libDAIProject

1.0

Generated by Doxygen 1.9.1

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 BipartiteGraphHandler Class Reference . . . 5

3.1.1 Detailed Description . . . 5

3.1.2 Member Function Documentation . . . 6

3.1.2.1 addEdge() . . . 6

3.1.2.2 addNode() . . . 6

3.1.2.3 isConnected() . . . 6

3.1.2.4 removeEdge() . . . 7

3.1.2.5 removeNode() . . . 7

3.2 FactorGraphBuilder Class Reference . . . 7

3.2.1 Detailed Description . . . 8

3.2.2 Member Function Documentation . . . 8

3.2.2.1 buildGraph() . . . 8

3.3 ImageProcessor Class Reference . . . 9

3.3.1 Detailed Description . . . 9

3.3.2 Member Function Documentation . . . 9

3.3.2.1 calculateHistogram() . . . 9

3.3.2.2 constructDifferenceImage() . . . 10

3.3.2.3 displayAndSaveEvidence() . . . 10

3.3.2.4 displayImage() . . . 11

3.3.2.5 normalizeDifferenceImage() . . . 11

3.3.2.6 readImage() . . . 12

3.4 ImageSegmentation Class Reference . . . 12

3.4.1 Detailed Description . . . 12

3.5 InferenceSolver Class Reference . . . 13

3.5.1 Detailed Description . . . 13

3.5.2 Member Function Documentation . . . 13

3.5.2.1 runInference() . . . 13

3.6 Settings Class Reference . . . 14

3.6.1 Detailed Description . . . 14

3.6.2 Member Function Documentation . . . 15

3.6.2.1 initialize() . . . 15

3.7 dai::Sprinkler Class Reference . . . 15

3.7.1 Detailed Description . . . 15

4 File Documentation 17

4.1 src/main.cpp File Reference . . . 17

4.1.1 Detailed Description 17

Index **19**

Chapter 1

Class Index

1.1 Class List

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

BipartiteGraphHandler
: Handles operations and visualizations for a bipartite graph 5

FactorGraphBuilder
: A class to build factor graphs based on image data using the Ising model 7

ImageProcessor
: Provides functionalities for reading, displaying, and processing images 9

ImageSegmentation
: Class to handle image segmentation processes using factor graphs and inference techniques . 12

InferenceSolver
: Class designed to handle the inference process on factor graphs using various algorithms . . . 13

Settings
: Manages configuration settings for various modules within the application 14

dai::Sprinkler
: Implements a Bayesian network simulation of the Sprinkler example using factor graphs 15

Chapter 2

File Index

2.1 File List

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

src/main.cpp
Main entry point for a multi-module application demonstrating various functionalities 17
src/**Settings.cpp** . . . . . **??**
src/**Settings.hpp** . . . . . **??**
src/BipartiteGraph/**BipartiteGraphHandler.cpp** . . . . . **??**
src/BipartiteGraph/**BipartiteGraphHandler.hpp** . . . . . **??**
src/ImageSegmentation/**FactorGraphBuilder.cpp** . . . . . **??**
src/ImageSegmentation/**FactorGraphBuilder.hpp** . . . . . **??**
src/ImageSegmentation/**ImageProcessor.cpp** . . . . . **??**
src/ImageSegmentation/**ImageProcessor.hpp** . . . . . **??**
src/ImageSegmentation/**ImageSegmentation.cpp** . . . . . **??**
src/ImageSegmentation/**ImageSegmentation.hpp** . . . . . **??**
src/ImageSegmentation/**InferenceSolver.cpp** . . . . . **??**
src/ImageSegmentation/**InferenceSolver.hpp** . . . . . **??**
src/Sprinkler/**Sprinkler.cpp** . . . . . **??**
src/Sprinkler/**Sprinkler.hpp** . . . . . **??**

Chapter 3

Class Documentation

3.1 BipartiteGraphHandler Class Reference

Handles operations and visualizations for a bipartite graph.

`#include <BipartiteGraphHandler.hpp>`

Public Member Functions

- BipartiteGraphHandler ()
  Default constructor initializes the bipartite graph with predefined nodes and edges.
- void operator() ()
  Invokes the main functionality of the bipartite graph handler.
- void addEdge (size_t node1, size_t node2)
  Adds an edge between two nodes if it doesn't already exist.
- void removeEdge (size_t node1, size_t node2)
  Removes an edge between two nodes if it exists.
- void addNode (size_t nodeType)
  Adds a new node of a specified type to the graph.
- void removeNode (size_t node, size_t nodeType)
  Removes a node of a specified type from the graph if it exists.
- bool isConnected (size_t node1, size_t node2)
  Checks if there is a path connecting two nodes.
- void displayGraphInfo ()
  Displays basic information about the graph such as number of nodes and edges.
- void displayConnectivity ()
  Displays the connectivity information of the graph.
- void displayGraphWithGraphviz ()
  Visualizes the bipartite graph using Graphviz and saves the visualization to a file.

3.1.1 Detailed Description

Handles operations and visualizations for a bipartite graph.

This class manages a bipartite graph and provides functionalities for adding and removing nodes and edges, checking connectivity, and visualizing the graph using Graphviz. It supports various operations to manipulate the graph structure and provides detailed visual feedback on the graph's current state.

Definition at line 19 of file BipartiteGraphHandler.hpp.

3.1.2 Member Function Documentation

3.1.2.1 addEdge()

```
void BipartiteGraphHandler::addEdge (
            size_t node1,
            size_t node2 )
```

Adds an edge between two nodes if it doesn't already exist.

Parameters

node1	Index of the first node.
node2	Index of the second node.

Definition at line 33 of file BipartiteGraphHandler.cpp.

3.1.2.2 addNode()

```
void BipartiteGraphHandler::addNode (
            size_t nodeType )
```

Adds a new node of a specified type to the graph.

Parameters

nodeType	Type of the node (1 for type 1, 2 for type 2).

Definition at line 58 of file BipartiteGraphHandler.cpp.

3.1.2.3 isConnected()

```
bool BipartiteGraphHandler::isConnected (
            size_t node1,
            size_t node2 )
```

Checks if there is a path connecting two nodes.

Parameters

node1	Index of the first node.
node2	Index of the second node.

Returns

bool True if there is a path, false otherwise.

Definition at line 93 of file BipartiteGraphHandler.cpp.

3.1.2.4 removeEdge()

```
void BipartiteGraphHandler::removeEdge (
            size_t node1,
            size_t node2 )
```

Removes an edge between two nodes if it exists.

Parameters

node1	Index of the first node.
node2	Index of the second node.

Definition at line 46 of file BipartiteGraphHandler.cpp.

3.1.2.5 removeNode()

```
void BipartiteGraphHandler::removeNode (
            size_t node,
            size_t nodeType )
```

Removes a node of a specified type from the graph if it exists.

Parameters

node	Index of the node to be removed.
nodeType	Type of the node (1 for type 1, 2 for type 2).

Definition at line 75 of file BipartiteGraphHandler.cpp.

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

- src/BipartiteGraph/BipartiteGraphHandler.hpp
- src/BipartiteGraph/BipartiteGraphHandler.cpp

3.2 FactorGraphBuilder Class Reference

A class to build factor graphs based on image data using the Ising model.

`#include <FactorGraphBuilder.hpp>`

Public Member Functions

- FactorGraph buildGraph (const CImg< unsigned char > &img, double J, double th_min, double th_max, double scale, double pbg, CImg< unsigned char > &evidence)

  Builds a factor graph from a given image using specified parameters.
- const std::vector< Var > & **getVars** () const

3.2.1 Detailed Description

A class to build factor graphs based on image data using the Ising model.

This class provides functionality to construct a factor graph from a given image, where each pixel's interaction is modeled using the Ising model. It allows for the adjustment of interaction strength, threshold levels, and scaling factors to manipulate the model's sensitivity to changes in pixel intensity.

Definition at line 19 of file FactorGraphBuilder.hpp.

3.2.2 Member Function Documentation

3.2.2.1 buildGraph()

```
FactorGraph FactorGraphBuilder::buildGraph (
            const CImg< unsigned char > & img,
            double J,
            double th_min,
            double th_max,
            double scale,
            double pbg,
            CImg< unsigned char > & evidence )
```

Builds a factor graph from a given image using specified parameters.

This method constructs a factor graph that models the relationship between image pixels and their intensity transformations based on the Ising model. The graph is built by defining variables and factors that represent binary states and their pairwise relationships, respectively. It also computes an evidence image that highlights significant intensity transitions based on the threshold level derived from the pixel intensities histogram.

Parameters

img	Reference to a CImg<unsigned char> object representing the input image.
J	Coupling constant of the Ising model representing interaction strength between pixels.
th_min	Minimum threshold for the factor intensity.
th_max	Maximum threshold for the factor intensity.
scale	Scaling factor used in the hyperbolic tangent function for threshold calculation.
pbg	Percentage of background pixels used to determine the histogram threshold.
evidence	Reference to a CImg<unsigned char> object to store the evidence image showing significant transitions.

Returns

FactorGraph An object representing the constructed factor graph.

Definition at line 22 of file FactorGraphBuilder.cpp.

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

- src/ImageSegmentation/FactorGraphBuilder.hpp
- src/ImageSegmentation/FactorGraphBuilder.cpp

3.3 ImageProcessor Class Reference

Provides functionalities for reading, displaying, and processing images.

`#include <ImageProcessor.hpp>`

Public Member Functions

- CImg< unsigned char > readImage (const char *filename)
  Reads an image from a file.
- void displayImage (const CImg< unsigned char > &image, const char *title)
  Displays an image with a given title.
- CImg< float > calculateHistogram (const CImg< unsigned char > &image)
  Calculates the histogram of an image.
- CImg< int > constructDifferenceImage (const CImg< unsigned char > &img1, const CImg< unsigned char > &img2)
  Constructs a difference image between two images.
- void normalizeDifferenceImage (CImg< int > &img, const CImg< unsigned char > &ref)
  Normalizes a difference image based on a reference image.
- void displayAndSaveEvidence (const CImg< unsigned char > &image, const char *title, const char *filename)
  Displays and saves an image.

3.3.1 Detailed Description

Provides functionalities for reading, displaying, and processing images.

This class utilizes the CImg library to handle various image processing operations, such as reading images from files, displaying images, calculating histograms, constructing difference images, normalizing images, and saving images after displaying.

Definition at line 15 of file ImageProcessor.hpp.

3.3.2 Member Function Documentation

3.3.2.1 calculateHistogram()

```
CImg< float > ImageProcessor::calculateHistogram (
            const CImg< unsigned char > & image )
```

Calculates the histogram of an image.

This method computes the histogram of the image's first channel (assumed to be grayscale) using 256 bins covering the range from 0 to 255. It returns the histogram as a CImg<float> object.

Parameters

image	Constant reference to a CImg<unsigned char> object representing the image.

Returns

CImg<float> Histogram of the image.

Definition at line 43 of file ImageProcessor.cpp.

3.3.2.2 constructDifferenceImage()

```
CImg< int > ImageProcessor::constructDifferenceImage (
            const CImg< unsigned char > & img1,
            const CImg< unsigned char > & img2 )
```

Constructs a difference image between two images.

This method calculates the absolute difference between two given images pixel by pixel and returns the resulting difference image. This can be used to highlight changes between two images of the same size.

Parameters

img1	Constant reference to a CImg<unsigned char> representing the first image.
img2	Constant reference to a CImg<unsigned char> representing the second image.

Returns

CImg<int> Difference image calculated from the two input images.

Definition at line 58 of file ImageProcessor.cpp.

3.3.2.3 displayAndSaveEvidence()

```
void ImageProcessor::displayAndSaveEvidence (
            const CImg< unsigned char > & image,
            const char * title,
            const char * filename )
```

Displays and saves an image.

This method displays an image with the specified title and saves it to a file in JPEG format with a quality of 100. The method prints both actions to the console for traceability.

Parameters

image	Constant reference to a CImg<unsigned char> object representing the image.
title	Pointer to a char array containing the title of the window in which the image is displayed.
filename	Pointer to a char array containing the name of the file to save the image.

Definition at line 104 of file ImageProcessor.cpp.

3.3.2.4 displayImage()

```
void ImageProcessor::displayImage (
            const CImg< unsigned char > & image,
            const char * title )
```

Displays an image with a given title.

This method displays the provided image in a window with the specified title. It uses the display method from the CImg library.

Parameters

image	Constant reference to a CImg<unsigned char> object representing the image to be displayed.
title	Pointer to a char array containing the title of the window.

Definition at line 29 of file ImageProcessor.cpp.

3.3.2.5 normalizeDifferenceImage()

```
void ImageProcessor::normalizeDifferenceImage (
            CImg< int > & img,
            const CImg< unsigned char > & ref )
```

Normalizes a difference image based on a reference image.

This method adjusts the intensity of a difference image based on the average intensity of a reference image. It performs normalization such that regions with higher intensity in the reference have less pronounced differences.

Parameters

img	Reference to a CImg<int> object representing the difference image to be normalized.
ref	Constant reference to a CImg<unsigned char> representing the reference image.

Definition at line 76 of file ImageProcessor.cpp.

3.3.2.6 readImage()

```
CImg< unsigned char > ImageProcessor::readImage (
            const char * filename )
```

Reads an image from a file.

This method loads an image from the specified file using the CImg library. It also prints a message to the console indicating the file being read. This function assumes the image is stored in an unsigned char format.

Parameters

filename	Pointer to a char array containing the name of the file.

Returns

CImg<unsigned char> An image object loaded from the specified file.

Definition at line 14 of file ImageProcessor.cpp.

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

- src/ImageSegmentation/ImageProcessor.hpp
- src/ImageSegmentation/ImageProcessor.cpp

3.4 ImageSegmentation Class Reference

Class to handle image segmentation processes using factor graphs and inference techniques.

```
#include <ImageSegmentation.hpp>
```

Public Member Functions

- void **operator()** ()

3.4.1 Detailed Description

Class to handle image segmentation processes using factor graphs and inference techniques.

This class encapsulates the entire process of segmenting an image based on differences between two images. It utilizes factor graphs to model the problem and applies inference methods to segment the images. The class is designed to be used as a callable object, allowing it to be invoked easily where needed.

Definition at line 17 of file ImageSegmentation.hpp.

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

- src/ImageSegmentation/ImageSegmentation.hpp
- src/ImageSegmentation/ImageSegmentation.cpp

3.5 InferenceSolver Class Reference

Class designed to handle the inference process on factor graphs using various algorithms.

`#include <InferenceSolver.hpp>`

Public Member Functions

- double runInference (FactorGraph &fg, std::string algOpts, size_t maxIter, double tol, std::vector< double > &m, size_t dimx, size_t dimy, CImgDisplay &disp)

  Executes the inference algorithm on a given factor graph to estimate the variable states.

3.5.1 Detailed Description

Class designed to handle the inference process on factor graphs using various algorithms.

This class encapsulates methods for running inference algorithms on factor graphs to solve problems such as image segmentation. It utilizes the libDAI library for managing the factor graphs and inference algorithms, and the CImg library for displaying the results. The class provides a method to run the inference, visualize the process, and output the results.

Definition at line 21 of file InferenceSolver.hpp.

3.5.2 Member Function Documentation

3.5.2.1 runInference()

```
double InferenceSolver::runInference (
            FactorGraph & fg,
            std::string algOpts,
            size_t maxIter,
            double tol,
            std::vector< double > & m,
            size_t dimx,
            size_t dimy,
            CImgDisplay & disp )
```

Executes the inference algorithm on a given factor graph to estimate the variable states.

This method sets up and runs an inference algorithm specified by a string containing options for the algorithm. It iteratively updates the beliefs about the variables' states, visualizes these beliefs, and monitors the convergence of the algorithm until a specified tolerance level is reached or the maximum number of iterations is exceeded. The method utilizes visual feedback for each iteration by displaying and saving the current state of beliefs.

Parameters

fg	Reference to a FactorGraph object on which inference is to be performed.
algOpts	String specifying the options for the inference algorithm.
maxIter	Maximum number of iterations the inference algorithm should run.
tol	Tolerance level for convergence of the inference algorithm.
m	Reference to a vector<double> that will store the final magnetizations (beliefs) of the variables.
dimx	Width of the image representing the beliefs for visualization.

Returns

double The maximum difference between iterations of the algorithm indicating convergence level.

Definition at line 24 of file InferenceSolver.cpp.

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

- src/ImageSegmentation/InferenceSolver.hpp
- src/ImageSegmentation/InferenceSolver.cpp

3.6 Settings Class Reference

Manages configuration settings for various modules within the application.

`#include <Settings.hpp>`

Static Public Member Functions

- static void initialize (int argc, char ∗∗argv)

  Initializes settings from command-line arguments.

Static Public Attributes

- static const char ∗ **file_i1** = nullptr
- static const char ∗ **file_i2** = nullptr
- static const char ∗ **file_o1** = nullptr
- static const char ∗ **file_o2** = nullptr
- static const char ∗ **belief** = nullptr
- static const char ∗ **bp_graphviz** = nullptr
- static const char ∗ **sprinkler_fg** = nullptr
- static double **J** = 0.0
- static double **th_min** = 0.0
- static double **th_max** = 0.0
- static double **scale** = 0.0
- static double **pbg** = 0.0
- static const char ∗ **infname** = nullptr
- static size_t **maxiter** = 0
- static double **tol** = 0.0
- static const char ∗ **file_fg** = nullptr
- static Operation **operation** = Operation::ImageSegmentation

3.6.1 Detailed Description

Manages configuration settings for various modules within the application.

This class contains static member variables that hold configuration settings such as file paths, algorithm parameters, and operation modes. These settings are initialized from command-line arguments using the CImg library functions. The class supports different operations such as Image Segmentation, Sprinkler Bayesian network, and Bipartite Graph handling, each with its own set of configurations.

Definition at line 23 of file Settings.hpp.

3.6.2 Member Function Documentation

3.6.2.1 initialize()

```
void Settings::initialize (
            int argc,
            char ** argv )  [static]
```

Initializes settings from command-line arguments.

This method parses command-line arguments to set up various configuration parameters for the application. It uses the CImg library's command-line parsing utility to extract values for each setting. The method supports switching between different operations and adjusts settings accordingly based on the provided arguments.

Parameters

argc	Number of command-line arguments.
argv	Array of command-line argument strings.

Definition at line 33 of file Settings.cpp.

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

- src/Settings.hpp
- src/Settings.cpp

3.7 dai::Sprinkler Class Reference

Implements a Bayesian network simulation of the Sprinkler example using factor graphs.

`#include <Sprinkler.hpp>`

Public Member Functions

- void **operator()** ()

3.7.1 Detailed Description

Implements a Bayesian network simulation of the Sprinkler example using factor graphs.

This class encapsulates the functionality to simulate the Sprinkler Bayesian network, commonly used in probabilistic graphical models for teaching and demonstration purposes. The network includes variables such as Cloudy, Sprinkler, Rain, and Wet Grass, with appropriate conditional probabilities. The class uses the libDAI library for creating and manipulating factor graphs and provides a method to run the entire simulation, which calculates various probabilities and outputs the results.

Definition at line 21 of file Sprinkler.hpp.

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

- src/Sprinkler/Sprinkler.hpp
- src/Sprinkler/Sprinkler.cpp

Chapter 4

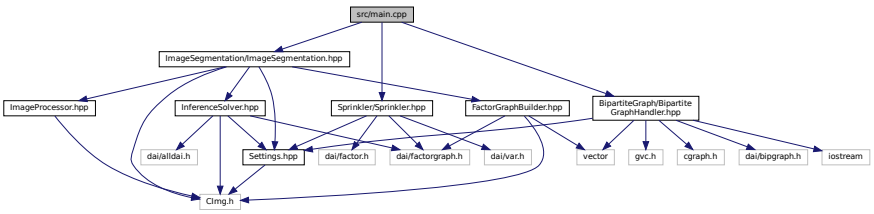
File Documentation

4.1 src/main.cpp File Reference

Main entry point for a multi-module application demonstrating various functionalities.

```
#include "BipartiteGraph/BipartiteGraphHandler.hpp"
#include "ImageSegmentation/ImageSegmentation.hpp"
#include "Sprinkler/Sprinkler.hpp"
```

Include dependency graph for main.cpp:



Functions

- int **main** (int argc, char ∗∗argv)

4.1.1 Detailed Description

Main entry point for a multi-module application demonstrating various functionalities.

This program initializes settings from command line arguments and executes one of several operations based on those settings. The operations include image segmentation, a Sprinkler Bayesian network simulation, and handling of a bipartite graph. Each operation is encapsulated in its own class and is executed depending on the user-specified option. This approach demonstrates modularity and the use of command line parameters to control application behavior.

Parameters

argc	Number of command line arguments.
argv	Array of command line arguments.

Returns

int Returns 0 upon successful execution, signaling normal program termination.

Index

addEdge
    BipartiteGraphHandler, 6
addNode
    BipartiteGraphHandler, 6

BipartiteGraphHandler, 5
    addEdge, 6
    addNode, 6
    isConnected, 6
    removeEdge, 7
    removeNode, 7
buildGraph
    FactorGraphBuilder, 8

calculateHistogram
    ImageProcessor, 9
constructDifferenceImage
    ImageProcessor, 10

dai::Sprinkler, 15
displayAndSaveEvidence
    ImageProcessor, 10
displayImage
    ImageProcessor, 11

FactorGraphBuilder, 7
    buildGraph, 8

ImageProcessor, 9
    calculateHistogram, 9
    constructDifferenceImage, 10
    displayAndSaveEvidence, 10
    displayImage, 11
    normalizeDifferenceImage, 11
    readImage, 11
ImageSegmentation, 12
InferenceSolver, 13
    runInference, 13
initialize
    Settings, 15
isConnected
    BipartiteGraphHandler, 6

normalizeDifferenceImage
    ImageProcessor, 11

readImage
    ImageProcessor, 11
removeEdge
    BipartiteGraphHandler, 7
removeNode
    BipartiteGraphHandler, 7
runInference
    InferenceSolver, 13

Settings, 14
    initialize, 15
src/main.cpp, 17