Optimal Partition

Generated by Doxygen 1.8.6

Thu Nov 12 2015 17:01:44

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 9

4.1 File List . . . 9

5 Class Documentation 11

5.1 AbstractSet Class Reference . . . 11

5.1.1 Detailed Description . . . 12

5.1.2 Constructor & Destructor Documentation . . . 13

5.1.2.1 ~AbstractSet . . . 13

5.1.3 Member Function Documentation . . . 13

5.1.3.1 computeOptimalPartition . . . 13

5.1.3.2 getOptimalPartition . . . 13

5.1.3.3 getOptimalPartitionList . . . 13

5.1.3.4 printOptimalPartition . . . 14

5.1.3.5 printOptimalPartitionList . . . 15

5.1.3.6 printOptimalPartitionListInCSV . . . 15

5.1.3.7 setObjectiveFunction . . . 15

5.2 BiPart Class Reference . . . 15

5.2.1 Detailed Description . . . 16

5.3 BiSet Class Reference . . . 16

5.3.1 Member Function Documentation . . . 17

5.3.1.1 computeOptimalPartition . . . 17

5.3.1.2 getOptimalPartition . . . 17

5.3.1.3 printOptimalPartition . . . 17

5.3.1.4 setObjectiveFunction . . . 18

5.4 BiSubset Class Reference . . . 18

5.5 BottleneckObjectiveValue Class Reference . . . 19
5.6 ChainVoterGraph Class Reference . . . 19
5.7 CompleteGraph Class Reference . . . 20
5.8 CompleteVoterGraph Class Reference . . . 20
5.8.1 Detailed Description . . . 21
5.8.2 Constructor & Destructor Documentation . . . 21
5.8.2.1 CompleteVoterGraph . . . 21
5.9 DataPointStruct Struct Reference . . . 21
5.10 Dataset Class Reference . . . 21
5.11 Datatree Class Reference . . . 22
5.12 EmptyVoterMeasurement Class Reference . . . 23
5.12.1 Detailed Description . . . 23
5.12.2 Constructor & Destructor Documentation . . . 23
5.12.2.1 EmptyVoterMeasurement . . . 23
5.13 FiliformGraph Class Reference . . . 24
5.14 Graph Class Reference . . . 24
5.14.1 Member Function Documentation . . . 26
5.14.1.1 computeOptimalPartition . . . 26
5.14.1.2 getOptimalPartition . . . 26
5.14.1.3 printOptimalPartition . . . 26
5.14.1.4 setObjectiveFunction . . . 26
5.15 GraphBasedUniSet Class Reference . . . 26
5.16 GraphComponent Class Reference . . . 27
5.17 HHNode Class Reference . . . 28
5.18 HierarchicalHierarchicalSet Class Reference . . . 28
5.18.1 Member Function Documentation . . . 29
5.18.1.1 computeOptimalPartition . . . 29
5.18.1.2 getOptimalPartition . . . 29
5.18.1.3 printOptimalPartition . . . 30
5.18.1.4 setObjectiveFunction . . . 30
5.19 HierarchicalOrderedSet Class Reference . . . 30
5.19.1 Member Function Documentation . . . 31
5.19.1.1 computeOptimalPartition . . . 31
5.19.1.2 getOptimalPartition . . . 31
5.19.1.3 printOptimalPartition . . . 32
5.19.1.4 setObjectiveFunction . . . 32
5.20 HierarchicalSet Class Reference . . . 32
5.20.1 Member Function Documentation . . . 33
5.20.1.1 computeOptimalPartition . . . 33
5.20.1.2 getOptimalPartition . . . 33

5.20.1.3 printOptimalPartition . . . 33
5.20.1.4 setObjectiveFunction . . . 34
5.21 HierarchicalUniSet Class Reference . . . 34
5.21.1 Detailed Description . . . 34
5.21.2 Constructor & Destructor Documentation . . . 35
5.21.2.1 HierarchicalUniSet . . . 35
5.22 HNode Class Reference . . . 35
5.23 HONode Class Reference . . . 36
5.24 InformationBottleneck Class Reference . . . 36
5.25 LogarithmicScore Class Reference . . . 37
5.25.1 Detailed Description . . . 37
5.25.2 Constructor & Destructor Documentation . . . 38
5.25.2.1 LogarithmicScore . . . 38
5.26 LogarithmicScoreValue Class Reference . . . 38
5.27 MacroVoterMeasurement Class Reference . . . 39
5.27.1 Detailed Description . . . 39
5.27.2 Constructor & Destructor Documentation . . . 39
5.27.2.1 MacroVoterMeasurement . . . 39
5.28 MarkovDataSet Class Reference . . . 39
5.29 MarkovProcess Class Reference . . . 40
5.29.1 Detailed Description . . . 41
5.29.2 Constructor & Destructor Documentation . . . 41
5.29.2.1 MarkovProcess . . . 41
5.29.3 Member Function Documentation . . . 41
5.29.3.1 computeStationaryDistribution . . . 41
5.29.3.2 computeTrajectory . . . 42
5.29.3.3 setDistribution . . . 42
5.29.3.4 setTransition . . . 42
5.29.3.5 setTransition . . . 42
5.29.4 Member Data Documentation . . . 42
5.29.4.1 distribution . . . 42
5.29.4.2 distributions . . . 42
5.29.4.3 lastDelay . . . 43
5.29.4.4 lastTime . . . 43
5.29.4.5 size . . . 43
5.29.4.6 transition . . . 43
5.29.4.7 transitions . . . 43
5.30 MarkovTrajectory Class Reference . . . 43
5.31 MicroVoterMeasurement Class Reference . . . 43
5.31.1 Detailed Description . . . 44

5.31.2 Constructor & Destructor Documentation . . . 44
5.31.2.1 MicroVoterMeasurement . . . 44
5.32 MultiPart Class Reference . . . 44
5.32.1 Detailed Description . . . 45
5.33 MultiSet Class Reference . . . 45
5.33.1 Detailed Description . . . 46
5.33.2 Constructor & Destructor Documentation . . . 46
5.33.2.1 MultiSet . . . 46
5.33.2.2 MultiSet . . . 47
5.33.2.3 MultiSet . . . 47
5.33.2.4 ∼MultiSet . . . 47
5.33.3 Member Function Documentation . . . 47
5.33.3.1 computeOptimalPartition . . . 47
5.33.3.2 getAtomicMultiSubset . . . 47
5.33.3.3 getAtomicMultiSubset . . . 47
5.33.3.4 getOptimalPartition . . . 47
5.33.3.5 getRandomAtomicMultiSubset . . . 48
5.33.3.6 printOptimalPartition . . . 48
5.33.3.7 setObjectiveFunction . . . 48
5.34 MultiSubset Class Reference . . . 48
5.35 NHONode Class Reference . . . 49
5.36 NHOSet Class Reference . . . 50
5.36.1 Member Function Documentation . . . 51
5.36.1.1 computeOptimalPartition . . . 51
5.36.1.2 getOptimalPartition . . . 52
5.36.1.3 printOptimalPartition . . . 52
5.36.1.4 setObjectiveFunction . . . 52
5.37 NonconstrainedOrderedSet Class Reference . . . 52
5.37.1 Member Function Documentation . . . 53
5.37.1.1 computeOptimalPartition . . . 53
5.37.1.2 getOptimalPartition . . . 54
5.37.1.3 printOptimalPartition . . . 54
5.37.1.4 setObjectiveFunction . . . 54
5.38 NonconstrainedSet Class Reference . . . 54
5.38.1 Member Function Documentation . . . 55
5.38.1.1 computeOptimalPartition . . . 55
5.38.1.2 getOptimalPartition . . . 55
5.38.1.3 printOptimalPartition . . . 56
5.38.1.4 setObjectiveFunction . . . 56
5.39 ObjectiveFunction Class Reference . . . 56

5.39.1 Detailed Description . . . 57
5.39.2 Constructor & Destructor Documentation . . . 57
5.39.2.1 ObjectiveFunction . . . 57
5.40 ObjectiveValue Class Reference . . . 57
5.41 OrderedDatatree Class Reference . . . 58
5.42 OrderedPartition Class Reference . . . 59
5.43 OrderedSet Class Reference . . . 59
5.43.1 Member Function Documentation . . . 60
5.43.1.1 computeOptimalPartition . . . 60
5.43.1.2 getOptimalPartition . . . 60
5.43.1.3 printOptimalPartition . . . 61
5.43.1.4 setObjectiveFunction . . . 61
5.44 OrderedUniSet Class Reference . . . 61
5.44.1 Detailed Description . . . 61
5.44.2 Constructor & Destructor Documentation . . . 62
5.44.2.1 OrderedUniSet . . . 62
5.45 Part Class Reference . . . 62
5.45.1 Detailed Description . . . 63
5.46 Partition Class Reference . . . 63
5.46.1 Detailed Description . . . 63
5.47 PredictionDataset Class Reference . . . 63
5.47.1 Detailed Description . . . 64
5.47.2 Constructor & Destructor Documentation . . . 64
5.47.2.1 PredictionDataset . . . 64
5.47.3 Member Function Documentation . . . 65
5.47.3.1 addTestValue . . . 65
5.47.3.2 addTestValue . . . 65
5.47.3.3 addTrainValue . . . 65
5.47.3.4 addTrainValue . . . 65
5.47.3.5 print . . . 66
5.48 RandomGraph Class Reference . . . 66
5.49 RelativeEntropy Class Reference . . . 66
5.50 RelativeObjectiveValue Class Reference . . . 67
5.51 Ring Class Reference . . . 68
5.51.1 Member Function Documentation . . . 69
5.51.1.1 computeOptimalPartition . . . 69
5.51.1.2 getOptimalPartition . . . 69
5.51.1.3 printOptimalPartition . . . 69
5.51.1.4 setObjectiveFunction . . . 70
5.52 RingGraph Class Reference . . . 70

5.53 Timer Class Reference . . . 70
5.54 TreeToAdd Struct Reference . . . 71
5.55 TwoCommunitiesVoterGraph Class Reference . . . 71
5.55.1 Detailed Description . . . 72
5.55.2 Constructor & Destructor Documentation . . . 72
5.55.2.1 TwoCommunitiesVoterGraph . . . 72
5.55.3 Member Function Documentation . . . 72
5.55.3.1 getCompactMarkovPartition . . . 72
5.55.3.2 getCompactMarkovPartition . . . 73
5.55.3.3 getCompactMarkovProcess . . . 73
5.55.4 Member Data Documentation . . . 73
5.55.4.1 community1 . . . 73
5.55.4.2 community2 . . . 73
5.55.4.3 contrarian1 . . . 73
5.55.4.4 contrarian2 . . . 73
5.55.4.5 interRate1 . . . 73
5.55.4.6 interRate2 . . . 74
5.55.4.7 intraRate1 . . . 74
5.55.4.8 intraRate2 . . . 74
5.55.4.9 size1 . . . 74
5.55.4.10 size2 . . . 74
5.56 UniSet Class Reference . . . 74
5.56.1 Detailed Description . . . 75
5.56.2 Constructor & Destructor Documentation . . . 75
5.56.2.1 UniSet . . . 75
5.56.3 Member Function Documentation . . . 75
5.56.3.1 initReached . . . 75
5.57 UniSubset Class Reference . . . 75
5.57.1 Detailed Description . . . 76
5.57.2 Constructor & Destructor Documentation . . . 76
5.57.2.1 UniSubset . . . 76
5.58 VoterBinning Class Reference . . . 77
5.59 VoterDataSet Class Reference . . . 77
5.60 VoterEdge Class Reference . . . 78
5.60.1 Detailed Description . . . 78
5.60.2 Constructor & Destructor Documentation . . . 78
5.60.2.1 VoterEdge . . . 78
5.60.3 Member Data Documentation . . . 78
5.60.3.1 node1 . . . 78
5.60.3.2 node2 . . . 78

5.60.3.3 weight . . . 78
5.61 VoterGraph Class Reference . . . 79
5.61.1 Detailed Description . . . 80
5.61.2 Constructor & Destructor Documentation . . . 80
5.61.2.1 VoterGraph . . . 80
5.61.3 Member Function Documentation . . . 80
5.61.3.1 addEdge . . . 80
5.61.3.2 addNode . . . 80
5.61.3.3 getMarkovPartition . . . 80
5.61.3.4 getMarkovPartition . . . 81
5.61.3.5 getMarkovProcess . . . 81
5.61.4 Member Data Documentation . . . 81
5.61.4.1 edgeNumber . . . 81
5.61.4.2 edgeSet . . . 81
5.61.4.3 edgeWeight . . . 81
5.61.4.4 nodeMap . . . 81
5.61.4.5 nodeNumber . . . 81
5.61.4.6 nodeSet . . . 81
5.61.4.7 nodeWeight . . . 82
5.61.4.8 process . . . 82
5.61.4.9 updateProcess . . . 82
5.62 VoterMeasurement Class Reference . . . 82
5.62.1 Detailed Description . . . 83
5.62.2 Constructor & Destructor Documentation . . . 83
5.62.2.1 VoterMeasurement . . . 83
5.62.3 Member Function Documentation . . . 83
5.62.3.1 addProbe . . . 83
5.62.3.2 print . . . 83
5.62.4 Member Data Documentation . . . 83
5.62.4.1 graph . . . 83
5.62.4.2 metricMap . . . 83
5.62.4.3 partition . . . 83
5.62.4.4 probeMap . . . 84
5.62.4.5 probeNumber . . . 84
5.62.4.6 type . . . 84
5.63 VoterMeasurementState Class Reference . . . 84
5.64 VoterMeasurementTrajectory Class Reference . . . 84
5.65 VoterNode Class Reference . . . 85
5.65.1 Detailed Description . . . 85
5.65.2 Constructor & Destructor Documentation . . . 85

5.65.2.1 VoterNode . . . 85
5.65.3 Member Data Documentation . . . 86
5.65.3.1 contrarian . . . 86
5.65.3.2 id . . . 86
5.65.3.3 inEdgeNumber . . . 86
5.65.3.4 inEdgeSet . . . 86
5.65.3.5 inEdgeWeight . . . 86
5.65.3.6 outEdgeNumber . . . 86
5.65.3.7 outEdgeSet . . . 86
5.65.3.8 outEdgeWeight . . . 86
5.65.3.9 weight . . . 86
5.66 VoterProbe Class Reference . . . 86
5.66.1 Detailed Description . . . 87
5.66.2 Constructor & Destructor Documentation . . . 87
5.66.2.1 VoterProbe . . . 87
5.66.3 Member Function Documentation . . . 87
5.66.3.1 addNode . . . 87
5.66.3.2 addNodes . . . 87
5.66.3.3 print . . . 88
5.66.3.4 setNodeSet . . . 88
5.66.4 Member Data Documentation . . . 88
5.66.4.1 graph . . . 88
5.66.4.2 nodeNumber . . . 88
5.66.4.3 nodeSet . . . 88
5.67 VoterState Class Reference . . . 88
5.68 VoterTrajectory Class Reference . . . 89

6 File Documentation **91**

6.1 /home/lamarche/programming/optimal_partition/src/abstract_set.hpp File Reference . . . 91
6.1.1 Detailed Description . . . 91
6.2 /home/lamarche/programming/optimal_partition/src/logarithmic_score.hpp File Reference . . . 91
6.2.1 Detailed Description . . . 92
6.3 /home/lamarche/programming/optimal_partition/src/markov_process.hpp File Reference . . . 92
6.3.1 Detailed Description . . . 92
6.4 /home/lamarche/programming/optimal_partition/src/multi_set.hpp File Reference . . . 93
6.4.1 Detailed Description . . . 93
6.5 /home/lamarche/programming/optimal_partition/src/objective_function.hpp File Reference . . . 93
6.5.1 Detailed Description . . . 94
6.6 /home/lamarche/programming/optimal_partition/src/prediction_dataset.hpp File Reference . . . 94
6.6.1 Detailed Description . . . 94

6.7 /home/lamarche/programming/optimal_partition/src/uni_set.hpp File Reference 94
6.7.1 Detailed Description 95
6.8 /home/lamarche/programming/optimal_partition/src/voter_graph.hpp File Reference 95
6.8.1 Detailed Description 97
6.8.2 Enumeration Type Documentation 97
6.8.2.1 MeasurementType 97
6.8.2.2 UpdateProcess 98
6.8.2.3 VoterMetric 98

Index **99**

Chapter 1

Main Page

This program is a toolbox to solve special versions of the Set Partitioning Problem, that is the combinatorial optimisation of a decomposable objective over a set of feasible partitions (defined according to specific algebraic structures: e.g., hierachies, sets of intervals, graphs). The objectives are mainly based on information theory, in the perspective of multilevel analysis of large-scale datasets, and the algorithms are based on dynamic programming. For details regarding the formal grounds of this work, please refer to:

Robin Lamarche-Perrin, Yves Demazeau and Jean-Marc Vincent. A Generic Set Partitioning Algorithm with Applications to Hierarchical and Ordered Sets. Technical Report 105/2014, Max-Planck-Institute for Mathematics in the Sciences, Leipzig, Germany, May 2014.

http://www.mis.mpg.de/publications/preprints/2014/prepr2014-105.html

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

AbstractSet . . . 11
- BiSet . . . 16
- Graph . . . 24
  - CompleteGraph . . . 20
  - FiliformGraph . . . 24
  - RandomGraph . . . 66
  - RingGraph . . . 70
- HierarchicalHierarchicalSet . . . 28
- HierarchicalOrderedSet . . . 30
- HierarchicalSet . . . 32
- MultiSet . . . 45
- NHOSet . . . 50
- NonconstrainedOrderedSet . . . 52
- NonconstrainedSet . . . 54
- OrderedSet . . . 59
- Ring . . . 68

BiSubset . . . 18
DataPointStruct . . . 21
Dataset . . . 21
Datatree . . . 22
GraphComponent . . . 27
HHNode . . . 28
HNode . . . 35
HONode . . . 36
MarkovDataSet . . . 39
MarkovProcess . . . 40
MarkovTrajectory . . . 43
MultiSubset . . . 48
NHONode . . . 49
ObjectiveFunction . . . 56
- InformationBottleneck . . . 36
- LogarithmicScore . . . 37
- RelativeEntropy . . . 66

ObjectiveValue . . . 57
- BottleneckObjectiveValue . . . 19
- LogarithmicScoreValue . . . 38
- RelativeObjectiveValue . . . 67

OrderedDatatree . . . 58
OrderedPartition . . . 59
Part . . . 62
  BiPart . . . 15
  MultiPart . . . 44
Partition . . . 63
PredictionDataset . . . 63
Timer . . . 70
TreeToAdd . . . 71
UniSet . . . 74
  GraphBasedUniSet . . . 26
  HierarchicalUniSet . . . 34
  OrderedUniSet . . . 61
UniSubset . . . 75
VoterBinning . . . 77
VoterDataSet . . . 77
VoterEdge . . . 78
VoterGraph . . . 79
  ChainVoterGraph . . . 19
  CompleteVoterGraph . . . 20
  TwoCommunitiesVoterGraph . . . 71
VoterMeasurement . . . 82
  EmptyVoterMeasurement . . . 23
  MacroVoterMeasurement . . . 39
  MicroVoterMeasurement . . . 43
VoterMeasurementState . . . 84
VoterMeasurementTrajectory . . . 84
VoterNode . . . 85
VoterProbe . . . 86
VoterState . . . 88
VoterTrajectory . . . 89

Chapter 3

Class Index

3.1 Class List

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

AbstractSet
: Abstract class defining a set of elements that one wants to partition while optimising some (decomposable) objective and preserving some algebraic constraints (set of feasible parts) 11

BiPart
: A bi-part is a subset of a bi-dimensional set of elements (individuals) 15

BiSet 16

BiSubset 18

BottleneckObjectiveValue 19

ChainVoterGraph 19

CompleteGraph 20

CompleteVoterGraph
: An interaction graph with edges between each pair of nodes (in both direction, with equal weight for each edge) 20

DataPointStruct 21

Dataset 21

Datatree 22

EmptyVoterMeasurement
: A measurement without any probe (no observation) 23

FiliformGraph 24

Graph 24

GraphBasedUniSet 26

GraphComponent 27

HHNode 28

HierarchicalHierarchicalSet 28

HierarchicalOrderedSet 30

HierarchicalSet 32

HierarchicalUniSet
: A uni-dimensional set of elements structured according to a complete binary hierarchy, and such that the feasible subsets are all the nodes of the hierarchy 34

HNode 35

HONode 36

InformationBottleneck 36

LogarithmicScore
: Class to define and compute the logarithmic score function in the case of point prediction . . . 37

LogarithmicScoreValue 38

MacroVoterMeasurement
: A measurement consisting in one probe observing all nodes of the interaction graph 39

MarkovDataSet 39

MarkovProcess
A finite Markov chain described by a discrete state space, an initial distribution, and a transition kernel . . . 40
MarkovTrajectory . . . 43
MicroVoterMeasurement
A measurement consisting in one probe for each node of the interaction graph . . . 43
MultiPart
A multi-part is a subset of a multi-dimensional set of elements (individuals) . . . 44
MultiSet
A multi-dimensional set of elements based on the Cartesian product of several uni-dimensional sets (UniSet) and their algebraic structures (feasible subsets and feasible refinements) . . . 45
MultiSubset . . . 48
NHONode . . . 49
NHOSet . . . 50
NonconstrainedOrderedSet . . . 52
NonconstrainedSet . . . 54
ObjectiveFunction
Abstract class defining an objective function to be associated to a constrained set in order to define the optimisation problem that one wants to solve . . . 56
ObjectiveValue . . . 57
OrderedDatatree . . . 58
OrderedPartition . . . 59
OrderedSet . . . 59
OrderedUniSet
A uni-dimensional set of elements with a total order, and such that the feasible subsets are all the intervals induced by this order . . . 61
Part
A part is a subset of a set of elements (individuals) represented by integers . . . 62
Partition
A partition is a collection of pairwise-disjoint and covering subsets (parts) of a set of elements . 63
PredictionDataset
Class to represent a data set for the prediction of a post-measurement from the knowledge of a pre-measurement (composed of a train set and a test set) . . . 63
RandomGraph . . . 66
RelativeEntropy . . . 66
RelativeObjectiveValue . . . 67
Ring . . . 68
RingGraph . . . 70
Timer . . . 70
TreeToAdd . . . 71
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) . . 71
UniSet
A uni-dimensional set of elements and its algebraic structure (feasible subsets and feasible refinements) . . . 74
UniSubset
A feasible subset associated to a uni-dimensional set of elements (UniSet) . . . 75
VoterBinning . . . 77
VoterDataSet . . . 77
VoterEdge
An edge of the interaction graph . . . 78
VoterGraph
The interaction graph describing a Voter Model . . . 79
VoterMeasurement
A measurement to observe the Voter Model according set of probes . . . 82
VoterMeasurementState . . . 84
VoterMeasurementTrajectory . . . 84

VoterNode
A node of the interaction graph 85
VoterProbe
A probe to observe the Voter Model according to a subset of nodes 86
VoterState 88
VoterTrajectory 89

Chapter 4

File Index

4.1 File List

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

/home/lamarche/programming/optimal_partition/src/abstract_set.hpp
Abstract class defining a set of elements that one wants to partition while optimising some (decomposable) objective and preserving some algebraic constraints (set of feasible parts) 91
/home/lamarche/programming/optimal_partition/src/**bi_set.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**bidimensional_relative_entropy.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**check_graph_datatree.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**csv_tools.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**dataset.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**datatree.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**graph.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**hierarchical_hierarchical_set.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**hierarchical_ordered_set.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**hierarchical_set.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**information_bottleneck.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/logarithmic_score.hpp
Classes to define and compute the logarithmic score function in the case of point prediction . . 91
/home/lamarche/programming/optimal_partition/src/**main.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/markov_process.hpp
Class to build a finite Markov chain 92
/home/lamarche/programming/optimal_partition/src/multi_set.hpp
Classes to represent multi-dimensional sets of elements and their algebraic structure (feasible subsets and feasible refinements) 93
/home/lamarche/programming/optimal_partition/src/**NHO_set.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**nonconstrained_ordered_set.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**nonconstrained_set.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/objective_function.hpp
Abstract class defining an objective function to be associated to a constrained set in order to define the optimisation problem that one wants to solve 93
/home/lamarche/programming/optimal_partition/src/**orderedset.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**partition.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/prediction_dataset.hpp
Class to represent a data set for the prediction of a post-measurement from the knowledge of a pre-measurement (composed of a train set and a test set) 94
/home/lamarche/programming/optimal_partition/src/**prediction_programs.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**programs.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**relative_entropy.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**ring.hpp** . . . . ??
/home/lamarche/programming/optimal_partition/src/**timer.hpp** . . . . ??

/home/lamarche/programming/optimal_partition/src/uni_set.hpp
Some classes to represent uni-dimensional sets of elements and their algebraic structure (feasible subsets and feasible refinements) 94
/home/lamarche/programming/optimal_partition/src/voter_graph.hpp
Classes to build an interaction graph (nodes and edges) describing a Voter Model and some observation tools (probes and measurements) 95

Chapter 5

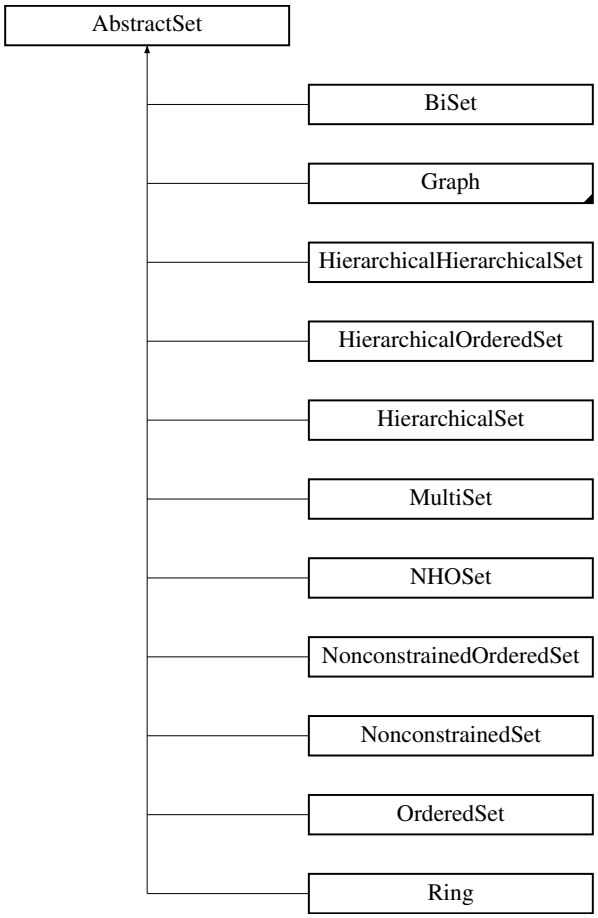
Class Documentation

5.1 AbstractSet Class Reference

Abstract class defining a set of elements that one wants to partition while optimising some (decomposable) objective and preserving some algebraic constraints (set of feasible parts)

`#include <abstract_set.hpp>`

Inheritance diagram for AbstractSet:



Public Member Functions

- virtual ~AbstractSet ()

  The objective that one wants to optimise (assumed to be decomposable: the objective of a partition is function of the objectives of its parts)
- virtual void setRandom ()=0

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- virtual void buildDataStructure ()=0

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- virtual void setObjectiveFunction (ObjectiveFunction *objective)=0

  Set the objective that one wants to optimise.
- virtual void print ()=0

  Print the set and its algebraic constraints.
- virtual void computeObjectiveValues ()=0

  *Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction *objective) should have been called first)*
- virtual void normalizeObjectiveValues ()=0

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- virtual void printObjectiveValues ()=0

  Print the value of the objective function for each feasible part.
- virtual void computeOptimalPartition (double parameter)=0

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- virtual Partition * getOptimalPartition (double parameter)=0

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- virtual void printOptimalPartition (double parameter)=0

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- PartitionList * getOptimalPartitionList (double threshold)

  Compute and return a list of partitions that fit with the algebraic constraints and that optimises the objective function that has been specified, while the parameter of the objective function varies on a proper ranged (defined by the objective itself)
- void printOptimalPartitionList (double threshold)

  Compute and print a list of partitions that fit with the algebraic constraints and that optimises the objective function that has been specified, while the parameter of the objective function varies on a proper ranged (defined by the objective itself)
- void printOptimalPartitionListInCSV (double threshold, Dataset *data, int dim, std::string fileName)

  Compute and print in a CSV file a list of partitions that fit with the algebraic constraints and that optimises the objective function that has been specified, while the parameter of the objective function varies on a proper ranged (defined by the objective itself)

Public Attributes

- ObjectiveFunction * **objective**

5.1.1 Detailed Description

Abstract class defining a set of elements that one wants to partition while optimising some (decomposable) objective and preserving some algebraic constraints (set of feasible parts)

5.1.2 Constructor & Destructor Documentation

5.1.2.1 AbstractSet::~AbstractSet ( ) [virtual]

The objective that one wants to optimise (assumed to be decomposable: the objective of a partition is function of the objectives of its parts)

Destructor

5.1.3 Member Function Documentation

5.1.3.1 virtual void AbstractSet::computeOptimalPartition (double *parameter*) [pure virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implemented in MultiSet, Graph, BiSet, NonconstrainedSet, Ring, NHOSet, HierarchicalHierarchicalSet, OrderedSet, NonconstrainedOrderedSet, HierarchicalOrderedSet, and HierarchicalSet.

5.1.3.2 virtual Partition* AbstractSet::getOptimalPartition (double *parameter*) [pure virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implemented in MultiSet, Graph, BiSet, NonconstrainedSet, Ring, NHOSet, HierarchicalHierarchicalSet, OrderedSet, NonconstrainedOrderedSet, HierarchicalOrderedSet, and HierarchicalSet.

5.1.3.3 PartitionList * AbstractSet::getOptimalPartitionList (double *threshold*)

Compute and return a list of partitions that fit with the algebraic constraints and that optimises the objective function that has been specified, while the parameter of the objective function varies on a proper ranged (defined by the objective itself)

Parameters

threshold	: The minimal distance between two successive parameters giving birth to two different partitions

Returns

: The resulting list of optimal partitions

5.1.3.4 virtual void AbstractSet::printOptimalPartition (double *parameter*) [pure virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implemented in MultiSet, Graph, BiSet, NonconstrainedSet, Ring, NHOSet, HierarchicalHierarchicalSet, OrderedSet, NonconstrainedOrderedSet, HierarchicalOrderedSet, and HierarchicalSet.

5.1.3.5 void AbstractSet::printOptimalPartitionList (double *threshold*)

Compute and print a list of partitions that fit with the algebraic constraints and that optimises the objective function that has been specified, while the parameter of the objective function varies on a proper ranged (defined by the objective itself)

Parameters

threshold	: The minimal distance between two successive parameters giving birth to two different partitions

5.1.3.6 void AbstractSet::printOptimalPartitionListInCSV (double *threshold,* **Dataset** ∗ *data,* int *dim,* std::string *fileName*)

Compute and print in a CSV file a list of partitions that fit with the algebraic constraints and that optimises the objective function that has been specified, while the parameter of the objective function varies on a proper ranged (defined by the objective itself)

Parameters

threshold	: The minimal distance between two successive parameters giving birth to two different partitions

5.1.3.7 virtual void AbstractSet::setObjectiveFunction (**ObjectiveFunction** ∗ *objective*) [pure virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

Implemented in MultiSet, Graph, BiSet, Ring, NonconstrainedSet, NHOSet, HierarchicalHierarchicalSet, OrderedSet, NonconstrainedOrderedSet, HierarchicalOrderedSet, and HierarchicalSet.

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

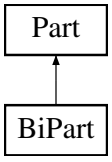
- /home/lamarche/programming/optimal_partition/src/abstract_set.hpp
- /home/lamarche/programming/optimal_partition/src/abstract_set.cpp

5.2 BiPart Class Reference

A bi-part is a subset of a bi-dimensional set of elements (individuals)

`#include <partition.hpp>`

Inheritance diagram for BiPart:



Public Member Functions

- **BiPart** (Part ∗part1, Part ∗part2, ObjectiveValue ∗value=0)
- **BiPart** (BiPart ∗biPart)
- bool **equal** (Part ∗p)
- void **print** (bool endl=false)
- int **printSize** ()

Public Attributes

- Part ∗ **firstPart**
- Part ∗ **secondPart**

5.2.1 Detailed Description

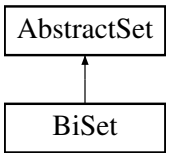
A bi-part is a subset of a bi-dimensional set of elements (individuals)

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

- /home/lamarche/programming/optimal_partition/src/partition.hpp
- /home/lamarche/programming/optimal_partition/src/partition.cpp

5.3 BiSet Class Reference

Inheritance diagram for BiSet:



Public Member Functions

- **BiSet** (UniSet ∗uniSet1, UniSet ∗uniSet2)
- void **initReached** ()
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction ∗m)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction ∗objective) should have been called first)
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)

- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition * getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- UniSet * **uniSet1**
- UniSet * **uniSet2**
- int **biSubsetNumber**
- int **atomicBiSubsetNumber**
- BiSubset * **firstBiSubset**
- BiSubset ** **biSubsetArray**

5.3.1 Member Function Documentation

5.3.1.1 void BiSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.3.1.2 Partition * BiSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.3.1.3 void BiSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.3.1.4 void BiSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/bi_set.hpp
- /home/lamarche/programming/optimal_partition/src/bi_set.cpp

5.4 BiSubset Class Reference

Public Member Functions

- **BiSubset** (UniSubset *uniSubset1, UniSubset *uniSubset2)
- void **print** ()
- void **printIndexSet** (bool endl=false)
- void **addBiSubsetSet** (BiSubsetSet *biSubsetSet)
- void **setObjectiveFunction** (ObjectiveFunction *m)
- void **buildDataStructure** ()
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue *maxQual=0)
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- void **buildOptimalPartition** (Partition *partition)

Public Attributes

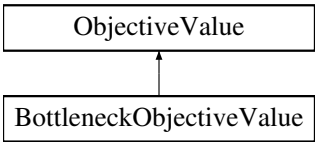
- UniSubset * **uniSubset1**
- UniSubset * **uniSubset2**
- int **num**
- bool **isAtomic**
- bool **reached**
- BiSubsetSetSet * **biSubsetSetSet**
- BiSet * **biSet**
- ObjectiveFunction * **objective**
- ObjectiveValue * **value**
- double **optimalValue**
- BiSubsetSet * **optimalCut**

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

- /home/lamarche/programming/optimal_partition/src/bi_set.hpp
- /home/lamarche/programming/optimal_partition/src/bi_set.cpp

5.5 BottleneckObjectiveValue Class Reference

Inheritance diagram for BottleneckObjectiveValue:



Public Member Functions

- **BottleneckObjectiveValue** (InformationBottleneck ∗objective, int index=-1)
- void **add** (ObjectiveValue ∗value)
- void **compute** ()
- void **compute** (ObjectiveValue ∗value1, ObjectiveValue ∗value2)
- void **compute** (ObjectiveValueSet ∗valueSet)
- void **normalize** (ObjectiveValue ∗q)
- void **print** (bool verbose=true)
- double **getValue** (double param)

Public Attributes

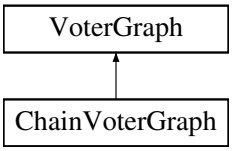
- int **index**
- double **pk**
- double ∗ **pkj**
- double ∗ **pj**
- double **lki**
- double **lkj**

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

- /home/lamarche/programming/optimal_partition/src/information_bottleneck.hpp
- /home/lamarche/programming/optimal_partition/src/information_bottleneck.cpp

5.6 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

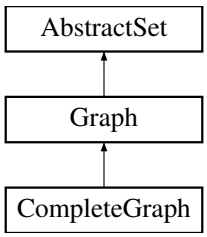
- int **size**
- double **contrarian**
- VoterNode ** **nodeArray**
- bool **ring**

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

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

5.7 CompleteGraph Class Reference

Inheritance diagram for CompleteGraph:



Public Member Functions

- **CompleteGraph** (int vNum)

Additional Inherited Members

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

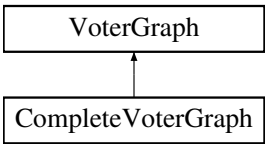
- /home/lamarche/programming/optimal_partition/src/graph.hpp

5.8 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

5.8.1 Detailed Description

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

5.8.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

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

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

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

5.9 DataPointStruct Struct Reference

Public Attributes

- std::vector< int > **parameters**
- float **time**
- int **memory**

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

- /home/lamarche/programming/optimal_partition/src/timer.hpp

5.10 Dataset Class Reference

Public Member Functions

- void **print** ()
- void **buildDataset** ()
- void **initValues** (double v=0)
- void **initRefValues** (double v=0)
- void **addLabel1** (std::string str)
- void **addLabel2** (std::string str)
- std::string **getLabel1** (int i)
- std::string **getLabel2** (int j)
- int **getIndex1** (std::string s1)
- int **getIndex2** (std::string s2)
- double **getValue** (int i, int j)
- double **getRefValue** (int i, int j)
- double **getValue** (std::string s1, std::string s2)
- double **getRefValue** (std::string s1, std::string s2)

- void **setValue** (int i, int j, double v)
- void **setRefValue** (int i, int j, double v)
- void **setValue** (std::string s1, std::string s2, double v)
- void **setRefValue** (std::string s1, std::string s2, double v)
- void **incrementValue** (int i, int j)
- void **incrementRefValue** (int i, int j)
- void **incrementValue** (std::string s1, std::string s2)
- void **incrementRefValue** (std::string s1, std::string s2)
- double * **getValues1** (std::string s2)
- double * **getValues2** (std::string s1)
- double * **getRefValues1** (std::string s2)
- double * **getRefValues2** (std::string s1)
- double * **getValues** (bool order=true)
- double * **getRefValues** (bool order=true)

Public Attributes

- int **size1**
- int **size2**
- std::map< std::string, int > * **indices1**
- std::map< std::string, int > * **indices2**
- std::map< int, std::string > * **labels1**
- std::map< int, std::string > * **labels2**
- double * **values**
- double * **refValues**

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

- /home/lamarche/programming/optimal_partition/src/dataset.hpp
- /home/lamarche/programming/optimal_partition/src/dataset.cpp

5.11 Datatree Class Reference

Public Member Functions

- **Datatree** (Datatree &tree)
- **Datatree** (int vertex=-1)
- void **setObjectiveFunction** (ObjectiveFunction *objective)
- std::string **toString** ()
- Vertices * **getAllVertices** ()
- Datatree * **addChild** (int v, bool print=true)
- Datatree * **findChild** (int v)
- Datatree * **findOrAddChild** (int v, bool print=true)
- void **addBipartition** (Datatree *n1, Datatree *n2)
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue *maxObjectiveValue=0)
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- Partition * **getOptimalPartition** (double parameter)
- void **print** (bool verbose=false)
- void **printVertices** (bool endl=true)
- PartSet * **getParts** ()
- void **printParts** ()
- PartitionList * **getAllPartitions** ()
- int **printPartitions** (bool print=true)

Public Attributes

- int **size**
- int **vertex**
- bool **wholeSet**
- ObjectiveFunction * **objective**
- Datatree * **parent**
- Datatree * **complement**
- TreesList * **complementList**
- TreesSet * **children**
- ObjectiveValue * **value**
- BipartitionsSet * **bipartitions**
- double **optimalValue**
- Bipartition * **optimalBipartition**
- bool **optimized**

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

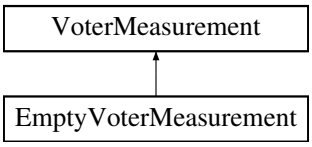
- /home/lamarche/programming/optimal_partition/src/datatree.hpp
- /home/lamarche/programming/optimal_partition/src/datatree.cpp

5.12 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

5.12.1 Detailed Description

A measurement without any probe (no observation)

5.12.2 Constructor & Destructor Documentation

5.12.2.1 EmptyVoterMeasurement::EmptyVoterMeasurement (VoterGraph * *graph*)

Constructor.

Parameters

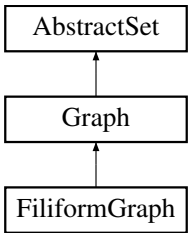
graph	: The interaction graph to be observed

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

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

5.13 FiliformGraph Class Reference

Inheritance diagram for FiliformGraph:



Public Member Functions

- **FiliformGraph** (int vNum)

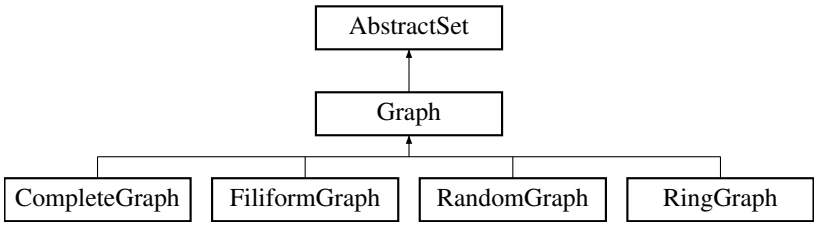
Additional Inherited Members

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

- /home/lamarche/programming/optimal_partition/src/graph.hpp
- /home/lamarche/programming/optimal_partition/src/graph.cpp

5.14 Graph Class Reference

Inheritance diagram for Graph:



Public Member Functions

- **Graph** (int size)
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction ∗m)

Set the objective that one wants to optimise.

- void print ()

  Print the set and its algebraic constraints.
- void **addEdge** (int v1, int v2)
- void **buildFromBinary** (int index)
- bool **areAdjacent** (int v1, int v2)
- bool **areAdjacent** (int v1, Vertices *v2)
- bool **areAdjacent** (Vertices *v1, int v2)
- bool **areAdjacent** (Vertices *v1, Vertices *v2)
- bool **areAdjacent** (int v1, VVertices *v2)
- bool **areAdjacent** (VVertices *v1, int v2)
- bool **areAdjacent** (VVertices *v1, VVertices *v2)
- Vertices * **getAdjacentVertices** (int v, int vMax=-1)
- void **printVertices** (Vertices *V)
- bool **isConnected** ()
- bool **isConnected** (Vertices *V)
- void **printDataStructure** (bool verbose=true)
- PartSet * **getParts** ()
- void **printParts** ()
- int **printPartitions** (bool print=true)
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  *Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction *objective) should have been called first)*
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void **buildDataStructureWithSlyce** ()
- void **slyce** (VVertices *R, VVertices *F, int m)
- void **enumerateSubsets** (VVertices *R, VVertices *F, int m, int n, VVertices *T, int q, int r)
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition * getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- int **size**
- Vertices ** **adjacencySets**
- GraphComponent ** **graphComponents**
- GraphComponentSet * **graphComponentSet**
- bool * **reachedVertices**
- ObjectiveValue * **value**

5.14.1 Member Function Documentation

5.14.1.1 void Graph::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.14.1.2 Partition * Graph::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.14.1.3 void Graph::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.14.1.4 void Graph::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

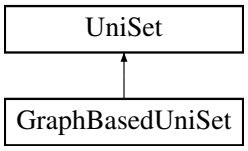
Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/graph.hpp
- /home/lamarche/programming/optimal_partition/src/graph.cpp

5.15 GraphBasedUniSet Class Reference

Inheritance diagram for GraphBasedUniSet:



Public Member Functions

- **GraphBasedUniSet** (Graph ∗graph)

Public Attributes

- Graph ∗ **graph**

Additional Inherited Members

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

- /home/lamarche/programming/optimal_partition/src/uni_set.hpp
- /home/lamarche/programming/optimal_partition/src/uni_set.cpp

5.16 GraphComponent Class Reference

Public Member Functions

- **GraphComponent** (Graph ∗graph)
- void **setVertices** (std::list< int > ∗vertexList)
- void **printDataStructure** (bool verbose=true)
- void **buildDataStructure** ()
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** ()
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- Partition ∗ **getOptimalPartition** (double parameter)

Public Attributes

- int **size**
- int ∗ **vertices**
- std::map< int, int > ∗ **order**
- Graph ∗ **graph**
- Datatree ∗ **datatree**

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

- /home/lamarche/programming/optimal_partition/src/graph.hpp
- /home/lamarche/programming/optimal_partition/src/graph.cpp

5.17 HHNode Class Reference

Public Member Functions

- **HHNode** (HNode ∗node1, HNode ∗node2)
- void **addChild1** (HHNode ∗node)
- void **addChild2** (HHNode ∗node)
- void **setObjectiveFunction** (ObjectiveFunction ∗m)
- void **print** ()
- void **printIndices** (bool endl=false)
- void **buildDataStructure** ()
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue ∗maxQual=0)
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- void **buildOptimalPartition** (Partition ∗partition)

Public Attributes

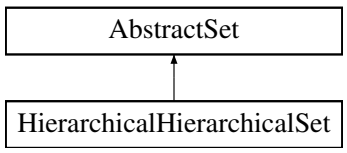
- HNode ∗ **node1**
- HNode ∗ **node2**
- HHNodeSet ∗ **children1**
- HHNodeSet ∗ **children2**
- ObjectiveFunction ∗ **objective**
- ObjectiveValue ∗ **value**
- double **optimalValue**
- int **optimalCut**

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

- /home/lamarche/programming/optimal_partition/src/hierarchical_hierarchical_set.hpp
- /home/lamarche/programming/optimal_partition/src/hierarchical_hierarchical_set.cpp

5.18 HierarchicalHierarchicalSet Class Reference

Inheritance diagram for HierarchicalHierarchicalSet:



Public Member Functions

- **HierarchicalHierarchicalSet** (HNode ∗hierarchy1, HNode ∗hierarchy2)
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction ∗m)

  Set the objective that one wants to optimise.

- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction ∗objective) should have been called first)
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition ∗ getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- HNode ∗ **hierarchy1**
- HNode ∗ **hierarchy2**
- HHNode ∗ **hyperarchy**

5.18.1 Member Function Documentation

5.18.1.1 void HierarchicalHierarchicalSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.18.1.2 Partition ∗ HierarchicalHierarchicalSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.18.1.3 void HierarchicalHierarchicalSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.18.1.4 void HierarchicalHierarchicalSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

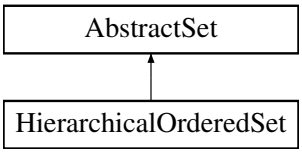
Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/hierarchical_hierarchical_set.hpp
- /home/lamarche/programming/optimal_partition/src/hierarchical_hierarchical_set.cpp

5.19 HierarchicalOrderedSet Class Reference

Inheritance diagram for HierarchicalOrderedSet:



Public Member Functions

- **HierarchicalOrderedSet** (HONode *hierarchy, int size)
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction *m)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.

- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  *Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction *objective) should have been called first)*
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition * getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- int **size**
- HONode * **hierarchy**

5.19.1 Member Function Documentation

5.19.1.1 void HierarchicalOrderedSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.19.1.2 Partition * HierarchicalOrderedSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.19.1.3 void HierarchicalOrderedSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.19.1.4 void HierarchicalOrderedSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

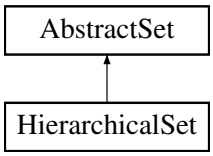
Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/hierarchical_ordered_set.hpp
- /home/lamarche/programming/optimal_partition/src/hierarchical_ordered_set.cpp

5.20 HierarchicalSet Class Reference

Inheritance diagram for HierarchicalSet:



Public Member Functions

- **HierarchicalSet** (HNode *hierarchy)
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction *m)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  *Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction *objective) should have been called first)*
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)

- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition * getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- HNode * **hierarchy**

5.20.1 Member Function Documentation

5.20.1.1 void HierarchicalSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.20.1.2 Partition * HierarchicalSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.20.1.3 void HierarchicalSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.20.1.4 void HierarchicalSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

Implements AbstractSet.

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

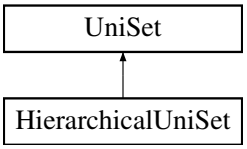
- /home/lamarche/programming/optimal_partition/src/hierarchical_set.hpp
- /home/lamarche/programming/optimal_partition/src/hierarchical_set.cpp

5.21 HierarchicalUniSet Class Reference

A uni-dimensional set of elements structured according to a complete binary hierarchy, and such that the feasible subsets are all the nodes of the hierarchy.

`#include <uni_set.hpp>`

Inheritance diagram for HierarchicalUniSet:



Public Member Functions

- HierarchicalUniSet (int depth)

  Depth of the complete binary hierarchy.

Public Attributes

- int **depth**

Additional Inherited Members

5.21.1 Detailed Description

A uni-dimensional set of elements structured according to a complete binary hierarchy, and such that the feasible subsets are all the nodes of the hierarchy.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 HierarchicalUniSet::HierarchicalUniSet (int *depth*)

Depth of the complete binary hierarchy.

Constructor

Parameters

size	: Depth of the complete binary hierarchy

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

- /home/lamarche/programming/optimal_partition/src/uni_set.hpp
- /home/lamarche/programming/optimal_partition/src/uni_set.cpp

5.22 HNode Class Reference

Public Member Functions

- **HNode** (int index=-1)
- void **addChild** (HNode ∗node)
- void **setObjectiveFunction** (ObjectiveFunction ∗m)
- void **print** ()
- void **printIndices** (bool endl=false)
- void **buildDataStructure** (HNode ∗root=0, int level=0, int num=0)
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue ∗maxQual=0)
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- Partition ∗ **getOptimalPartition** (double parameter)
- void **buildOptimalPartition** (Partition ∗partition)

Public Attributes

- int **index**
- std::set< int > ∗ **indices**
- int **level**
- int **size**
- int **width**
- int **num**
- HNode ∗ **root**
- HNodeSet ∗ **children**
- ObjectiveFunction ∗ **objective**
- ObjectiveValue ∗ **value**
- double **optimalValue**
- bool **optimalCut**

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

- /home/lamarche/programming/optimal_partition/src/hierarchical_set.hpp
- /home/lamarche/programming/optimal_partition/src/hierarchical_set.cpp

5.23 HONode Class Reference

Public Member Functions

- **HONode** (int size, int index=-1)
- int **getIndex** (int i, int j)
- void **addChild** (HONode ∗node)
- void **setObjectiveFunction** (ObjectiveFunction ∗m)
- void **print** ()
- void **buildDataStructure** (int level=0)
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue ∗maxQual=0)
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- Partition ∗ **getOptimalPartition** (double parameter)
- void **buildOptimalPartition** (Partition ∗partition, int pi=0, int pj=-1)

Public Attributes

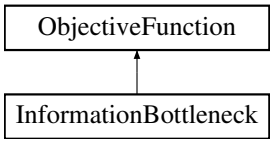
- int **index**
- int **level**
- std::set< int > ∗ **indices**
- HONodeSet ∗ **children**
- ObjectiveFunction ∗ **objective**
- int **size**
- ObjectiveValue ∗∗ **qualities**
- double ∗ **optimalValues**
- int ∗ **optimalCuts**

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

- /home/lamarche/programming/optimal_partition/src/hierarchical_ordered_set.hpp
- /home/lamarche/programming/optimal_partition/src/hierarchical_ordered_set.cpp

5.24 InformationBottleneck Class Reference

Inheritance diagram for InformationBottleneck:



Public Member Functions

- **InformationBottleneck** (MarkovProcess ∗process)
- void setRandom ()

  Randomly set the initial data from which the objective function is computed.
- ObjectiveValue ∗ newObjectiveValue (int index=-1)

This method is called by child classes of AbstractSet (do not use directly)

- double **getParameter** (double unit)
- double **getUnitDistance** (double uMin, double uMax)
- double **getIntermediaryUnit** (double uMin, double uMax)

Public Attributes

- MarkovProcess * **process**

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

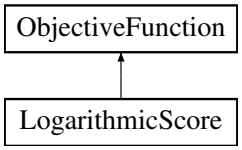
- /home/lamarche/programming/optimal_partition/src/information_bottleneck.hpp
- /home/lamarche/programming/optimal_partition/src/information_bottleneck.cpp

5.25 LogarithmicScore Class Reference

Class to define and compute the logarithmic score function in the case of point prediction.

`#include <logarithmic_score.hpp>`

Inheritance diagram for LogarithmicScore:



Public Member Functions

- LogarithmicScore (PredictionDataset *dataset, int prior=0)

  Constructor.
- ~LogarithmicScore ()

  Destructor.
- void setRandom ()

  Randomly set the initial data from which the objective function is computed.
- ObjectiveValue * newObjectiveValue (int index=-1)

  This method is called by child classes of AbstractSet (do not use directly)
- void computeObjectiveValues ()

  This method is called by child classes of AbstractSet (do not use directly)
- void printObjectiveValues (bool verbose=true)

  This method is called by child classes of AbstractSet (do not use directly)
- double **getParameter** (double unit)
- double **getUnitDistance** (double uMin, double uMax)
- double **getIntermediaryUnit** (double uMin, double uMax)

Additional Inherited Members

5.25.1 Detailed Description

Class to define and compute the logarithmic score function in the case of point prediction.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 LogarithmicScore::LogarithmicScore (PredictionDataset * *dataset*, int *prior* = 0)

Constructor.

Parameters

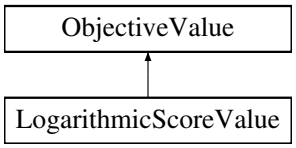
dataset	: The prediction data set that is use to evaluate the score function (from a train set and a test set both containing pre-observations and post-observations)
prior	: A prior giving the number of times each couple of (pre and post) observations has been observed in addition to the train set

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

- /home/lamarche/programming/optimal_partition/src/logarithmic_score.hpp
- /home/lamarche/programming/optimal_partition/src/logarithmic_score.cpp

5.26 LogarithmicScoreValue Class Reference

Inheritance diagram for LogarithmicScoreValue:



Public Member Functions

- **LogarithmicScoreValue** (LogarithmicScore *objective)
- void **add** (ObjectiveValue *value)
- void **compute** ()
- void **compute** (ObjectiveValue *value1, ObjectiveValue *value2)
- void **compute** (ObjectiveValueSet *valueset)
- void **normalize** (ObjectiveValue *q)
- void **print** (bool verbosex=true)
- double **getValue** (double param)

Public Attributes

- int **preSize**
- int **postSize**
- int * **trainCountArray**
- int * **testCountArray**
- int **trainCountTotal**
- int **testCountTotal**
- double **score**
- bool **infinite**

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

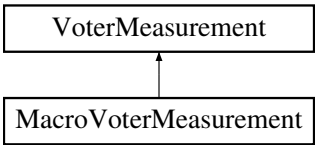
- /home/lamarche/programming/optimal_partition/src/logarithmic_score.hpp
- /home/lamarche/programming/optimal_partition/src/logarithmic_score.cpp

5.27 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

5.27.1 Detailed Description

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

5.27.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

graph	: The interaction graph to be observed
metrics	: The set of metrics associated to the macro-probe

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

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

5.28 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/optimal_partition/src/markov_process.hpp
- /home/lamarche/programming/optimal_partition/src/markov_process.cpp

5.29 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 *current-Partition, 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

5.29.1 Detailed Description

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

5.29.2 Constructor & Destructor Documentation

5.29.2.1 MarkovProcess::MarkovProcess (int *s*)

Constructor.

Parameters

size	: The size of the Markov chain state space

Author

Robin Lamarche-Perrin

Date

22/01/2015

5.29.3 Member Function Documentation

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

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

Parameters

threshold	: 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

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

Compute a possible trajectory of the Markov chain.

Parameters

length	: Length of the trajectory

5.29.3.3 void MarkovProcess::setDistribution (double * *array*)

Set the initial distribution.

Parameters

array	: An array of probabilities (summing to 1) with the size of the Markov chain state space

5.29.3.4 void MarkovProcess::setTransition (double * *array*)

Set the transition kernel.

Parameters

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

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

Set one line of the transition kernel.

Parameters

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

5.29.4 Member Data Documentation

5.29.4.1 double* MarkovProcess::distribution

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

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

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

5.29.4.3 int MarkovProcess::lastDelay

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

5.29.4.4 int MarkovProcess::lastTime

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

5.29.4.5 int MarkovProcess::size

The size of the Markov chain state space

5.29.4.6 double* MarkovProcess::transition

The transition kernel of the Markov chain (1 step)

5.29.4.7 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/optimal_partition/src/markov_process.hpp
- /home/lamarche/programming/optimal_partition/src/markov_process.cpp

5.30 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:

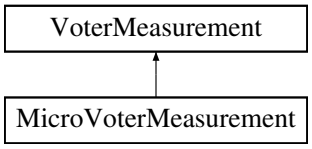
- /home/lamarche/programming/optimal_partition/src/markov_process.hpp
- /home/lamarche/programming/optimal_partition/src/markov_process.cpp

5.31 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

5.31.1 Detailed Description

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

5.31.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

graph	: The interaction graph to be observed
metric	: The set of metric associated to the micro-probe

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

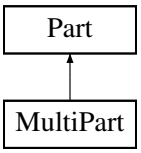
- /home/lamarche/programming/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.32 MultiPart Class Reference

A multi-part is a subset of a multi-dimensional set of elements (individuals)

`#include <partition.hpp>`

Inheritance diagram for MultiPart:



Public Member Functions

- **MultiPart** (Part **partArray, int dimension, ObjectiveValue *value=0)
- **MultiPart** (MultiPart *multiPart)
- bool **equal** (Part *p)
- void **print** (bool endl=false)
- int **printSize** ()

Public Attributes

- int **dimension**
- Part ** **partArray**

5.32.1 Detailed Description

A multi-part is a subset of a multi-dimensional set of elements (individuals)

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

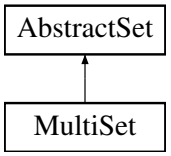
- /home/lamarche/programming/optimal_partition/src/partition.hpp
- /home/lamarche/programming/optimal_partition/src/partition.cpp

5.33 MultiSet Class Reference

A multi-dimensional set of elements based on the Cartesian product of several uni-dimensional sets (UniSet) and their algebraic structures (feasible subsets and feasible refinements)

`#include <multi_set.hpp>`

Inheritance diagram for MultiSet:



Public Member Functions

- MultiSet (UniSet *uniSet)

  Number of feasible subsets.
- MultiSet (UniSet **uniSetArray, int dimension)

  Constructor.
- MultiSet (std::vector< UniSet * > *uniSetVector)

  Constructor.
- virtual ~MultiSet ()
- MultiSubset * getAtomicMultiSubset (int index)
- MultiSubset * getAtomicMultiSubset (int *indices)
- MultiSubset * getRandomAtomicMultiSubset ()
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction *objective)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction ∗objective) should have been called first)

- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)

- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.

- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

- Partition ∗ getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- int **dimension**
- int atomicMultiSubsetNumber

  Number of dimensions.

- int multiSubsetNumber

  Number of elements (e.g., atomic feasible subsets)

Protected Member Functions

- int **getNum** (int ∗multiNum)
- int ∗ **getMultiNum** (int num)
- void **initReached** ()

Protected Attributes

- UniSet ∗∗ **uniSetArray**
- MultiSubset ∗ **firstMultiSubset**
- MultiSubset ∗∗ **multiSubsetArray**
- MultiSubset ∗∗ **atomicMultiSubsetArray**

5.33.1 Detailed Description

A multi-dimensional set of elements based on the Cartesian product of several uni-dimensional sets (UniSet) and their algebraic structures (feasible subsets and feasible refinements)

5.33.2 Constructor & Destructor Documentation

5.33.2.1 MultiSet::MultiSet (UniSet ∗ *uniSet*)

Number of feasible subsets.

Constructor for a one-dimensional set

Parameters

uniSet	: Pointer to one uni-dimensional set (UniSet)

5.33.2.2 MultiSet::MultiSet (UniSet ** *uniSetArray,* int *dimension*)

Constructor.

Parameters

uniSetArray	: Array of pointers to uni-dimensional sets from which the Cartesian product is computed
dimension	: Number of uni-dimensional sets

5.33.2.3 MultiSet::MultiSet (std::vector< UniSet * > * *uniSetVector*)

Constructor.

Parameters

uniSetVector	: Vector of pointers to uni-dimensional sets from which the Cartesian product is computed

5.33.2.4 MultiSet::~MultiSet ( ) [virtual]

Destructor

5.33.3 Member Function Documentation

5.33.3.1 void MultiSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.33.3.2 MultiSubset * MultiSet::getAtomicMultiSubset (int *index*)

Access to an element (e.g., atomic feasible subset) from its index IN THE CASE OF A ONE-DIMENSIONAL SET /param index : The index of the element to access /return A pointer to the unique atomic feasible subset that contains the element

5.33.3.3 MultiSubset * MultiSet::getAtomicMultiSubset (int * *indices*)

Access to an element (e.g., atomic feasible subset) from its indices /param indices : The indices of the element to access /return A pointer to the unique atomic feasible subset that contains the element

5.33.3.4 Partition * MultiSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.33.3.5 MultiSubset * MultiSet::getRandomAtomicMultiSubset ( )

Access to a random element (e.g., atomic feasible subset) /return A pointer to the unique atomic feasible subset that contains the element

5.33.3.6 void MultiSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.33.3.7 void MultiSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/multi_set.hpp
- /home/lamarche/programming/optimal_partition/src/multi_set.cpp

5.34 MultiSubset Class Reference

Public Member Functions

- **MultiSubset** (UniSubset **uniSubsetArray, int dimension)
- void **print** ()
- void **printIndexSet** (bool endl=false)
- void **addMultiSubsetSet** (MultiSubsetSet *multiSubsetSet)
- void **setObjectiveFunction** (ObjectiveFunction *m)
- void **buildDataStructure** ()
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue *maxQual=0)
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- void **buildOptimalPartition** (Partition *partition)

Public Attributes

- int **dimension**
- UniSubset ∗∗ **uniSubsetArray**
- int **num**
- int **atomicNum**
- bool **isAtomic**
- bool **reached**
- MultiSubsetSetSet ∗ **multiSubsetSetSet**
- MultiSet ∗ **multiSet**
- ObjectiveFunction ∗ **objective**
- ObjectiveValue ∗ **value**
- double **optimalValue**
- MultiSubsetSet ∗ **optimalCut**

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

- /home/lamarche/programming/optimal_partition/src/multi_set.hpp
- /home/lamarche/programming/optimal_partition/src/multi_set.cpp

5.35 NHONode Class Reference

Public Member Functions

- **HONode** (int size, int index=-1)
- int **getIndex** (int i, int j)
- void **addChild** (HONode ∗node)
- void **setObjectiveFunction** (ObjectiveFunction ∗m)
- void **print** ()
- void **buildDataStructure** (int level=0)
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue ∗maxQual=0)
- void **printObjectiveValues** ()
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- Partition ∗ **getOptimalPartition** (double parameter)
- void **buildOptimalPartition** (Partition ∗partition, int pi=0, int pj=-1)

Public Attributes

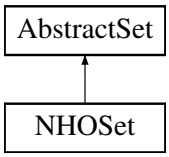
- int **index**
- int **level**
- std::set< int > ∗ **indices**
- HONodeSet ∗ **children**
- ObjectiveFunction ∗ **objective**
- int **size**
- ObjectiveValue ∗∗ **qualities**
- double ∗ **optimalValues**
- int ∗ **optimalCuts**

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

- /home/lamarche/programming/optimal_partition/src/NHO_set.hpp

5.36 NHOSet Class Reference

Inheritance diagram for NHOSet:



Public Member Functions

- **NHOSet** (int NSize, HNode ∗HHierarchy, int OSize)
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction ∗m)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction ∗objective) should have been called first)
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition ∗ getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- int **NSize**
- HNode ∗ **HHierarchy**
- int **OSize**
- NHODatatree ∗ **NHODatatree**

5.36.1 Member Function Documentation

5.36.1.1 void NHOSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.36.1.2 Partition* NHOSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.36.1.3 void NHOSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.36.1.4 void NHOSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

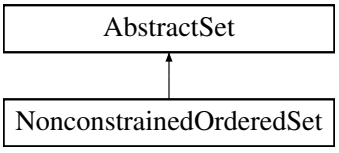
Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/NHO_set.hpp

5.37 NonconstrainedOrderedSet Class Reference

Inheritance diagram for NonconstrainedOrderedSet:



Public Member Functions

- **NonconstrainedOrderedSet** (int size1, int size2)
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction *m)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  *Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction *objective) should have been called first)*
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition * getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- int **size1**
- int **size2**
- OrderedDatatree * **dataTree**
- ObjectiveValue ** **qualities**

5.37.1 Member Function Documentation

5.37.1.1 void NonconstrainedOrderedSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.37.1.2 Partition * NonconstrainedOrderedSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.37.1.3 void NonconstrainedOrderedSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.37.1.4 void NonconstrainedOrderedSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

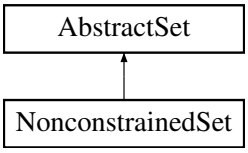
Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/nonconstrained_ordered_set.hpp
- /home/lamarche/programming/optimal_partition/src/nonconstrained_ordered_set.cpp

5.38 NonconstrainedSet Class Reference

Inheritance diagram for NonconstrainedSet:



Public Member Functions

- **NonconstrainedSet** (int size)
- void setRandom ()

Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)

- void setObjectiveFunction (ObjectiveFunction *m)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.
- void **printDataTree** (bool verbose=true)
- void **printParts** ()
- int **printPartitions** (bool print=true)
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  *Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction *objective) should have been called first)*
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition * getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- int **size**
- Datatree * **dataTree**
- ObjectiveValue ** **qualities**

5.38.1 Member Function Documentation

5.38.1.1 void NonconstrainedSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.38.1.2 Partition * NonconstrainedSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.38.1.3 void NonconstrainedSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.38.1.4 void NonconstrainedSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

Implements AbstractSet.

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

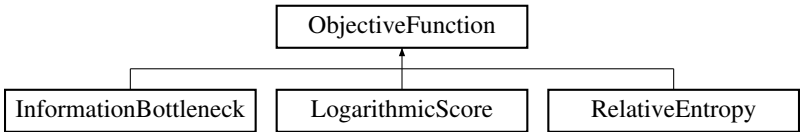
- /home/lamarche/programming/optimal_partition/src/nonconstrained_set.hpp
- /home/lamarche/programming/optimal_partition/src/nonconstrained_set.cpp

5.39 ObjectiveFunction Class Reference

Abstract class defining an objective function to be associated to a constrained set in order to define the optimisation problem that one wants to solve.

`#include <objective_function.hpp>`

Inheritance diagram for ObjectiveFunction:



Public Member Functions

- ObjectiveFunction ()

  True if one deals with a maximisation problem, and false if one deals with a minimisation problem.
- virtual ~ObjectiveFunction ()

Destructor.

- virtual void setRandom ()=0

  Randomly set the initial data from which the objective function is computed.

- virtual void computeObjectiveValues ()=0

  This method is called by child classes of AbstractSet (do not use directly)

- virtual void printObjectiveValues (bool verbose=true)=0

  This method is called by child classes of AbstractSet (do not use directly)

- virtual ObjectiveValue * newObjectiveValue (int index=-1)=0

  This method is called by child classes of AbstractSet (do not use directly)

Public Attributes

- bool **maximize**

Friends

- class **AbstractSet**

5.39.1 Detailed Description

Abstract class defining an objective function to be associated to a constrained set in order to define the optimisation problem that one wants to solve.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 ObjectiveFunction::ObjectiveFunction ( )

True if one deals with a maximisation problem, and false if one deals with a minimisation problem.

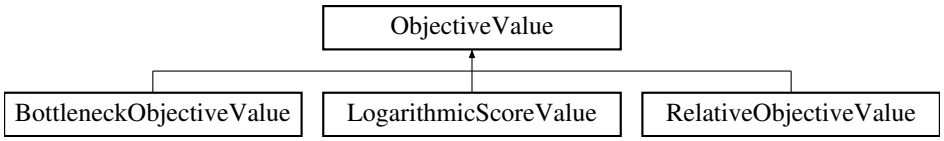
Constructor

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

- /home/lamarche/programming/optimal_partition/src/objective_function.hpp
- /home/lamarche/programming/optimal_partition/src/objective_function.cpp

5.40 ObjectiveValue Class Reference

Inheritance diagram for ObjectiveValue:



Public Member Functions

- virtual void **add** (ObjectiveValue *value)=0
- virtual void **compute** ()=0
- virtual void **compute** (ObjectiveValue *value1, ObjectiveValue *value2)=0

- virtual void **compute** (ObjectiveValueSet ∗valueSet)=0
- virtual void **normalize** (ObjectiveValue ∗normalizingValue)=0
- virtual double **getValue** (double param)=0
- virtual void **print** (bool verbose=true)=0

Public Attributes

- ObjectiveFunction ∗ **objective**

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

- /home/lamarche/programming/optimal_partition/src/objective_function.hpp
- /home/lamarche/programming/optimal_partition/src/objective_function.cpp

5.41 OrderedDatatree Class Reference

Public Member Functions

- **OrderedDatatree** (OrderedDatatree &tree)
- **OrderedDatatree** (int size2, int vertex=-1)
- void **setObjectiveFunction** (ObjectiveFunction ∗objective)
- Vertices ∗ **getAllVertices** ()
- int **getIndex** (int i, int j)
- OrderedDatatree ∗ **addChild** (int v, bool print=true)
- OrderedDatatree ∗ **findChild** (int v)
- OrderedDatatree ∗ **findOrAddChild** (int v, bool print=true)
- void **addBipartition** (OrderedDatatree ∗n1, OrderedDatatree ∗n2)
- void **computeObjectiveValues** ()
- void **normalizeObjectiveValues** (ObjectiveValue ∗maxObjectiveValue=0)
- void **printObjectiveValues** ()
- void **buildOptimalPartition** (Partition ∗partition, int pi=0, int pj=-1)
- void **computeOptimalPartition** (double parameter)
- void **printOptimalPartition** (double parameter)
- Partition ∗ **getOptimalPartition** (double parameter)
- void **print** (bool verbose=false)
- void **printVertices** (bool endl=true)

Public Attributes

- int **size1**
- int **size2**
- int **vertex**
- bool **wholeSet**
- ObjectiveFunction ∗ **objective**
- OrderedDatatree ∗ **parent**
- OrderedDatatree ∗ **complement**
- OrderedTreesList ∗ **complementList**
- OrderedTreesSet ∗ **children**
- OrderedBipartitionsSet ∗ **bipartitions**
- ObjectiveValue ∗∗ **qualities**
- double ∗ **optimalValues**
- int ∗ **optimalCuts**

- OrderedBipartition ∗∗ **optimalBipartitions**
- bool **optimized**

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

- /home/lamarche/programming/optimal_partition/src/datatree.hpp
- /home/lamarche/programming/optimal_partition/src/datatree.cpp

5.42 OrderedPartition Class Reference

Public Member Functions

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

Public Attributes

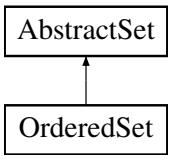
- int **microSize**
- int ∗ **optimalCut**
- double **param**
- double **beta**
- std::string **string**
- double **entropy**
- double **information**

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

- /home/lamarche/programming/optimal_partition/src/partition.hpp

5.43 OrderedSet Class Reference

Inheritance diagram for OrderedSet:



Public Member Functions

- **OrderedSet** (int s)
- int **getIndex** (int i, int j)
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void setObjectiveFunction (ObjectiveFunction ∗m)

  Set the objective that one wants to optimise.
- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)

- void computeObjectiveValues ()

  Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction ∗objective) should have been called first)
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- Partition ∗ getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Public Attributes

- int **size**
- ObjectiveValue ∗∗ **qualities**
- double ∗ **optimalValues**
- int ∗ **optimalCuts**

5.43.1 Member Function Documentation

5.43.1.1 void OrderedSet::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.43.1.2 Partition ∗ OrderedSet::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.43.1.3 void OrderedSet::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.43.1.4 void OrderedSet::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

Implements AbstractSet.

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

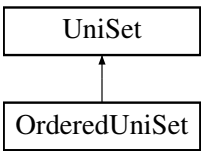
- /home/lamarche/programming/optimal_partition/src/orderedset.hpp
- /home/lamarche/programming/optimal_partition/src/orderedset.cpp

5.44 OrderedUniSet Class Reference

A uni-dimensional set of elements with a total order, and such that the feasible subsets are all the intervals induced by this order.

`#include <uni_set.hpp>`

Inheritance diagram for OrderedUniSet:



Public Member Functions

- OrderedUniSet (int size)

  Number of ordered elements.

Public Attributes

- int **size**

Additional Inherited Members

5.44.1 Detailed Description

A uni-dimensional set of elements with a total order, and such that the feasible subsets are all the intervals induced by this order.

5.44.2 Constructor & Destructor Documentation

5.44.2.1 OrderedUniSet::OrderedUniSet (int *size*)

Number of ordered elements.

Constructor

Parameters

size	: Number of ordered elements

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

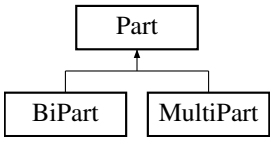
- /home/lamarche/programming/optimal_partition/src/uni_set.hpp
- /home/lamarche/programming/optimal_partition/src/uni_set.cpp

5.45 Part Class Reference

A part is a subset of a set of elements (individuals) represented by integers.

`#include <partition.hpp>`

Inheritance diagram for Part:



Public Member Functions

- **Part** (ObjectiveValue ∗value=0)
- **Part** (Part ∗part)
- **Part** (Datatree ∗node, ObjectiveValue ∗value=0)
- void **addIndividual** (int i, bool front=false, int value=-1)
- Vertices ∗ **getVertices** ()
- bool **contains** (int i)
- virtual bool **equal** (Part ∗p)
- virtual void **print** (bool endl=false)
- virtual int **printSize** ()

Public Attributes

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

5.45.1 Detailed Description

A part is a subset of a set of elements (individuals) represented by integers.

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

- /home/lamarche/programming/optimal_partition/src/partition.hpp
- /home/lamarche/programming/optimal_partition/src/partition.cpp

5.46 Partition Class Reference

A partition is a collection of pairwise-disjoint and covering subsets (parts) of a set of elements.

`#include <partition.hpp>`

Public Member Functions

- **Partition** (ObjectiveFunction *objective=0, double parameter=0)
- **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**
- double **parameter**
- std::list< Part * > * **parts**
- ObjectiveValue * **value**

5.46.1 Detailed Description

A partition is a collection of pairwise-disjoint and covering subsets (parts) of a set of elements.

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

- /home/lamarche/programming/optimal_partition/src/partition.hpp
- /home/lamarche/programming/optimal_partition/src/partition.cpp

5.47 PredictionDataset Class Reference

Class to represent a data set for the prediction of a post-measurement from the knowledge of a pre-measurement (composed of a train set and a test set)

`#include <prediction_dataset.hpp>`

Public Member Functions

- PredictionDataset (MultiSet ∗preMeasurement, MultiSet ∗postMeasurement)

  The number of observations associated to a couple (pre-value, post-value)
- ∼PredictionDataset ()

  Destructor.
- void addTrainValue (MultiSubset ∗preValue, MultiSubset ∗postValue, int count=1)

  Add a couple of (pre and post) observations to the train set.
- void addTestValue (MultiSubset ∗preValue, MultiSubset ∗postValue, int count=1)

  Add a couple of (pre and post) observations to the test set.
- void addTrainValue (int preIndex, int postIndex, int count=1)

  Add a couple of (pre and post) observations to the train set ***in the case of one-dimensional measurements***
- void addTestValue (int preIndex, int postIndex, int count=1)

  Add a couple of (pre and post) observations to the test set ***in the case of one-dimensional measurements***
- void print ()

Public Attributes

- MultiSet ∗ **preMultiSet**
- MultiSet ∗ postMultiSet

  The pre-measurement modelled as a structured multi-dimensional set of elements (the possible observation values)
- std::vector< MultiSubset ∗ > ∗ trainPreValues

  The post-measurement modelled as a structured multi-dimensional set of elements (the possible observation values)
- std::vector< MultiSubset ∗ > ∗ trainPostValues

  A vector of pre-observations that are used to train the predictor.
- std::vector< int > ∗ trainCountValues

  A vector of post-observations that are used to train the predictor.
- std::vector< MultiSubset ∗ > ∗ testPreValues

  The number of observations associated to a couple (pre-value, post-value)
- std::vector< MultiSubset ∗ > ∗ testPostValues

  A vector of pre-observations that are used to test the predictor.
- std::vector< int > ∗ testCountValues

  A vector of post-observations that are used to test the predictor.

5.47.1 Detailed Description

Class to represent a data set for the prediction of a post-measurement from the knowledge of a pre-measurement (composed of a train set and a test set)

5.47.2 Constructor & Destructor Documentation

5.47.2.1 PredictionDataset::PredictionDataset (MultiSet ∗ *preMeasurement,* MultiSet ∗ *postMeasurement*)

The number of observations associated to a couple (pre-value, post-value)

Constructor

Parameters

preMeasurement	: The pre-measurement modelled as a structured multi-dimensional set of elements (the possible observation values)
post-Measurement	: The post-measurement modelled as a structured multi-dimensional set of elements (the possible observation values)

5.47.3 Member Function Documentation

5.47.3.1 void PredictionDataset::addTestValue (MultiSubset * *preValue,* MultiSubset * *postValue,* int *count* = 1)

Add a couple of (pre and post) observations to the test set.

Parameters

preValue	: A pointer to the feasible subset that has been pre-observed in the multi-dimensional set representing the pre-measurement. It should always be an element of the set, that is an atomic feasible subset
preValue	: A pointer to the feasible subset that has been post-observed in the multi-dimensional set representing the pre-measurement. It should always be an element of the set, that is an atomic feasible subset
count	: The number of times the couple has been observed

5.47.3.2 void PredictionDataset::addTestValue (int *preIndex,* int *postIndex,* int *count* = 1)

Add a couple of (pre and post) observations to the test set **in the case of one-dimensional measurements**

Parameters

preIndex	: The index of the element that has been pre-observed in the one-dimensional set representing the pre-measurement
postIndex	: The index of the element that has been post-observed in the one-dimensional set representing the pre-measurement
count	: The number of times the couple has been observed

5.47.3.3 void PredictionDataset::addTrainValue (MultiSubset * *preValue,* MultiSubset * *postValue,* int *count* = 1)

Add a couple of (pre and post) observations to the train set.

Parameters

preValue	: A pointer to the feasible subset that have been pre-observed in the multi-dimensional set representing the pre-measurement. It should always be an element of the set, that is an atomic feasible subset
preValue	: A pointer to the feasible subset that have been post-observed in the multi-dimensional set representing the pre-measurement. It should always be an element of the set, that is an atomic feasible subset
count	: The number of times the couple has been observed

5.47.3.4 void PredictionDataset::addTrainValue (int *preIndex,* int *postIndex,* int *count* = 1)

Add a couple of (pre and post) observations to the train set **in the case of one-dimensional measurements**

Parameters

preIndex	: The index of the element that has been pre-observed in the one-dimensional set representing the pre-measurement
postIndex	: The index of the element that has been post-observed in the one-dimensional set representing the pre-measurement
count	: The number of times the couple has been observed

5.47.3.5 void PredictionDataset::print ( )

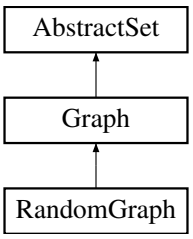
/brief Print the data set.

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

- /home/lamarche/programming/optimal_partition/src/prediction_dataset.hpp
- /home/lamarche/programming/optimal_partition/src/prediction_dataset.cpp

5.48 RandomGraph Class Reference

Inheritance diagram for RandomGraph:



Public Member Functions

- **RandomGraph** (int vNum, int eNum)

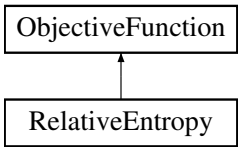
Additional Inherited Members

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

- /home/lamarche/programming/optimal_partition/src/graph.hpp
- /home/lamarche/programming/optimal_partition/src/graph.cpp

5.49 RelativeEntropy Class Reference

Inheritance diagram for RelativeEntropy:



Public Member Functions

- **RelativeEntropy** (int size, double *values=0, double *refValues=0)
- void setRandom ()

  Randomly set the initial data from which the objective function is computed.
- ObjectiveValue * newObjectiveValue (int index=-1)

  This method is called by child classes of AbstractSet (do not use directly)
- void computeObjectiveValues ()

  This method is called by child classes of AbstractSet (do not use directly)
- void printObjectiveValues (bool verbose=true)

  This method is called by child classes of AbstractSet (do not use directly)
- double **getParameter** (double unit)
- double **getUnitDistance** (double uMin, double uMax)
- double **getIntermediaryUnit** (double uMin, double uMax)

Public Attributes

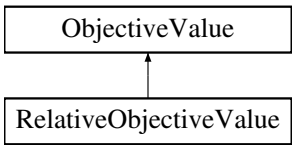
- int **size**
- double * **values**
- double * **refValues**

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

- /home/lamarche/programming/optimal_partition/src/relative_entropy.hpp
- /home/lamarche/programming/optimal_partition/src/relative_entropy.cpp

5.50 RelativeObjectiveValue Class Reference

Inheritance diagram for RelativeObjectiveValue:



Public Member Functions

- **RelativeObjectiveValue** (RelativeEntropy *objective, int index=-1)
- void **add** (ObjectiveValue *value)
- void **compute** ()
- void **compute** (ObjectiveValue *value1, ObjectiveValue *value2)
- void **compute** (ObjectiveValueSet *valueset)
- void **normalize** (ObjectiveValue *q)
- void **print** (bool verbose=true)
- double **getValue** (double param)

Public Attributes

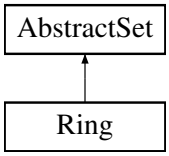
- int **index**
- double **sumValue**
- double **sumRefValue**
- double **microInfo**
- double **divergence**
- double **sizeReduction**

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

- /home/lamarche/programming/optimal_partition/src/relative_entropy.hpp
- /home/lamarche/programming/optimal_partition/src/relative_entropy.cpp

5.51 Ring Class Reference

Inheritance diagram for Ring:



Public Member Functions

- **Ring** (int s)
- int **getIndex** (int i, int j)
- void setObjectiveFunction (ObjectiveFunction *m)

  Set the objective that one wants to optimise.
- void setRandom ()

  Randomly set the algebraic constraints for quick experiments (warning: this method is not always implemented)
- void print ()

  Print the set and its algebraic constraints.
- void buildDataStructure ()

  Build a proper data structure to represent the set and its algebraic constraints (warning: this method should always be called after instantiating and parameterising a set, and before calling any other method, such as print(), computeObjectiveValues(), computeOptimalPartition (double parameter), etc.)
- void computeObjectiveValues ()

  *Compute the value of the objective function for each feasible part (warning: setObjectiveFunction (ObjectiveFunction *objective) should have been called first)*
- void normalizeObjectiveValues ()

  Finish computing the value of the objective function for each feasible part when normalisation is required (warning: only after computeObjectiveValues() has been called)
- void printObjectiveValues ()

  Print the value of the objective function for each feasible part.
- void computeOptimalPartition (double parameter)

  Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.
- void printOptimalPartition (double parameter)

  Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

- Partition * getOptimalPartition (double parameter)

  Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)
- AbstractSet * **getRandomSet** (int size)

Public Attributes

- int **size**
- double * **values**
- double * **sumValues**
- double * **microInfos**
- double * **sizeReductions**
- double * **divergences**
- double * **optimalQualities**
- int * **optimalCuts**
- int **firstOptimalCut**
- int **lastOptimalCut**

5.51.1 Member Function Documentation

5.51.1.1 void Ring::computeOptimalPartition (double *parameter*) [virtual]

Compute a partition that fits with the algebraic constraints and that optimises the objective function that has been specified.

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.51.1.2 Partition * Ring::getOptimalPartition (double *parameter*) [virtual]

Compute and return a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Returns

: The resulting optimal partition

Implements AbstractSet.

5.51.1.3 void Ring::printOptimalPartition (double *parameter*) [virtual]

Compute and print a partition that fits with the algebraic constraints and that optimises the objective function that has been specified (warning: this method is not always implemented, but one can obtain a similar result by using getOptimalPartition (double parameter) and by calling print() on the result)

Parameters

parameter	: The parameter of the objective function to be optimised (if the objective is parametrised)

Implements AbstractSet.

5.51.1.4 void Ring::setObjectiveFunction (ObjectiveFunction * *objective*) [virtual]

Set the objective that one wants to optimise.

Parameters

objective	: The objective function itself

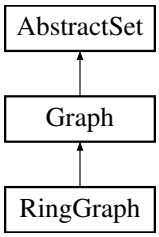
Implements AbstractSet.

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

- /home/lamarche/programming/optimal_partition/src/ring.hpp
- /home/lamarche/programming/optimal_partition/src/ring.cpp

5.52 RingGraph Class Reference

Inheritance diagram for RingGraph:



Public Member Functions

- **RingGraph** (int vNum)

Additional Inherited Members

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

- /home/lamarche/programming/optimal_partition/src/graph.hpp
- /home/lamarche/programming/optimal_partition/src/graph.cpp

5.53 Timer Class Reference

Public Member Functions

- **Timer** (char *file=0, int dimension=1, bool append=false)
- void **start** (int size, std::string text="")
- void **start** (std::vector< int > parameters, std::string text="")
- void **startTime** ()
- void **startMemory** ()

- void **stop** (std::string text="")
- void **stopTime** ()
- void **stopMemory** ()
- void **step** (std::string text="")
- void **print** (char *fName)

Public Attributes

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

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

- /home/lamarche/programming/optimal_partition/src/timer.hpp
- /home/lamarche/programming/optimal_partition/src/timer.cpp

5.54 TreeToAdd Struct Reference

Public Attributes

- Datatree * **node**
- Datatree * **nodeToAdd**
- Vertices * **vertices**
- Vertices * **adjVertices**

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

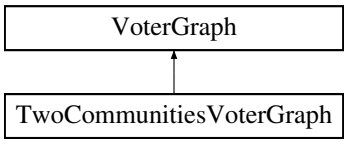
- /home/lamarche/programming/optimal_partition/src/graph.cpp

5.55 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 of 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

5.55.1 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)

5.55.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

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

5.55.3 Member Function Documentation

5.55.3.1 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 of METRIC_ACTIVE_EDGES)

Parameters

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

Returns

The computed partition over the lumped state space

5.55.3.2 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

measurement	: The measurement used to partition the lumped state space

Returns

The computed partition over the lumped state space

5.55.3.3 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

5.55.4 Member Data Documentation

5.55.4.1 std::set<VoterNode*>* TwoCommunitiesVoterGraph::community1

The set of nodes in community 1

5.55.4.2 std::set<VoterNode*>* TwoCommunitiesVoterGraph::community2

The set of nodes in community 2

5.55.4.3 double TwoCommunitiesVoterGraph::contrarian1

The contrarian rate of nodes in community 1

5.55.4.4 double TwoCommunitiesVoterGraph::contrarian2

The contrarian rate of nodes in community 2

5.55.4.5 double TwoCommunitiesVoterGraph::interRate1

The weight of edges from community 1 to community 2

5.55.4.6 double TwoCommunitiesVoterGraph::interRate2

The weight of edges from community 2 to community 1

5.55.4.7 double TwoCommunitiesVoterGraph::intraRate1

The weight of edges within community 1

5.55.4.8 double TwoCommunitiesVoterGraph::intraRate2

The weight of edges within community 2

5.55.4.9 int TwoCommunitiesVoterGraph::size1

The size of community 1

5.55.4.10 int TwoCommunitiesVoterGraph::size2

The size of community 2

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

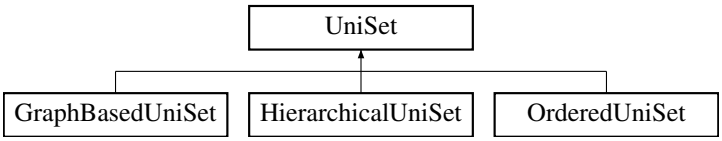
- /home/lamarche/programming/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.56 UniSet Class Reference

A uni-dimensional set of elements and its algebraic structure (feasible subsets and feasible refinements)

`#include <uni_set.hpp>`

Inheritance diagram for UniSet:



Public Member Functions

- UniSet (UniSubset *firstUniSubset)

  Constructor.
- ~UniSet ()

  Destructor.
- void buildDataStructure ()

  Build a proper data structure to represent the uni-dimensional set of elements and its algebraic structure (warning: this method should be called after construction, and before actually using the set)
- void print ()

  Print the current state of the set and its algebraic structure.

Protected Member Functions

- void initReached ()

  (Optional) A probe of a voter model that has been used to build this uni-dimensional set

Protected Attributes

- int **atomicUniSubsetNumber**
- int uniSubsetNumber

  Number of elements (i.e., atomic feasible subsets)
- UniSubset ∗∗ atomicUniSubsetArray

  Number of feasible subsets.
- UniSubset ∗∗ uniSubsetArray

  Array of pointers to all elements (i.e., atomic feasible subsets)
- UniSubset ∗ firstUniSubset

  Array of pointers to all feasible subsets.
- VoterMeasurement ∗ voterMeasurement

  Top subset in the lattice of feasible subsets (assumed to be unique and to include all feasible subsets)
- VoterProbe ∗ voterProbe

  (Optional) A probe measurement of a voter model that has been used to build this uni-dimensional set

5.56.1 Detailed Description

A uni-dimensional set of elements and its algebraic structure (feasible subsets and feasible refinements)

5.56.2 Constructor & Destructor Documentation

5.56.2.1 UniSet::UniSet (UniSubset ∗ *firstUniSubset*)

Constructor.

Parameters

firstUniSubset	: Top subset in the lattice of feasible subsets

5.56.3 Member Function Documentation

5.56.3.1 void UniSet::initReached ( ) [protected]

(Optional) A probe of a voter model that has been used to build this uni-dimensional set

Initialise the `reached` field of all feasible subsets to `false` (used by other methods to run through the algebraic structure in a recursive fashion without considering twice the same subset)

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

- /home/lamarche/programming/optimal_partition/src/uni_set.hpp
- /home/lamarche/programming/optimal_partition/src/uni_set.cpp

5.57 UniSubset Class Reference

A feasible subset associated to a uni-dimensional set of elements (UniSet)

`#include <uni_set.hpp>`

Public Member Functions

- UniSubset (int index=-1)

  Set of the refinements of this subset, that is the set of all partitions of this subset that are made of other feasible subsets; this hence properly defines the algebraic structure.
- ~UniSubset ()

  Destructor.
- void print ()

  Print the actual state of this subset.
- void printIndexSet (bool endl=false)

  Print indexes of the elements in this subset.
- void addUniSubsetSet (UniSubsetSet *uniSubsetSet)

  Add a refinement to this subset, that is a partition of this subset that is made of other feasible subsets.

Public Attributes

- UniSet * **uniSet**
- bool isAtomic

  Pointer to the set of elements to which this subset is associated.
- int atomicNum

  `true` *if and only if this subset is actually an element of the associated set (i.e., an atomic feasible subset)*
- int num

  If this subset is an element (i.e., an atomic feasible subset), identifier of this subset among all the elements of the associated set; if not, always equal to `-1`
- IndexSet * indexSet

  Identifier of this subset among all the feasible subsets of the associated set.
- UniSubsetSetSet * uniSubsetSetSet

  Indexes of all the elements in this subset (only one index / one element in the case of an atomic subset)

5.57.1 Detailed Description

A feasible subset associated to a uni-dimensional set of elements (UniSet)

5.57.2 Constructor & Destructor Documentation

5.57.2.1 UniSubset::UniSubset (int *index* = `-1`)

Set of the refinements of this subset, that is the set of all partitions of this subset that are made of other feasible subsets; this hence properly defines the algebraic structure.

Constructor

Parameters

index	: The index of the unique element in the case of an atomic subset

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

- /home/lamarche/programming/optimal_partition/src/uni_set.hpp
- /home/lamarche/programming/optimal_partition/src/uni_set.cpp

5.58 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/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.59 VoterDataSet Class Reference

Public Member Functions

- **VoterDataSet** (VoterGraph *graph, int time, int delay, int trainSize, int testSize, int trainLength, int testLength)
- PredictionDataset * **getPredictionDataset** (MultiSet *preSet, MultiSet *postSet)
- void **estimateTransitionMap** (VoterMeasurement *preM, VoterMeasurement *postM)
- void **printTransitionMap** ()
- void **print** (int size=-1)
- 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/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.60 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

5.60.1 Detailed Description

An edge of the interaction graph.

5.60.2 Constructor & Destructor Documentation

5.60.2.1 VoterEdge::VoterEdge (VoterNode * *node1*, VoterNode * *node2*, double *weight* = 1)

Constructor.

Parameters

node1	: Incoming node
node2	: Outcoming node
weight	: 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

5.60.3 Member Data Documentation

5.60.3.1 VoterNode* VoterEdge::node1

Incoming node

5.60.3.2 VoterNode* VoterEdge::node2

Outcoming node

5.60.3.3 double VoterEdge::weight

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

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

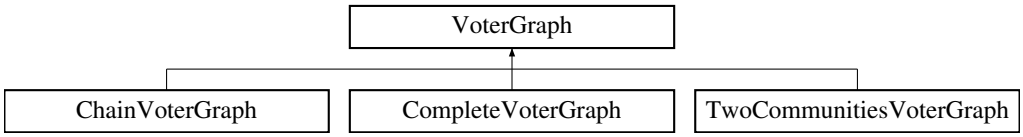
- /home/lamarche/programming/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.61 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 of 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

5.61.1 Detailed Description

The interaction graph describing a Voter Model.

5.61.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

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

5.61.3 Member Function Documentation

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

Add an edge to the graph.

Parameters

node1	: Incoming node
node2	: outcoming node
weight	: 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

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

Add a node to the graph.

Parameters

weight	: 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
contrarian	: The contrarian rate of the node to be added

Returns

The added node

5.61.3.3 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 of METRIC_ACTIVE_EDGES)

Parameters

probe	: The probe used to partition the state space
metric	: The metric of the probe (e.g., METRIC_MACRO_STATE of METRIC_ACTIVE_EDGES)

Returns

The computed partition

5.61.3.4 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

measurement	: The measurement used to partition the state space

Returns

The computed partition

5.61.3.5 MarkovProcess * VoterGraph::getMarkovProcess ( )

Build the Markov chain associated to the described Voter Model.

Returns

The computed Markov chain

5.61.4 Member Data Documentation

5.61.4.1 int VoterGraph::edgeNumber

The total number of edges in the graph

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

The set of all edges

5.61.4.3 double VoterGraph::edgeWeight

The sum of the weight of all edges

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

The map of all nodes organized by id

5.61.4.5 int VoterGraph::nodeNumber

The total number of nodes in the graph

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

The set of all nodes

5.61.4.7 double VoterGraph::nodeWeight

The sum of the weight of all nodes

5.61.4.8 MarkovProcess* VoterGraph::process

The Markov chain associated to the described Voter Model

5.61.4.9 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:

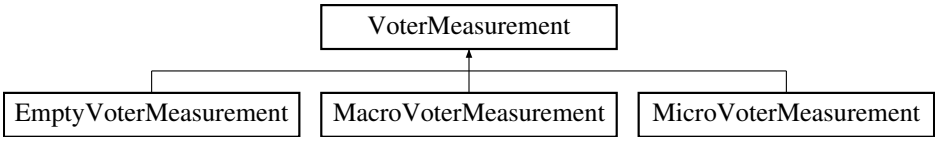
- /home/lamarche/programming/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.62 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)
- OrderedUniSet * **getOrderedUniSet** ()
- std::vector< OrderedUniSet * > * **getOrderedUniSetVector** ()
- 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**

5.62.1 Detailed Description

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

5.62.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

graph	: The interaction graph to be observed
type	: The name of the measurement

5.62.3 Member Function Documentation

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

Add a probe to the measurement.

Parameters

node	: The probe to be added
metric	: The metric associated to the added probe (e.g., METRIC_MACRO_STATE of METRIC_ACTIVE_EDGES)

5.62.3.2 void VoterMeasurement::print (bool *endl* = `false`)

Print the measurement details.

Parameters

endl	: Line break after printing if true

5.62.4 Member Data Documentation

5.62.4.1 VoterGraph* VoterMeasurement::graph

The interaction graph to be observed

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

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

5.62.4.3 Partition* VoterMeasurement::partition

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

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

The map of constituting probes organized by probe numbers

5.62.4.5 int VoterMeasurement::probeNumber

The number of probes constituting the measurement

5.62.4.6 std::string VoterMeasurement::type

The name of the measurement

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

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

5.63 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/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.64 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/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.65 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

5.65.1 Detailed Description

A node of the interaction graph.

5.65.2 Constructor & Destructor Documentation

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

Constructor.

Parameters

id	: Unique id within the graph
weight	: 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
contrarian	: Contrarian rate of the node

Author

Robin Lamarche-Perrin

Date

22/01/2015

5.65.3 Member Data Documentation

5.65.3.1 double VoterNode::contrarian

Contrarian rate of the node

5.65.3.2 int VoterNode::id

Unique id within the graph

5.65.3.3 int VoterNode::inEdgeNumber

Sum of the weight of incoming edges

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

Set of incoming edges

5.65.3.5 int VoterNode::inEdgeWeight

Sum of the weight of incoming nodes

5.65.3.6 int VoterNode::outEdgeNumber

Sum of the weight of outcoming edges

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

Set of outcoming edges

5.65.3.8 int VoterNode::outEdgeWeight

Sum of the weight of outcoming nodes

5.65.3.9 double VoterNode::weight

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

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

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

5.66 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

5.66.1 Detailed Description

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

5.66.2 Constructor & Destructor Documentation

5.66.2.1 VoterProbe::VoterProbe (VoterGraph * *graph*)

Constructor.

Parameters

graph	: The interaction graph to be observed

5.66.3 Member Function Documentation

5.66.3.1 void VoterProbe::addNode (VoterNode * *node*)

Add an observed node to the probe.

Parameters

node	: The node to be added

5.66.3.2 void VoterProbe::addNodes (unsigned long int *i*)

Add a set of observed nodes to the probe.

Parameters

graph	: 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)

5.66.3.3 void VoterProbe::print (bool *endl* = `false`)

Print the probe details.

Parameters

endl	: Line break after printing if true

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

Set the set of observed nodes.

Parameters

node	: The set to be associated

5.66.4 Member Data Documentation

5.66.4.1 VoterGraph* VoterProbe::graph

The interaction graph to be observed

5.66.4.2 int VoterProbe::nodeNumber

The number of observed nodes

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

The set of observed nodes

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

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

5.67 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/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

5.68 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/optimal_partition/src/voter_graph.hpp
- /home/lamarche/programming/optimal_partition/src/voter_graph.cpp

Chapter 6

File Documentation

6.1 /home/lamarche/programming/optimal_partition/src/abstract_set.hpp File Reference

Abstract class defining a set of elements that one wants to partition while optimising some (decomposable) objective and preserving some algebraic constraints (set of feasible parts)

```
#include "timer.hpp"
#include "objective_function.hpp"
#include "partition.hpp"
#include "dataset.hpp"
```

Classes

- class AbstractSet

  Abstract class defining a set of elements that one wants to partition while optimising some (decomposable) objective and preserving some algebraic constraints (set of feasible parts)

6.1.1 Detailed Description

Abstract class defining a set of elements that one wants to partition while optimising some (decomposable) objective and preserving some algebraic constraints (set of feasible parts)

Author

Robin Lamarche-Perrin

Date

06/11/2015

6.2 /home/lamarche/programming/optimal_partition/src/logarithmic_score.hpp File Reference

Classes to define and compute the logarithmic score function in the case of point prediction.

```
#include "objective_function.hpp"
#include "prediction_dataset.hpp"
```

Classes

- class LogarithmicScore

  Class to define and compute the logarithmic score function in the case of point prediction.
- class LogarithmicScoreValue

6.2.1 Detailed Description

Classes to define and compute the logarithmic score function in the case of point prediction.

Author

Robin Lamarche-Perrin

Date

06/11/2015

6.3 /home/lamarche/programming/optimal_partition/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

Functions

- long unsigned int **nChoosek** (int n, int k)

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/optimal_partition/src/multi_set.hpp File Reference

Classes to represent multi-dimensional sets of elements and their algebraic structure (feasible subsets and feasible refinements)

```
#include <list>
#include "uni_set.hpp"
#include "abstract_set.hpp"
#include "voter_graph.hpp"
```

Classes

- class MultiSet

  A multi-dimensional set of elements based on the Cartesian product of several uni-dimensional sets (UniSet) and their algebraic structures (feasible subsets and feasible refinements)
- class MultiSubset

Typedefs

- typedef std::list< MultiSubset * > **MultiSubsetSet**
- typedef std::list
  < MultiSubsetSet * > **MultiSubsetSetSet**

6.4.1 Detailed Description

Classes to represent multi-dimensional sets of elements and their algebraic structure (feasible subsets and feasible refinements)

Author

Robin Lamarche-Perrin

Date

06/11/2015

6.5 /home/lamarche/programming/optimal_partition/src/objective_function.hpp File Reference

Abstract class defining an objective function to be associated to a constrained set in order to define the optimisation problem that one wants to solve.

```
#include <set>
```

Classes

- class ObjectiveFunction

  Abstract class defining an objective function to be associated to a constrained set in order to define the optimisation problem that one wants to solve.
- class ObjectiveValue

Typedefs

- typedef std::set
  < ObjectiveValue * > **ObjectiveValueSet**

6.5.1 Detailed Description

Abstract class defining an objective function to be associated to a constrained set in order to define the optimisation problem that one wants to solve.

Author

Robin Lamarche-Perrin

Date

06/11/2015

6.6 /home/lamarche/programming/optimal_partition/src/prediction_dataset.hpp File Reference

Class to represent a data set for the prediction of a post-measurement from the knowledge of a pre-measurement (composed of a train set and a test set)

```
#include "multi_set.hpp"
```

Classes

- class PredictionDataset

  Class to represent a data set for the prediction of a post-measurement from the knowledge of a pre-measurement (composed of a train set and a test set)

6.6.1 Detailed Description

Class to represent a data set for the prediction of a post-measurement from the knowledge of a pre-measurement (composed of a train set and a test set)

Author

Robin Lamarche-Perrin

Date

06/11/2015

6.7 /home/lamarche/programming/optimal_partition/src/uni_set.hpp File Reference

Some classes to represent uni-dimensional sets of elements and their algebraic structure (feasible subsets and feasible refinements)

```
#include <list>
#include "bi_set.hpp"
#include "multi_set.hpp"
#include "graph.hpp"
#include "voter_graph.hpp"
```

Classes

- class UniSet

  A uni-dimensional set of elements and its algebraic structure (feasible subsets and feasible refinements)
- class OrderedUniSet

  A uni-dimensional set of elements with a total order, and such that the feasible subsets are all the intervals induced by this order.
- class HierarchicalUniSet

  A uni-dimensional set of elements structured according to a complete binary hierarchy, and such that the feasible subsets are all the nodes of the hierarchy.
- class GraphBasedUniSet
- class UniSubset

  A feasible subset associated to a uni-dimensional set of elements (UniSet)

Typedefs

- typedef std::list< UniSubset * > **UniSubsetSet**
- typedef std::list< UniSubsetSet * > **UniSubsetSetSet**
- typedef std::list< int > **IndexSet**

6.7.1 Detailed Description

Some classes to represent uni-dimensional sets of elements and their algebraic structure (feasible subsets and feasible refinements)

Author

Robin Lamarche-Perrin

Date

06/11/2015

6.8 /home/lamarche/programming/optimal_partition/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 <vector>
#include "markov_process.hpp"
#include "partition.hpp"
#include "uni_set.hpp"
#include "multi_set.hpp"
#include "prediction_dataset.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.

Functions

- 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=MACRO_STATE, int binning=0)

6.8.1 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

6.8.2 Enumeration Type Documentation

6.8.2.1 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)

6.8.2.2 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 outcoming nodes

UPDATE_EDGES Edge-driven interactions: An edge is chosen at each simulation step, its incoming node acts on its outcoming node

6.8.2.3 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

~AbstractSet
    AbstractSet, 13
~MultiSet
    MultiSet, 47
/home/lamarche/programming/optimal_partition/src/abstract-
        _set.hpp, 91
/home/lamarche/programming/optimal_partition/src/logarithmic-
        _score.hpp, 91
/home/lamarche/programming/optimal_partition/src/markov-
        _process.hpp, 92
/home/lamarche/programming/optimal_partition/src/multi-
        _set.hpp, 93
/home/lamarche/programming/optimal_partition/src/objective-
        _function.hpp, 93
/home/lamarche/programming/optimal_partition/src/prediction-
        _dataset.hpp, 94
/home/lamarche/programming/optimal_partition/src/uni-
        _set.hpp, 94
/home/lamarche/programming/optimal_partition/src/voter-
        _graph.hpp, 95

ACTIVE_EDGES
    voter_graph.hpp, 98
AbstractSet, 11
    ~AbstractSet, 13
    computeOptimalPartition, 13
    getOptimalPartition, 13
    getOptimalPartitionList, 13
    printOptimalPartition, 13
    printOptimalPartitionList, 15
    printOptimalPartitionListInCSV, 15
    setObjectiveFunction, 15
addEdge
    VoterGraph, 80
addNode
    VoterGraph, 80
    VoterProbe, 87
addNodes
    VoterProbe, 87
addProbe
    VoterMeasurement, 83
addTestValue
    PredictionDataset, 65
addTrainValue
    PredictionDataset, 65

BiPart, 15
BiSet, 16
    computeOptimalPartition, 17
    getOptimalPartition, 17
    printOptimalPartition, 17
    setObjectiveFunction, 18
BiSubset, 18
BottleneckObjectiveValue, 19

ChainVoterGraph, 19
community1
    TwoCommunitiesVoterGraph, 73
community2
    TwoCommunitiesVoterGraph, 73
CompleteGraph, 20
CompleteVoterGraph, 20
    CompleteVoterGraph, 21
    CompleteVoterGraph, 21
computeOptimalPartition
    AbstractSet, 13
    BiSet, 17
    Graph, 26
    HierarchicalHierarchicalSet, 29
    HierarchicalOrderedSet, 31
    HierarchicalSet, 33
    MultiSet, 47
    NHOSet, 51
    NonconstrainedOrderedSet, 53
    NonconstrainedSet, 55
    OrderedSet, 60
    Ring, 69
computeStationaryDistribution
    MarkovProcess, 41
computeTrajectory
    MarkovProcess, 42
contrarian
    VoterNode, 86
contrarian1
    TwoCommunitiesVoterGraph, 73
contrarian2
    TwoCommunitiesVoterGraph, 73

DataPointStruct, 21
Dataset, 21
Datatree, 22
distribution
    MarkovProcess, 42
distributions
    MarkovProcess, 42

edgeNumber
    VoterGraph, 81
edgeSet
    VoterGraph, 81

edgeWeight
VoterGraph, 81
EmptyVoterMeasurement, 23
EmptyVoterMeasurement, 23
EmptyVoterMeasurement, 23

FiliformGraph, 24

getAtomicMultiSubset
MultiSet, 47
getCompactMarkovPartition
TwoCommunitiesVoterGraph, 72, 73
getCompactMarkovProcess
TwoCommunitiesVoterGraph, 73
getMarkovPartition
VoterGraph, 80, 81
getMarkovProcess
VoterGraph, 81
getOptimalPartition
AbstractSet, 13
BiSet, 17
Graph, 26
HierarchicalHierarchicalSet, 29
HierarchicalOrderedSet, 31
HierarchicalSet, 33
MultiSet, 47
NHOSet, 52
NonconstrainedOrderedSet, 53
NonconstrainedSet, 55
OrderedSet, 60
Ring, 69
getOptimalPartitionList
AbstractSet, 13
getRandomAtomicMultiSubset
MultiSet, 48
Graph, 24
computeOptimalPartition, 26
getOptimalPartition, 26
printOptimalPartition, 26
setObjectiveFunction, 26
graph
VoterMeasurement, 83
VoterProbe, 88
GraphBasedUniSet, 26
GraphComponent, 27

HHNode, 28
HNode, 35
HONode, 36
HierarchicalHierarchicalSet, 28
computeOptimalPartition, 29
getOptimalPartition, 29
printOptimalPartition, 30
setObjectiveFunction, 30
HierarchicalOrderedSet, 30
computeOptimalPartition, 31
getOptimalPartition, 31
printOptimalPartition, 31
setObjectiveFunction, 32
HierarchicalSet, 32
computeOptimalPartition, 33
getOptimalPartition, 33
printOptimalPartition, 33
setObjectiveFunction, 34
HierarchicalUniSet, 34
HierarchicalUniSet, 35
HierarchicalUniSet, 35

id
VoterNode, 86
inEdgeNumber
VoterNode, 86
inEdgeSet
VoterNode, 86
inEdgeWeight
VoterNode, 86
InformationBottleneck, 36
initReached
UniSet, 75
interRate1
TwoCommunitiesVoterGraph, 73
interRate2
TwoCommunitiesVoterGraph, 73
intraRate1
TwoCommunitiesVoterGraph, 74
intraRate2
TwoCommunitiesVoterGraph, 74

lastDelay
MarkovProcess, 42
lastTime
MarkovProcess, 43
LogarithmicScore, 37
LogarithmicScore, 38
LogarithmicScore, 38
LogarithmicScoreValue, 38

M_AGENT1
voter_graph.hpp, 97
M_AGENT1_ALLSIZES1
voter_graph.hpp, 98
M_AGENT1_MACRO
voter_graph.hpp, 98
M_AGENT1_MESO1
voter_graph.hpp, 98
M_AGENT1_MESO1_MESO2
voter_graph.hpp, 98
M_AGENT1_MESO2
voter_graph.hpp, 98
M_AGENT1_SOMESIZES1
voter_graph.hpp, 98
M_ALLSIZES1
voter_graph.hpp, 98
M_EMPTY
voter_graph.hpp, 98
M_MACRO
voter_graph.hpp, 98
M_MESO1

voter_graph.hpp, 97
M_MESO1_MESO2
voter_graph.hpp, 98
M_MESO2
voter_graph.hpp, 97
M_MICRO
voter_graph.hpp, 97
M_SOMESIZES1
voter_graph.hpp, 98
MACRO_STATE
voter_graph.hpp, 98
MAJORITY
voter_graph.hpp, 98
MacroVoterMeasurement, 39
MacroVoterMeasurement, 39
MacroVoterMeasurement, 39
MarkovDataSet, 39
MarkovProcess, 40
computeStationaryDistribution, 41
computeTrajectory, 42
distribution, 42
distributions, 42
lastDelay, 42
lastTime, 43
MarkovProcess, 41
MarkovProcess, 41
setDistribution, 42
setTransition, 42
size, 43
transition, 43
transitions, 43
MarkovTrajectory, 43
MeasurementType
voter_graph.hpp, 97
metricMap
VoterMeasurement, 83
MicroVoterMeasurement, 43
MicroVoterMeasurement, 44
MicroVoterMeasurement, 44
MultiPart, 44
MultiSet, 45
∼MultiSet, 47
computeOptimalPartition, 47
getAtomicMultiSubset, 47
getOptimalPartition, 47
getRandomAtomicMultiSubset, 48
MultiSet, 46, 47
MultiSet, 46, 47
printOptimalPartition, 48
setObjectiveFunction, 48
MultiSubset, 48

NHONode, 49
NHOSet, 50
computeOptimalPartition, 51
getOptimalPartition, 52
printOptimalPartition, 52
setObjectiveFunction, 52
node1
VoterEdge, 78
node2
VoterEdge, 78
nodeMap
VoterGraph, 81
nodeNumber
VoterGraph, 81
VoterProbe, 88
nodeSet
VoterGraph, 81
VoterProbe, 88
nodeWeight
VoterGraph, 81
NonconstrainedOrderedSet, 52
computeOptimalPartition, 53
getOptimalPartition, 53
printOptimalPartition, 54
setObjectiveFunction, 54
NonconstrainedSet, 54
computeOptimalPartition, 55
getOptimalPartition, 55
printOptimalPartition, 56
setObjectiveFunction, 56

ObjectiveFunction, 56
ObjectiveFunction, 57
ObjectiveFunction, 57
ObjectiveValue, 57
OrderedDatatree, 58
OrderedPartition, 59
OrderedSet, 59
computeOptimalPartition, 60
getOptimalPartition, 60
printOptimalPartition, 60
setObjectiveFunction, 61
OrderedUniSet, 61
OrderedUniSet, 62
OrderedUniSet, 62
outEdgeNumber
VoterNode, 86
outEdgeSet
VoterNode, 86
outEdgeWeight
VoterNode, 86

Part, 62
Partition, 63
partition
VoterMeasurement, 83
PredictionDataset, 63
addTestValue, 65
addTrainValue, 65
PredictionDataset, 64
PredictionDataset, 64
print, 66
print
PredictionDataset, 66
VoterMeasurement, 83
VoterProbe, 88

printOptimalPartition
AbstractSet, 13
BiSet, 17
Graph, 26
HierarchicalHierarchicalSet, 30
HierarchicalOrderedSet, 31
HierarchicalSet, 33
MultiSet, 48
NHOSet, 52
NonconstrainedOrderedSet, 54
NonconstrainedSet, 56
OrderedSet, 60
Ring, 69
printOptimalPartitionList
AbstractSet, 15
printOptimalPartitionListInCSV
AbstractSet, 15
probeMap
VoterMeasurement, 83
probeNumber
VoterMeasurement, 84
process
VoterGraph, 82

RandomGraph, 66
RelativeEntropy, 66
RelativeObjectiveValue, 67
Ring, 68
computeOptimalPartition, 69
getOptimalPartition, 69
printOptimalPartition, 69
setObjectiveFunction, 70
RingGraph, 70

setDistribution
MarkovProcess, 42
setNodeSet
VoterProbe, 88
setObjectiveFunction
AbstractSet, 15
BiSet, 18
Graph, 26
HierarchicalHierarchicalSet, 30
HierarchicalOrderedSet, 32
HierarchicalSet, 34
MultiSet, 48
NHOSet, 52
NonconstrainedOrderedSet, 54
NonconstrainedSet, 56
OrderedSet, 61
Ring, 70
setTransition
MarkovProcess, 42
size
MarkovProcess, 43
size1
TwoCommunitiesVoterGraph, 74
size2
TwoCommunitiesVoterGraph, 74

Timer, 70
transition
MarkovProcess, 43
transitions
MarkovProcess, 43
TreeToAdd, 71
TwoCommunitiesVoterGraph, 71
community1, 73
community2, 73
contrarian1, 73
contrarian2, 73
getCompactMarkovPartition, 72, 73
getCompactMarkovProcess, 73
interRate1, 73
interRate2, 73
intraRate1, 74
intraRate2, 74
size1, 74
size2, 74
TwoCommunitiesVoterGraph, 72
TwoCommunitiesVoterGraph, 72
type
VoterMeasurement, 84

UPDATE_EDGES
voter_graph.hpp, 98
UPDATE_NODES
voter_graph.hpp, 98
UniSet, 74
initReached, 75
UniSet, 75
UniSet, 75
UniSubset, 75
UniSubset, 76
UniSubset, 76
UpdateProcess
voter_graph.hpp, 98
updateProcess
VoterGraph, 82

voter_graph.hpp
ACTIVE_EDGES, 98
M_AGENT1, 97
M_AGENT1_ALLSIZES1, 98
M_AGENT1_MACRO, 98
M_AGENT1_MESO1, 98
M_AGENT1_MESO1_MESO2, 98
M_AGENT1_MESO2, 98
M_AGENT1_SOMESIZES1, 98
M_ALLSIZES1, 98
M_EMPTY, 98
M_MACRO, 98
M_MESO1, 97
M_MESO1_MESO2, 98
M_MESO2, 97
M_MICRO, 97
M_SOMESIZES1, 98
MACRO_STATE, 98
MAJORITY, 98

UPDATE_EDGES, 98
UPDATE_NODES, 98
voter_graph.hpp
MeasurementType, 97
UpdateProcess, 98
VoterMetric, 98
VoterBinning, 77
VoterDataSet, 77
VoterEdge, 78
node1, 78
node2, 78
VoterEdge, 78
VoterEdge, 78
weight, 78
VoterGraph, 79
addEdge, 80
addNode, 80
edgeNumber, 81
edgeSet, 81
edgeWeight, 81
getMarkovPartition, 80, 81
getMarkovProcess, 81
nodeMap, 81
nodeNumber, 81
nodeSet, 81
nodeWeight, 81
process, 82
updateProcess, 82
VoterGraph, 80
VoterGraph, 80
VoterMeasurement, 82
addProbe, 83
graph, 83
metricMap, 83
partition, 83
print, 83
probeMap, 83
probeNumber, 84
type, 84
VoterMeasurement, 83
VoterMeasurement, 83
VoterMeasurementState, 84
VoterMeasurementTrajectory, 84
VoterMetric
voter_graph.hpp, 98
VoterNode, 85
contrarian, 86
id, 86
inEdgeNumber, 86
inEdgeSet, 86
inEdgeWeight, 86
outEdgeNumber, 86
outEdgeSet, 86
outEdgeWeight, 86
VoterNode, 85
VoterNode, 85
weight, 86
VoterProbe, 86
addNode, 87
addNodes, 87
graph, 88
nodeNumber, 88
nodeSet, 88
print, 88
setNodeSet, 88
VoterProbe, 87
VoterProbe, 87
VoterState, 88
VoterTrajectory, 89

weight
VoterEdge, 78
VoterNode, 86