the two communities, possibly with different weights)

5.16.2 Constructor & Destructor Documentation

5.16.2.1 `TwoCommunitiesVoterGraph::TwoCommunitiesVoterGraph (int size1, int size2, double intraRate1, double intraRate2, double interRate1, double interRate2, double contrarian1, double contrarian2, int update = UPDATE_EDGES)`

Constructor.

Parameters

<i>size1</i>	: The size of community 1
<i>size2</i>	: The size of community 2
<i>intraRate1</i>	: The weight of edges within community 1
<i>intraRate2</i>	: The weight of edges within community 2
<i>interRate1</i>	: The weight of edges from community 1 to community 2
<i>interRate2</i>	: The weight of edges from community 2 to community 1
<i>contrarian1</i>	: The contrarian rate of nodes in community 1
<i>contrarian2</i>	: The contrarian rate of nodes in community 2
<i>update</i>	: How the system evolves at each simulation step (UPDATE_NODES or UPDATE_EDGES)

5.16.3 Member Function Documentation

5.16.3.1 `Partition * TwoCommunitiesVoterGraph::getCompactMarkovPartition (VoterProbe * probe, VoterMetric metric)`

Build the partition of the lumped Markov chain state space (see getCompactMarkovProcess) associated to a probe with a given metric (e.g., METRIC_MACRO_STATE or METRIC_ACTIVE_EDGES)

Parameters

<i>probe</i>	: The probe used to partition the lumped state space
<i>metric</i>	: The metric of the probe (e.g., METRIC_MACRO_STATE or METRIC_ACTIVE_EDGES)

Returns

The computed partition over the lumped state space

5.16.3.2 `Partition * TwoCommunitiesVoterGraph::getCompactMarkovPartition (VoterMeasurement * measurement)`

Build the partition of the lumped Markov chain state space (see getCompactMarkovProcess) associated to a measurement (i.e., a set of probes)

Parameters

<i>measurement</i>	: The measurement used to partition the lumped state space
--------------------	--

Returns

The computed partition over the lumped state space

5.16.3.3 `MarkovProcess * TwoCommunitiesVoterGraph::getCompactMarkovProcess ()`

Build the Markov chain associated to the graph, lumped according to the macro-state of community 1, the macro-state of community 2, and the state of the first agent in community 1.

Returns

The computed lumped Markov chain

5.16.4 Member Data Documentation

5.16.4.1 `std::set<VoterNode*>* TwoCommunitiesVoterGraph::community1`

The set of nodes in community 1

5.16.4.2 `std::set<VoterNode*>* TwoCommunitiesVoterGraph::community2`

The set of nodes in community 2

5.16.4.3 `double TwoCommunitiesVoterGraph::contrarian1`

The contrarian rate of nodes in community 1

5.16.4.4 `double TwoCommunitiesVoterGraph::contrarian2`

The contrarian rate of nodes in community 2

5.16.4.5 `double TwoCommunitiesVoterGraph::interRate1`

The weight of edges from community 1 to community 2

5.16.4.6 `double TwoCommunitiesVoterGraph::interRate2`

The weight of edges from community 2 to community 1

5.16.4.7 `double TwoCommunitiesVoterGraph::intraRate1`

The weight of edges within community 1

5.16.4.8 `double TwoCommunitiesVoterGraph::intraRate2`

The weight of edges within community 2

5.16.4.9 `int TwoCommunitiesVoterGraph::size1`

The size of community 1

5.16.4.10 `int TwoCommunitiesVoterGraph::size2`

The size of community 2

The documentation for this class was generated from the following files:

- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp](#)

5.17 VoterBinning Class Reference

Public Member Functions

- **VoterBinning** ([VoterDataSet](#) *data)
- void **print** (bool verbose=false)

Public Attributes

- [VoterDataSet](#) * **data**
- int **size**
- int **binNumber**
- int * **cuts**
- double **score**

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.cpp](#)

5.18 VoterDataSet Class Reference

Public Member Functions

- **VoterDataSet** ([VoterGraph](#) *graph, int time, int delay, int trainSize, int testSize, int trainLength, int testLength)
- void **estimateTransitionMap** ([VoterMeasurement](#) *preM, [VoterMeasurement](#) *postM)
- void **printTransitionMap** ()
- double **getLogScore** ([VoterMeasurement](#) *preM, [VoterMeasurement](#) *postM, int prior=0)
- double **getQuadScore** ([VoterMeasurement](#) *preM, [VoterMeasurement](#) *postM, int prior=0)
- [VoterBinning](#) * **getOptimalBinning** ([VoterMeasurement](#) *preM, [VoterMeasurement](#) *postM, int prior=0, int realTrainSize=-1, bool verbose=false)

Public Attributes

- [VoterGraph](#) * **graph**
- int **time**
- int **delay**
- int **trainSize**
- int **testSize**
- int **trainLength**
- int **testLength**
- [VoterTrajectory](#) ** **trajectories**
- TransitionMap * **transMap**

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.cpp](#)

5.19 VoterEdge Class Reference

An edge of the interaction graph.

```
#include <voter_graph.hpp>
```

Public Member Functions

- [VoterEdge](#) ([VoterNode](#) *[node1](#), [VoterNode](#) *[node2](#), double [weight](#)=1)
Constructor.
- [~VoterEdge](#) ()
Destructor.

Public Attributes

- [VoterNode](#) * [node1](#)
- [VoterNode](#) * [node2](#)
- double [weight](#)

5.19.1 Detailed Description

An edge of the interaction graph.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 [VoterEdge::VoterEdge](#) ([VoterNode](#) * [node1](#), [VoterNode](#) * [node2](#), double [weight](#) = 1)

Constructor.

Parameters

<i>node1</i>	: Incoming node
<i>node2</i>	: Outcoming node
<i>weight</i>	: Determines the probability to select this edge (relatively to other edges) at each simulation step when the updateProcess variable of the graph is set to UPDATE_EDGES

5.19.3 Member Data Documentation

5.19.3.1 [VoterNode](#)* [VoterEdge::node1](#)

Incoming node

5.19.3.2 [VoterNode](#)* [VoterEdge::node2](#)

Outcoming node

5.19.3.3 double [VoterEdge::weight](#)

Determines the probability to select this edge (relatively to other edges) at each simulation step when the update-Process variable of the graph is set to UPDATE_EDGES

The documentation for this class was generated from the following files:

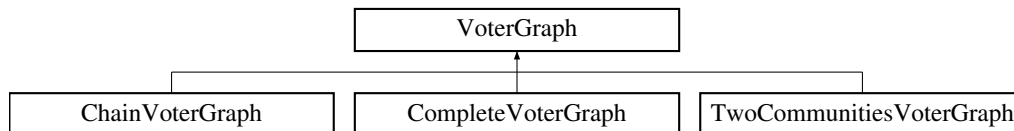
- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp](#)

5.20 VoterGraph Class Reference

The interaction graph describing a Voter Model.

```
#include <voter_graph.hpp>
```

Inheritance diagram for VoterGraph:



Public Member Functions

- [VoterGraph](#) (int update=[UPDATE_EDGES](#))
Constructor.
- virtual [~VoterGraph](#) ()
Destructor.
- void [print](#) ()
Print the graph structure and details.
- [VoterNode](#) * [addNode](#) (double weight=1, double contrarian=0)
Add a node to the graph.
- [VoterEdge](#) * [addEdge](#) ([VoterNode](#) *node1, [VoterNode](#) *node2, double weight=1)
Add an edge to the graph.
- void [fillEdges](#) ()
Add an edge between each pair of nodes in the graph (in both direction, with equal weight for each edge)
- [VoterNode](#) * [getRandomNode](#) ()
- [VoterNode](#) * [getUniformRandomNode](#) ()
- [VoterEdge](#) * [getRandomEdge](#) ([VoterNode](#) *node)
- [MarkovProcess](#) * [getMarkovProcess](#) ()
Build the Markov chain associated to the described Voter Model.
- [Partition](#) * [getMarkovPartition](#) ([VoterProbe](#) *probe, [VoterMetric](#) metric)
Build the partition of the Markov chain state space associated to a probe with a given metric (e.g., METRIC_MACRO_STATE or METRIC_ACTIVE_EDGES)
- [Partition](#) * [getMarkovPartition](#) ([VoterMeasurement](#) *measurement)
Build the partition of the Markov chain state space associated to a measurement (i.e., a set of probes)

Public Attributes

- int [updateProcess](#)
- int [complete](#)
- int [nodeNumber](#)
- int [edgeNumber](#)
- double [nodeWeight](#)
- double [edgeWeight](#)
- std::map< int, [VoterNode](#) * > * [nodeMap](#)
- std::set< [VoterNode](#) * > * [nodeSet](#)
- std::set< [VoterEdge](#) * > * [edgeSet](#)
- [MarkovProcess](#) * [process](#)

5.20.1 Detailed Description

The interaction graph describing a Voter Model.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 VoterGraph::VoterGraph (int *update* = UPDATE_EDGES)

Constructor.

Parameters

<i>update</i>	: How the system evolves at each simulation step (UPDATE_NODES or UPDATE_EDGES)
---------------	---

5.20.3 Member Function Documentation

5.20.3.1 VoterEdge * VoterGraph::addEdge (VoterNode * *node1*, VoterNode * *node2*, double *weight* = 1)

Add an edge to the graph.

Parameters

<i>node1</i>	: Incoming node
<i>node2</i>	: outgoing node
<i>weight</i>	: Determines the probability to select the edge to be added (relatively to other edges) at each simulation step when the updateProcess variable of the graph is set to UPDATE_EDGES

Returns

The added edge

5.20.3.2 VoterNode * VoterGraph::addNode (double *weight* = 1, double *contrarian* = 0)

Add a node to the graph.

Parameters

<i>weight</i>	: Determines the probability to select the node to be added (relatively to other nodes) at each simulation step when the updateProcess variable of the graph is set to UPDATE_NODES
<i>contrarian</i>	: The contrarian rate of the node to be added

Returns

The added node

5.20.3.3 Partition * VoterGraph::getMarkovPartition (VoterProbe * *probe*, VoterMetric *metric*)

Build the partition of the Markov chain state space associated to a probe with a given metric (e.g., METRIC_MACRO_STATE or METRIC_ACTIVE_EDGES)

Parameters

<i>probe</i>	: The probe used to partition the state space
<i>metric</i>	: The metric of the probe (e.g., METRIC_MACRO_STATE or METRIC_ACTIVE_EDGES)

Returns

The computed partition

5.20.3.4 Partition * VoterGraph::getMarkovPartition (VoterMeasurement * measurement)

Build the partition of the Markov chain state space associated to a measurement (i.e., a set of probes)

Parameters

<i>measurement</i>	: The measurement used to partition the state space
--------------------	---

Returns

The computed partition

5.20.3.5 MarkovProcess * VoterGraph::getMarkovProcess ()

Build the Markov chain associated to the described Voter Model.

Returns

The computed Markov chain

5.20.4 Member Data Documentation**5.20.4.1 int VoterGraph::edgeNumber**

The total number of edges in the graph

5.20.4.2 std::set<VoterEdge*> * VoterGraph::edgeSet

The set of all edges

5.20.4.3 double VoterGraph::edgeWeight

The sum of the weight of all edges

5.20.4.4 std::map<int,VoterNode*> * VoterGraph::nodeMap

The map of all nodes organized by id

5.20.4.5 int VoterGraph::nodeNumber

The total number of nodes in the graph

5.20.4.6 std::set<VoterNode*> * VoterGraph::nodeSet

The set of all nodes

5.20.4.7 double VoterGraph::nodeWeight

The sum of the weight of all nodes

5.20.4.8 MarkovProcess* VoterGraph::process

The Markov chain associated to the described Voter Model

5.20.4.9 int VoterGraph::updateProcess

How the system evolves at each simulation step (UPDATE_NODES or UPDATE_EDGES)

The documentation for this class was generated from the following files:

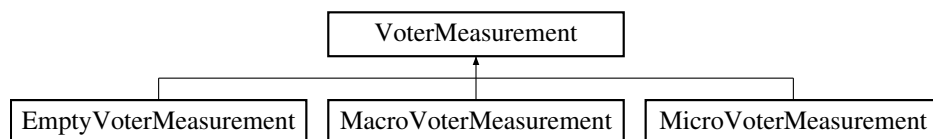
- /home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp
- /home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp

5.21 VoterMeasurement Class Reference

A measurement to observe the Voter Model according set of probes.

```
#include <voter_graph.hpp>
```

Inheritance diagram for VoterMeasurement:



Public Member Functions

- [VoterMeasurement](#) ([VoterGraph](#) *graph, std::string type)
Constructor.
- [~VoterMeasurement](#) ()
Destructor.
- void [addProbe](#) ([VoterProbe](#) *probe, [VoterMetric](#) metric, int binning=0)
Add a probe to the measurement.
- int [getCardinality](#) ()
- [VoterMeasurementState](#) * [getState](#) ([VoterState](#) *state)
- void [print](#) (bool endl=false)
Print the measurement details.

Public Attributes

- [VoterGraph](#) * graph
- std::string type
- [Partition](#) * partition
- int probeNumber
- std::map< int, [VoterProbe](#) * > * probeMap
- std::map< int, [VoterMetric](#) > * metricMap
- std::map< int, int > * binningMap

5.21.1 Detailed Description

A measurement to observe the Voter Model according set of probes.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 VoterMeasurement::VoterMeasurement (VoterGraph * *graph*, std::string *type*)

Constructor.

Parameters

<i>graph</i>	: The interaction graph to be observed
<i>type</i>	: The name of the measurement

5.21.3 Member Function Documentation

5.21.3.1 void VoterMeasurement::addProbe (VoterProbe * *probe*, VoterMetric *metric*, int *binning* = 0)

Add a probe to the measurement.

Parameters

<i>node</i>	: The probe to be added
<i>metric</i>	: The metric associated to the added probe (e.g., METRIC_MACRO_STATE or METRIC_ACTIVE_EDGES)

5.21.3.2 void VoterMeasurement::print (bool *endl* = false)

Print the measurement details.

Parameters

<i>endl</i>	: Line break after printing if true
-------------	-------------------------------------

5.21.4 Member Data Documentation

5.21.4.1 VoterGraph* VoterMeasurement::graph

The interaction graph to be observed

5.21.4.2 std::map<int,VoterMetric>* VoterMeasurement::metricMap

The map of metrics (e.g., METRIC_MACRO_STATE or METRIC_ACTIVE_EDGES) associated to each constituting probe organized by probe numbers

5.21.4.3 Partition* VoterMeasurement::partition

The partition of the Markov chain state space corresponding to the measurement

5.21.4.4 std::map<int,VoterProbe*>* VoterMeasurement::probeMap

The map of constituting probes organized by probe numbers

5.21.4.5 int VoterMeasurement::probeNumber

The number of probes constituting the measurement

5.21.4.6 std::string VoterMeasurement::type

The name of the measurement

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp
- /home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp

5.22 VoterMeasurementState Class Reference

Public Member Functions

- **VoterMeasurementState** ([VoterMeasurement](#) *measurement)
- void **init** (int value)
- bool **isEqual** ([VoterMeasurementState](#) *state)
- void **print** ()

Public Attributes

- [VoterMeasurement](#) * **measurement**
- int **size**
- int * **probeStates**

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp
- /home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp

5.23 VoterMeasurementTrajectory Class Reference

Public Member Functions

- **VoterMeasurementTrajectory** ([VoterMeasurement](#) *measurement, [VoterTrajectory](#) *trajectory)
- void **print** ()

Public Attributes

- [VoterMeasurement](#) * **measurement**
- [VoterGraph](#) * **graph**
- int **length**
- [VoterMeasurementState](#) ** **states**

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp
- /home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp

5.24 VoterNode Class Reference

A node of the interaction graph.

```
#include <voter_graph.hpp>
```

Public Member Functions

- `VoterNode` (int `id`, double `weight`=1, double `contrarian`=0)
Constructor.
- `~VoterNode` ()
Destructor.

Public Attributes

- int `id`
- double `weight`
- double `contrarian`
- int `inEdgeWeight`
- int `inEdgeNumber`
- `std::set< VoterEdge * > * inEdgeSet`
- int `outEdgeWeight`
- int `outEdgeNumber`
- `std::set< VoterEdge * > * outEdgeSet`

5.24.1 Detailed Description

A node of the interaction graph.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 `VoterNode::VoterNode (int i, double w = 1, double c = 0)`

Constructor.

Parameters

<i>id</i>	: Unique id within the graph
<i>weight</i>	: Determines the probability to select this node (relatively to other nodes) at each simulation step when the updateProcess variable of the graph is set to UPDATE_NODES
<i>contrarian</i>	: Contrarian rate of the node

Author

Robin Lamarche-Perrin

Date

22/01/2015

5.24.3 Member Data Documentation

5.24.3.1 `double VoterNode::contrarian`

Contrarian rate of the node

5.24.3.2 int VoterNode::id

Unique id within the graph

5.24.3.3 int VoterNode::inEdgeNumber

Sum of the weight of incoming edges

5.24.3.4 std::set<VoterEdge*>* VoterNode::inEdgeSet

Set of incoming edges

5.24.3.5 int VoterNode::inEdgeWeight

Sum of the weight of incoming nodes

5.24.3.6 int VoterNode::outEdgeNumber

Sum of the weight of outgoing edges

5.24.3.7 std::set<VoterEdge*>* VoterNode::outEdgeSet

Set of outgoing edges

5.24.3.8 int VoterNode::outEdgeWeight

Sum of the weight of outgoing nodes

5.24.3.9 double VoterNode::weight

Determines the probability to select this node (relatively to other nodes) at each simulation step when the update-Process variable of the graph is set to UPDATE_NODES

The documentation for this class was generated from the following files:

- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp](#)
- [/home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp](#)

5.25 VoterProbe Class Reference

A probe to observe the Voter Model according to a subset of nodes.

```
#include <voter_graph.hpp>
```

Public Member Functions

- [VoterProbe \(VoterGraph *graph\)](#)
Constructor.
- [~VoterProbe \(\)](#)
Destructor.

- void **setNodeSet** (std::set< [VoterNode](#) * > *set)
Set the set of observed nodes.
- void **addNode** ([VoterNode](#) *node)
Add an observed node to the probe.
- void **addNodes** (unsigned long int i)
Add a set of observed nodes to the probe.
- int **getCardinality** ([VoterMetric](#) metric, int binning=0)
- int **getState** ([VoterState](#) *state, [VoterMetric](#) metric, int binning=0)
- void **print** (bool endl=false)
Print the probe details.

Public Attributes

- [VoterGraph](#) * graph
- int [nodeNumber](#)
- std::set< [VoterNode](#) * > * [nodeSet](#)

5.25.1 Detailed Description

A probe to observe the Voter Model according to a subset of nodes.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 VoterProbe::VoterProbe ([VoterGraph](#) * graph)

Constructor.

Parameters

<i>graph</i>	: The interaction graph to be observed
--------------	--

5.25.3 Member Function Documentation

5.25.3.1 void VoterProbe::addNode ([VoterNode](#) * node)

Add an observed node to the probe.

Parameters

<i>node</i>	: The node to be added
-------------	------------------------

5.25.3.2 void VoterProbe::addNodes (unsigned long int i)

Add a set of observed nodes to the probe.

Parameters

<i>graph</i>	: A binary number indicating for each node of the graph if it should (1) or should not (0) be added (the nodes are ordered according to their unique id)
--------------	--

5.25.3.3 void VoterProbe::print (bool endl = false)

Print the probe details.

Parameters

<i>endl</i>	: Line break after printing if true
-------------	-------------------------------------

5.25.3.4 void VoterProbe::setNodeSet (std::set< VoterNode * > * set)

Set the set of observed nodes.

Parameters

<i>node</i>	: The set to be associated
-------------	----------------------------

5.25.4 Member Data Documentation

5.25.4.1 VoterGraph* VoterProbe::graph

The interaction graph to be observed

5.25.4.2 int VoterProbe::nodeNumber

The number of observed nodes

5.25.4.3 std::set<VoterNode*>* VoterProbe::nodeSet

The set of observed nodes

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.cpp](#)

5.26 VoterState Class Reference

Public Member Functions

- **VoterState** ([VoterGraph](#) *graph)
- **VoterState** ([VoterState](#) *state)
- void **print** ()
- void **setFromMicroUniform** ()
- void **setFromMacroUniform** ()
- [VoterState](#) * **getNextState** ()

Public Attributes

- [VoterGraph](#) * **graph**
- int **size**
- bool * **agentStates**

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.cpp](#)

5.27 VoterTrajectory Class Reference

Public Member Functions

- **VoterTrajectory** ([VoterGraph](#) *graph, int time, int length)
- void **print** ()

Public Attributes

- [VoterGraph](#) * **graph**
- int **time**
- int **length**
- [VoterState](#) ** **states**

The documentation for this class was generated from the following files:

- /home/lamarche/programming/multilevel_prediction/src/[voter_graph.hpp](#)
- /home/lamarche/programming/multilevel_prediction/src/voter_graph.cpp

Chapter 6

File Documentation

6.1 /home/lamarche/programming/multilevel_prediction/src/csv_tools.hpp File Reference

Tools to handle input and output CSV files.

```
#include <vector>
#include <string>
#include <time.h>
```

Typedefs

- typedef std::vector< std::string > **CSVLine**

Functions

- void **deleteCSV** (std::string fileName)
- void **openInputCSV** (std::ifstream &file, std::string fileName)
- bool **isInputCSVEmpy** (std::ifstream &file)
- bool **hasCSVLine** (std::ifstream &file)
- void **getCSVLine** (std::ifstream &file, CSVLine &line, int sizeMax=8)
- void **printCSVLine** (CSVLine &line)
- void **parseCSVFile** (std::string fileName)
- int **getCSVSize** (std::string fileName)
- void **closeInputCSV** (std::ifstream &file)
- void **openOutputCSV** (std::ofstream &file, std::string fileName, bool erase=false)
- void **addCSVLine** (std::ofstream &file, CSVLine &line)
- void **addCSVField** (std::ofstream &file, int field, bool endField=true)
- void **addCSVField** (std::ofstream &file, double field, bool endField=true, int prec=0)
- void **addCSVField** (std::ofstream &file, std::string field, bool endField=true)
- void **addCSVNAField** (std::ofstream &file, bool endField=true)
- void **addCSVNULLField** (std::ofstream &file, bool endField=true)
- void **endCSVField** (std::ofstream &file)
- void **endCSVLine** (std::ofstream &file)
- void **closeOutputCSV** (std::ofstream &file)
- double **string2double** (std::string str)
- std::string **int2string** (int value)
- std::string **float2string** (float value, int prec=10)
- std::string **double2string** (double value, int prec=10)
- time_t **date2time** (std::string date)

Variables

- bool **VERBOSE**
- int **VERBOSE_TAB**
- const bool **CSV_QUOTES** = false
- const char **QUOTE_CHAR** = '"'
- const char **ESCAPE_CHAR** = '\\'
- const char **FIELD_DELIM** = ','
- const char **LINE_DELIM** = '\n'
- const int **DATE_INDEX** = 3
- const int **TITLE_INDEX** = 4
- const int **BODY_INDEX** = 5
- const int **MAX_INDEX** = 17

6.1.1 Detailed Description

Tools to handle input and output CSV files.

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.1.2 Function Documentation

6.1.2.1 void deleteCSV (std::string *fileName*)

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.1.3 Variable Documentation

6.1.3.1 bool VERBOSE

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.2 /home/lamarche/programming/multilevel_prediction/src/main.hpp File Reference

Main program to run tests and experiments (see for example class VoterExperiment)

```
#include "csv_tools.hpp"
#include "voter_graph.hpp"
#include "markov_process.hpp"
```


Functions

- void **computeInformationMeasures** ()
- void **testScoreFunctions** ()
- void **testMarkovProcess** ()
- void **testVoterGraph** ()
- void **testMeasuresWithAggregation** ()
- long unsigned int **nChoosek** (int n, int k)

6.2.1 Detailed Description

Main program to run tests and experiments (see for example class VoterExperiment)

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.3 /home/lamarche/programming/multilevel_prediction/src/markov_process.hpp File Reference

Class to build a finite Markov chain.

```
#include <vector>
#include "partition.hpp"
```

Classes

- class [MarkovProcess](#)
A finite Markov chain described by a discrete state space, an initial distribution, and a transition kernel.
- class [MarkovTrajectory](#)
- class [MarkovDataSet](#)

6.3.1 Detailed Description

Class to build a finite Markov chain.

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.4 /home/lamarche/programming/multilevel_prediction/src/partition.hpp File Reference

Classes to build partitions of the state space of Markov chains.

```
#include <set>
#include <list>
```

Classes

- class [Part](#)
A part of a partition (i.e., a set of individuals)
- class [Partition](#)
A partition (i.e., a set of disjoint and covering parts)
- class [OrderedPartition](#)

Typedefs

- typedef std::set< [Part](#) * > **PartSet**
- typedef std::list< [Partition](#) * > **PartitionList**

6.4.1 Detailed Description

Classes to build partitions of the state space of Markov chains.

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.5 /home/lamarche/programming/multilevel_prediction/src/timer.hpp File Reference

Tools to mesure computation times and memory consumption during the program execution.

```
#include <list>
#include <time.h>
```

Classes

- struct [DataPointStruct](#)
The computation time and memory consumption associated to an experiment of a given size.
- class [Timer](#)
A timer mesuring computation times and memory consumption during the program execution.

Typedefs

- typedef struct [DataPointStruct](#) [DataPoint](#)
The computation time and memory consumption associated to an experiment of a given size.

Functions

- int **getMemory** ()

6.5.1 Detailed Description

Tools to mesure computation times and memory consumption during the program execution.

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.6 /home/lamarche/programming/multilevel_prediction/src/voter_experiment.hpp File Reference

Tools to run prediction experiments on Voter Models.

```
#include <fstream>
#include "csv_tools.hpp"
#include "partition.hpp"
#include "voter_graph.hpp"
#include "markov_process.hpp"
```

Classes

- class [TwoCommunitiesExperiment](#)
Programming a prediction experiment on a two-communities Voter Model.
- class [ChainExperiment](#)

Typedefs

- typedef std::set
< [TwoCommunitiesExperiment](#) * > **TwoCommunitiesExperimentSet**
- typedef std::set
< [ChainExperiment](#) * > **ChainExperimentSet**

Functions

- void [twoCommunitiesExperiment](#) (TwoCommunitiesExperimentSet *expSet, std::string fileName)
Run a set of programmed experiments.
- void **chainExperiment** (ChainExperimentSet *expSet, std::string fileName)
- void **addMeasurement** (MeasurementSet *set, [VoterGraph](#) *VG, [MeasurementType](#) type, [VoterMetric](#) metric)
- void **addMultiMeasurement** (MeasurementSet *set, [VoterGraph](#) *VG, [MeasurementType](#) type, [VoterMetric](#) metric)
- [VoterMeasurement](#) * **getMeasurement** ([VoterGraph](#) *VG, [MeasurementType](#) type, [VoterMetric](#) metric=[MA-CRO_STATE](#), int binning=0)
- void **addTwoCommunitiesHeaderToCSV** (std::string fileName)
- void **addTwoCommunitiesPartHeaderToCSV** (std::string fileName, std::string type)
- void **computeTwoCommunitiesMeasures** (std::ofstream &csvFile, [MarkovProcess](#) *MP, std::string type, int update, int size1, int size2, double intraR1, double intraR2, double interR1, double interR2, double contrarian1, double contrarian2, [VoterMeasurement](#) *preM, [VoterMeasurement](#) *postM, int time, int delay, [Partition](#) *microP)

- void **computeTwoCommunitiesMeasuresWithAggregation** (std::ofstream &csvFile, [MarkovProcess](#) *MP, std::string type, int update, int size1, int size2, double intraR1, double intraR2, double interR1, double interR2, double contrarian1, double contrarian2, [VoterMeasurement](#) *preM, [VoterMeasurement](#) *postM, int time, int delay, double threshold)
- void **computeTwoCommunitiesPartMeasures** (std::ofstream &csvFile, [MarkovProcess](#) *MP, int size1, int size2, [VoterMeasurement](#) *preM, [VoterMeasurement](#) *postM, int time, int delay)

6.6.1 Detailed Description

Tools to run prediction experiments on Voter Models.

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.6.2 Function Documentation

6.6.2.1 void twoCommunitiesExperiment (TwoCommunitiesExperimentSet * expSet, std::string fileName)

Run a set of programmed experiments.

Parameters

<i>expSet</i>	: Experiment set
<i>fileName</i>	: Output file name

6.7 /home/lamarche/programming/multilevel_prediction/src/voter_graph.hpp File Reference

Classes to build an interaction graph (nodes and edges) describing a Voter Model and some observation tools (probes and measurements)

```
#include <map>
#include <cstdlib>
#include "markov_process.hpp"
#include "partition.hpp"
```

Classes

- class [VoterNode](#)
A node of the interaction graph.
- class [VoterEdge](#)
An edge of the interaction graph.
- class [VoterGraph](#)
The interaction graph describing a Voter Model.
- class [CompleteVoterGraph](#)
An interaction graph with edges between each pair of nodes (in both direction, with equal weight for each edge)
- class [TwoCommunitiesVoterGraph](#)

An interaction graph consisting in two communities of nodes (complete graph within each community, complete interaction between the two communities, possibly with different weights)

- class [ChainVoterGraph](#)
- class [VoterProbe](#)

A probe to observe the Voter Model according to a subset of nodes.

- class [VoterMeasurement](#)

A measurement to observe the Voter Model according set of probes.

- class [MacroVoterMeasurement](#)

A measurement consisting in one probe observing all nodes of the interaction graph.

- class [MicroVoterMeasurement](#)

A measurement consisting in one probe for each node of the interaction graph.

- class [EmptyVoterMeasurement](#)

A measurement without any probe (no observation)

- class [VoterState](#)
- class [VoterMeasurementState](#)
- class [VoterTrajectory](#)
- class [VoterMeasurementTrajectory](#)
- class [VoterDataSet](#)
- class [VoterBinning](#)

Typedefs

- typedef std::set
 < [VoterMeasurement](#) * > **MeasurementSet**
- typedef std::set< std::pair
 < [MeasurementType](#), [VoterMetric](#) > > **SpecMeasurementSet**
- typedef std::map
 < [VoterMeasurementState](#) *, int * > **ProbabilityMap**
- typedef std::pair
 < **ProbabilityMap** *, int * > **ProbabilityPair**
- typedef std::map
 < [VoterMeasurementState](#)
 *, **ProbabilityPair** * > **TransitionMap**

Enumerations

- enum [VoterMetric](#) {
 [MACRO_STATE](#), [MAJORITY](#), [MAJ_1PC](#), [MAJ_2PC](#),
 [MAJ_3PC](#), [MAJ_4PC](#), [MAJ_5PC](#), [MAJ_6PC](#),
 [MAJ_7PC](#), [MAJ_8PC](#), [MAJ_9PC](#), [MAJ_10PC](#),
 [MAJ_20PC](#), [MAJ_30PC](#), [MAJ_40PC](#), [MAJ_50PC](#),
 [MAJ_60PC](#), [MAJ_70PC](#), [MAJ_80PC](#), [MAJ_90PC](#),
 [MAJ_2B](#), [MAJ_3B](#), [MAJ_4B](#), [MAJ_6B](#),
 [MAJ_8B](#), [MAJ_10B](#), [MAJ_12B](#), [MAJ_20B](#),
 [MAJ_40B](#), [ACTIVE_EDGES](#) }
- A metric associated to a probe.*
- enum [UpdateProcess](#) { [UPDATE_NODES](#), [UPDATE_EDGES](#) }
- The way a Voter Model evolves at each simulation step, by randomly choosing a node or an edge for interaction.*
- enum [MeasurementType](#) {
 [M_MICRO](#), [M_AGENT1](#), [M_MESO1](#), [M_MESO2](#),
 [M_MACRO](#), [M_EMPTY](#), [M_ALLSIZES1](#), [M_SOMESIZES1](#),
 [M_AGENT1_ALLSIZES1](#), [M_AGENT1_SOMESIZES1](#), [M_ALLNEIGHBORHOODS](#), [M_AGENT1_MESO1](#),
 [M_AGENT1_MESO2](#), [M_AGENT1_MACRO](#), [M_AGENT1_MESO1_MESO2](#), [M_MESO1_MESO2](#) }
- A specific measurement in the case of a two-communities interaction graphs.*

6.7.1 Detailed Description

Classes to build an interaction graph (nodes and edges) describing a Voter Model and some observation tools (probes and measurements)

Author

Robin Lamarche-Perrin

Date

22/01/2015

6.7.2 Enumeration Type Documentation

6.7.2.1 enum MeasurementType

A specific measurement in the case of a two-communities interaction graphs.

Enumerator

M_MICRO Microscopic state
M_AGENT1 State of the first node in community 1
M_MESO1 Aggregated state of all nodes in community 1
M_MESO2 Aggregated state of all nodes in community 2
M_MACRO Aggregated state of nodes in both communities
M_EMPTY No observation
M_ALLSIZES1 Aggregated state of node subsets of all sizes within community 1
M_SOMESIZES1 Aggregated state of node subsets of some sizes within community 1
M_AGENT1_ALLSIZES1 Join measurement (see above)
M_AGENT1_SOMESIZES1 Join measurement (see above)
M_AGENT1_MESO1 Join measurement (see above)
M_AGENT1_MESO2 Join measurement (see above)
M_AGENT1_MACRO Join measurement (see above)
M_AGENT1_MESO1_MESO2 Join measurement (see above)
M_MESO1_MESO2 Join measurement (see above)

6.7.2.2 enum UpdateProcess

The way a Voter Model evolves at each simulation step, by randomly choosing a node or an edge for interaction.

Enumerator

UPDATE_NODES Node-driven interactions: A node is chosen at each simulation step, it acts on one of its outgoing nodes
UPDATE_EDGES Edge-driven interactions: An edge is chosen at each simulation step, its incoming node acts on its outgoing node

6.7.2.3 enum VoterMetric

A metric associated to a probe.

Enumerator

MACRO_STATE The probe returns the number of observed nodes in state 1

MAJORITY The probe returns 0 (resp. 1) if the majority of agents are in state 0 (resp. 1), and NA if there is a strict equality

ACTIVE_EDGES The probe returns the probability that one of the observed nodes will change during the next simulation step

Index

/home/lamarche/programming/multilevel_prediction/src/csv-
_tools.hpp, 45

/home/lamarche/programming/multilevel_prediction/src/main-
hpp, 46

/home/lamarche/programming/multilevel_prediction/src/main-
_process.hpp, 47

/home/lamarche/programming/multilevel_prediction/src/partition-
hpp, 47

/home/lamarche/programming/multilevel_prediction/src/time-
hpp, 48

/home/lamarche/programming/multilevel_prediction/src/voter-
_experiment.hpp, 49

/home/lamarche/programming/multilevel_prediction/src/voter-
_graph.hpp, 50

ACTIVE_EDGES
voter_graph.hpp, 53

addEdge
VoterGraph, 34

addNode
VoterGraph, 34
VoterProbe, 41

addNodes
VoterProbe, 41

addProbe
VoterMeasurement, 37

ChainExperiment, 9
contrarianMax, 9
contrarianMin, 9
contrarianStep, 10
delayMax, 10
delayMin, 10
delayStep, 10
id, 10
postMeasurements, 10
preMeasurements, 10
threshold, 10
timeMax, 10
timeMin, 10
timeStep, 10
update, 10

ChainVoterGraph, 11

community1
TwoCommunitiesVoterGraph, 30

community2
TwoCommunitiesVoterGraph, 30

compactModel
TwoCommunitiesExperiment, 24

CompleteVoterGraph, 11

CompleteVoterGraph, 12

CompleteVoterGraph, 12

computeStationaryDistribution
MarkovProcess, 16

computeTrajectory
MarkovProcess, 16

contrarian
VoterNode, 39

contrarian1
TwoCommunitiesVoterGraph, 30

contrarian1Max
TwoCommunitiesExperiment, 24

contrarian1Min
TwoCommunitiesExperiment, 24

contrarian1Step
TwoCommunitiesExperiment, 24

contrarian2
TwoCommunitiesVoterGraph, 30

contrarian2Max
TwoCommunitiesExperiment, 24

contrarian2Min
TwoCommunitiesExperiment, 24

contrarian2Step
TwoCommunitiesExperiment, 24

contrarianMax
ChainExperiment, 9

contrarianMin
ChainExperiment, 9

contrarianStep
ChainExperiment, 10

csv_tools.hpp
deleteCSV, 46
VERBOSE, 46

DataPointStruct, 12

delayMax
ChainExperiment, 10
TwoCommunitiesExperiment, 24

delayMin
ChainExperiment, 10
TwoCommunitiesExperiment, 24

delayStep
ChainExperiment, 10
TwoCommunitiesExperiment, 25

deleteCSV
csv_tools.hpp, 46

distribution
MarkovProcess, 17

distributions
MarkovProcess, 17

- edgeNumber
 - VoterGraph, [35](#)
- edgeSet
 - VoterGraph, [35](#)
- edgeWeight
 - VoterGraph, [35](#)
- EmptyVoterMeasurement, [12](#)
 - EmptyVoterMeasurement, [13](#)
 - EmptyVoterMeasurement, [13](#)
- equalContrarian
 - TwoCommunitiesExperiment, [25](#)
- equalInterRate
 - TwoCommunitiesExperiment, [25](#)
- equalIntraRate
 - TwoCommunitiesExperiment, [25](#)
- equalSize
 - TwoCommunitiesExperiment, [25](#)
- getCompactMarkovPartition
 - TwoCommunitiesVoterGraph, [29](#)
- getCompactMarkovProcess
 - TwoCommunitiesVoterGraph, [29](#)
- getMarkovPartition
 - VoterGraph, [34](#), [35](#)
- getMarkovProcess
 - VoterGraph, [35](#)
- graph
 - VoterMeasurement, [37](#)
 - VoterProbe, [42](#)
- id
 - ChainExperiment, [10](#)
 - TwoCommunitiesExperiment, [25](#)
 - VoterNode, [39](#)
- id_number
 - TwoCommunitiesExperiment, [25](#)
- inEdgeNumber
 - VoterNode, [40](#)
- inEdgeSet
 - VoterNode, [40](#)
- inEdgeWeight
 - VoterNode, [40](#)
- interR1Max
 - TwoCommunitiesExperiment, [25](#)
- interR1Min
 - TwoCommunitiesExperiment, [25](#)
- interR1Step
 - TwoCommunitiesExperiment, [25](#)
- interR2Max
 - TwoCommunitiesExperiment, [26](#)
- interR2Min
 - TwoCommunitiesExperiment, [26](#)
- interR2Step
 - TwoCommunitiesExperiment, [26](#)
- interRate1
 - TwoCommunitiesVoterGraph, [30](#)
- interRate2
 - TwoCommunitiesVoterGraph, [30](#)
- intraR1Max
 - TwoCommunitiesExperiment, [26](#)
- intraR1Min
 - TwoCommunitiesExperiment, [26](#)
- intraR1Step
 - TwoCommunitiesExperiment, [26](#)
- intraR2Max
 - TwoCommunitiesExperiment, [26](#)
- intraR2Min
 - TwoCommunitiesExperiment, [26](#)
- intraR2Step
 - TwoCommunitiesExperiment, [26](#)
- intraRate1
 - TwoCommunitiesVoterGraph, [30](#)
- intraRate2
 - TwoCommunitiesVoterGraph, [30](#)
- lastDelay
 - MarkovProcess, [17](#)
- lastTime
 - MarkovProcess, [17](#)
- M_AGENT1
 - voter_graph.hpp, [52](#)
- M_AGENT1_ALLSIZES1
 - voter_graph.hpp, [52](#)
- M_AGENT1_MACRO
 - voter_graph.hpp, [52](#)
- M_AGENT1_MESO1
 - voter_graph.hpp, [52](#)
- M_AGENT1_MESO1_MESO2
 - voter_graph.hpp, [52](#)
- M_AGENT1_MESO2
 - voter_graph.hpp, [52](#)
- M_AGENT1_SOMESIZES1
 - voter_graph.hpp, [52](#)
- M_ALLSIZES1
 - voter_graph.hpp, [52](#)
- M_EMPTY
 - voter_graph.hpp, [52](#)
- M_MACRO
 - voter_graph.hpp, [52](#)
- M_MESO1
 - voter_graph.hpp, [52](#)
- M_MESO1_MESO2
 - voter_graph.hpp, [52](#)
- M_MESO2
 - voter_graph.hpp, [52](#)
- M_MICRO
 - voter_graph.hpp, [52](#)
- M_SOMESIZES1
 - voter_graph.hpp, [52](#)
- MACRO_STATE
 - voter_graph.hpp, [53](#)
- MAJORITY
 - voter_graph.hpp, [53](#)
- MacroVoterMeasurement, [13](#)
 - MacroVoterMeasurement, [14](#)
 - MacroVoterMeasurement, [14](#)
- MarkovDataSet, [14](#)

- MarkovProcess, 14
 - computeStationaryDistribution, 16
 - computeTrajectory, 16
 - distribution, 17
 - distributions, 17
 - lastDelay, 17
 - lastTime, 17
 - MarkovProcess, 16
 - MarkovProcess, 16
 - setDistribution, 17
 - setTransition, 17
 - size, 17
 - transition, 17
 - transitions, 18
- MarkovTrajectory, 18
- MeasurementType
 - voter_graph.hpp, 52
- metricMap
 - VoterMeasurement, 37
- MicroVoterMeasurement, 18
 - MicroVoterMeasurement, 19
 - MicroVoterMeasurement, 19
- node1
 - VoterEdge, 32
- node2
 - VoterEdge, 32
- nodeMap
 - VoterGraph, 35
- nodeNumber
 - VoterGraph, 35
 - VoterProbe, 42
- nodeSet
 - VoterGraph, 35
 - VoterProbe, 42
- nodeWeight
 - VoterGraph, 35
- oppositeInterRate
 - TwoCommunitiesExperiment, 26
- OrderedPartition, 19
- outEdgeNumber
 - VoterNode, 40
- outEdgeSet
 - VoterNode, 40
- outEdgeWeight
 - VoterNode, 40
- Part, 19
 - Part, 20
- Partition, 20
- partition
 - VoterMeasurement, 37
- postMeasurements
 - ChainExperiment, 10
 - TwoCommunitiesExperiment, 26
- preMeasurements
 - ChainExperiment, 10
 - TwoCommunitiesExperiment, 26
- print
 - VoterMeasurement, 37
 - VoterProbe, 41
- probeMap
 - VoterMeasurement, 37
- probeNumber
 - VoterMeasurement, 37
- process
 - VoterGraph, 36
- setDistribution
 - MarkovProcess, 17
- setNodeSet
 - VoterProbe, 42
- setTransition
 - MarkovProcess, 17
- size
 - MarkovProcess, 17
- size1
 - TwoCommunitiesVoterGraph, 30
- size1Max
 - TwoCommunitiesExperiment, 27
- size1Min
 - TwoCommunitiesExperiment, 27
- size1Step
 - TwoCommunitiesExperiment, 27
- size2
 - TwoCommunitiesVoterGraph, 30
- size2Max
 - TwoCommunitiesExperiment, 27
- size2Min
 - TwoCommunitiesExperiment, 27
- size2Step
 - TwoCommunitiesExperiment, 27
- threshold
 - ChainExperiment, 10
 - TwoCommunitiesExperiment, 27
- timeMax
 - ChainExperiment, 10
 - TwoCommunitiesExperiment, 27
- timeMin
 - ChainExperiment, 10
 - TwoCommunitiesExperiment, 27
- timeStep
 - ChainExperiment, 10
 - TwoCommunitiesExperiment, 27
- Timer, 21
- transition
 - MarkovProcess, 17
- transitions
 - MarkovProcess, 18
- TwoCommunitiesExperiment, 21
 - compactModel, 24
 - contrarian1Max, 24
 - contrarian1Min, 24
 - contrarian1Step, 24
 - contrarian2Max, 24
 - contrarian2Min, 24

- contrarian2Step, 24
- delayMax, 24
- delayMin, 24
- delayStep, 25
- equalContrarian, 25
- equalInterRate, 25
- equalIntraRate, 25
- equalSize, 25
- id, 25
- id_number, 25
- interR1Max, 25
- interR1Min, 25
- interR1Step, 25
- interR2Max, 26
- interR2Min, 26
- interR2Step, 26
- intraR1Max, 26
- intraR1Min, 26
- intraR1Step, 26
- intraR2Max, 26
- intraR2Min, 26
- intraR2Step, 26
- oppositeInterRate, 26
- postMeasurements, 26
- preMeasurements, 26
- size1Max, 27
- size1Min, 27
- size1Step, 27
- size2Max, 27
- size2Min, 27
- size2Step, 27
- threshold, 27
- timeMax, 27
- timeMin, 27
- timeStep, 27
- TwoCommunitiesExperiment, 23
- TwoCommunitiesExperiment, 23
- update, 27
- twoCommunitiesExperiment
 - voter_experiment.hpp, 50
- TwoCommunitiesVoterGraph, 28
 - community1, 30
 - community2, 30
 - contrarian1, 30
 - contrarian2, 30
 - getCompactMarkovPartition, 29
 - getCompactMarkovProcess, 29
 - interRate1, 30
 - interRate2, 30
 - intraRate1, 30
 - intraRate2, 30
 - size1, 30
 - size2, 30
 - TwoCommunitiesVoterGraph, 29
 - TwoCommunitiesVoterGraph, 29
- type
 - VoterMeasurement, 38
- UPDATE_EDGES
 - voter_graph.hpp, 52
- UPDATE_NODES
 - voter_graph.hpp, 52
- update
 - ChainExperiment, 10
 - TwoCommunitiesExperiment, 27
- UpdateProcess
 - voter_graph.hpp, 52
- updateProcess
 - VoterGraph, 36
- VERBOSE
 - csv_tools.hpp, 46
- voter_graph.hpp
 - ACTIVE_EDGES, 53
 - M_AGENT1, 52
 - M_AGENT1_ALLSIZES1, 52
 - M_AGENT1_MACRO, 52
 - M_AGENT1_MESO1, 52
 - M_AGENT1_MESO1_MESO2, 52
 - M_AGENT1_MESO2, 52
 - M_AGENT1_SOMESIZES1, 52
 - M_ALLSIZES1, 52
 - M_EMPTY, 52
 - M_MACRO, 52
 - M_MESO1, 52
 - M_MESO1_MESO2, 52
 - M_MESO2, 52
 - M_MICRO, 52
 - M_SOMESIZES1, 52
 - MACRO_STATE, 53
 - MAJORITY, 53
 - UPDATE_EDGES, 52
 - UPDATE_NODES, 52
- voter_experiment.hpp
 - twoCommunitiesExperiment, 50
- voter_graph.hpp
 - MeasurementType, 52
 - UpdateProcess, 52
 - VoterMetric, 52
- VoterBinning, 31
- VoterDataSet, 31
- VoterEdge, 32
 - node1, 32
 - node2, 32
 - VoterEdge, 32
 - VoterEdge, 32
 - weight, 32
- VoterGraph, 33
 - addEdge, 34
 - addNode, 34
 - edgeNumber, 35
 - edgeSet, 35
 - edgeWeight, 35
 - getMarkovPartition, 34, 35
 - getMarkovProcess, 35
 - nodeMap, 35
 - nodeNumber, 35
 - nodeSet, 35

- nodeWeight, [35](#)
- process, [36](#)
- updateProcess, [36](#)
- VoterGraph, [34](#)
- VoterGraph, [34](#)
- VoterMeasurement, [36](#)
 - addProbe, [37](#)
 - graph, [37](#)
 - metricMap, [37](#)
 - partition, [37](#)
 - print, [37](#)
 - probeMap, [37](#)
 - probeNumber, [37](#)
 - type, [38](#)
 - VoterMeasurement, [37](#)
 - VoterMeasurement, [37](#)
- VoterMeasurementState, [38](#)
- VoterMeasurementTrajectory, [38](#)
- VoterMetric
 - voter_graph.hpp, [52](#)
- VoterNode, [39](#)
 - contrarian, [39](#)
 - id, [39](#)
 - inEdgeNumber, [40](#)
 - inEdgeSet, [40](#)
 - inEdgeWeight, [40](#)
 - outEdgeNumber, [40](#)
 - outEdgeSet, [40](#)
 - outEdgeWeight, [40](#)
 - VoterNode, [39](#)
 - VoterNode, [39](#)
 - weight, [40](#)
- VoterProbe, [40](#)
 - addNode, [41](#)
 - addNodes, [41](#)
 - graph, [42](#)
 - nodeNumber, [42](#)
 - nodeSet, [42](#)
 - print, [41](#)
 - setNodeSet, [42](#)
 - VoterProbe, [41](#)
 - VoterProbe, [41](#)
- VoterState, [42](#)
- VoterTrajectory, [43](#)
- weight
 - VoterEdge, [32](#)
 - VoterNode, [40](#)