Fixed Packet Solver

Generated by Doxygen 1.9.2

Chapter 1

Directory Structure

This project has the following files:

- `formatter.cpp`: Converts raw data into required format.
  - Input : Raw data (List of edges)
  - Output : Formatted data (Input for NaivePLW)
  - Status : Complete
- `NaivePLW.cpp` : Implementation of the algorithm given by our paper.
  - Input : Hitting index; unweighted, undirected Graph; Edge weight function, **b** of L**x** = **b**, ϵ: error param
  - Output : Column vector **x**
  - Status : Working on Phase-2

1.1 Input format for PLW-SOLVE

The Input for PLW-SOLVE is formatted as:

```
<G> // G is the undirected, weighted graph corresponding to L
<b> // b is the column vector in Lx = b
<epsilon> // epsilon is the error parameter for the algorithm's termination
```

or as:

```
0 // Denotes Debug/Analysis mode
<G> // G is the undirected, weighted graph corresponding to L
<b> // b is the column vector in Lx = b
<epsilon> // epsilon is the error parameter for the algorithm's termination
<x> // x is the final solution to Lx = b, computed through standard solvers
```

Note the below protocol for the exact formats.

1.2 Data Format for PLW-SOLVE Entities

1.2.1 HashMaps

1.2.1.1 Hitting Table Index

```
<number of nodes>
<start node> <end node> <indexed node> <occurences> // 1st Entry
<start node> <end node> <indexed node> <occurences> // 2nd Entry
...
...
...
<start node> <end node> <indexed node> <occurences> // <number of nodes> ^ 3 th Entry
```

1.2.2 Graphs

1.2.2.1 Unweighted & Undirected

```
<number of nodes>
<number of edges>
<node_1> <node_2> //edge number 1
<node_1> <node_2> //edge number 2
...
...
...
<node_1> <node_2> //edge number <number of edges>
```

1.2.2.2 Weighted & Undirected

```
<number of nodes>
<number of edges>
<node_1> <node_2> <weight> //edge number 1
<node_1> <node_2> <weight> //edge number 2
...
...
...
<node_1> <node_2> <weight> //edge number <number of edges>
```

1.2.3 Matrices

```
<number of rows>
<number of columns>
<E> <E> <E> <E> ... <E> //space seperated row 1 of the matrix
<E> <E> <E> <E> ... <E> //space seperated row 2 of the matrix
...
...
...
<E> <E> <E> <E> ... <E> //space seperated row <number of rows> of the matrix
```

1.2.4 Vectors and Functions

1.2.4.1 Weight Vector

```
<number of edges> //number of edges
<node_1> <node_2> <weight> //edge number 1
<node_1> <node_2> <weight> //edge number 2
...
...
...
<node_1> <node_2> <weight> //edge number <number of edges>
```

1.2.4.2 Column Vector for system of linear equations (b)

```
<number of entries in vector> //number of vector entries
<element> // First element
<element> // Second element
...
...
...
<element> // <number of entries in vector>th element
```

1.2.5 Singleton Values

Error Parameter (**ϵ**)

```
<epsilon> //Value of error parameter
```

Number of samples(N)

```
<Number of samples> //Total number of samples
```

Chapter 2

File Index

2.1 File List

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

Chapter 3

File Documentation

3.1 src/naive/NaivePLW.cpp File Reference

One-sink Laplacian Solver Using Random Walks.

```
#include <bits/stdc++.h>
```

Macros

- #define **nll** ""
  empty string alias
- #define **br** " "
  space alias
- #define **nl** std::endl
  newline alias
- #define **in**(a) std::cin >> a;
  single input alias
- #define **in2**(a, b) std::cin >> a >> b;
  double input alias
- #define **in3**(a, b, c) std::cin >> a >> b >> c;
  triple input alias
- #define **out**(a) std::cout << a;
  single output alias
- #define **out2**(a, b) std::cout << a << br << b;
  double output alias
- #define **out3**(a, b, c) std::cout << a << br << b << br << c;
  triple output alias
- #define **outs**(a) out2(a,nll)
  output with space alias
- #define **outn**(a) std::cout << a << nl;
  output with newline alias

Functions

- template<typename T >
  void **incontainer** (std::vector< T > &v)
  Inputs space separated dynamic array.
- template<typename T >
  void **outcontainer** (std::vector< T > &v)
  Outputs space separated dynamic array.

- template<typename T >
  void **inmatrix** (std::vector< std::vector< T > > &v)

  Inputs space separated entries of newline separated dynamic arrays forming a matrix.
- template<typename T >
  void **outmatrix** (std::vector< std::vector< T > > &v)

  Outputs a matrix.
- void DFS (int node, std::vector< int > &visited, int &cnt)

  Performs DFS on the graph storing connected counter in cnt.
- bool checkConnected ()

  Check if the given graph is connected.
- template<typename T >
  bool **cumDist** ()

  Checks whether the.
- void throwError (std::string err)

  Throws Error and exits the program.
- template<typename T >
  T distSelector (const std::vector< std::pair< double, T > > &dist)

  Sample from a given distribution.
- void generateHittingTable (int start, int end)

  Generates Hitting Table between the two given vertices.
- void init ()

  Initializes the chain.
- void bootstrap ()

  Bootstraps the chain and finds s.
- void runChainSerial ()

  Run the chain serially.
- void runChain ()

  Run Phase two.
- void end ()

  Completion Formalities.
- template<typename T >
  bool cumDist (const std::vector< std::pair< double, T > > &dist)

  Checks if the given array is a Cumulative distribution or not.
- int main ()

  Main function.

Variables

- int **n**

  Number of Nodes in Graph.
- int **m**

  Number of Edges in Graph.
- int **s**

  Vertex chosen via bootstrapping indicating high stationary prob. state.
- int **u**

  The index of the sink vertex.
- int **timer** = 0

  Timer for running the chain serially.
- int **N** = -1

  Number of samples for bootstrapping.
- int **d** = 5

  Stores the nunber of chains to run.

- double **sb**

  Stores the sum of non-sink column vectors.
- double **eps** = 1

  Stores the bound on error required.
- std::mutex **io_lock**

  Declares variables for storing timestamps and durations if -DTIMER is passed as a flag.
- std::vector< int > **identity**
- std::vector< int > **X**

  State vector for multi-dimension markov chain.
- std::vector< int > **Q**

  Occupancy vector for each node.
- std::vector< double > **mu**

  Error vector mu as per definition.
- std::vector< double > **b**

  Column vector b as per definition.
- std::vector< double > **j**

  Column vector j as per definition.
- std::vector< double > **D**

  Stores total weight sum for the vertices.
- std::vector< std::pair< double, int > > **sources**

  Distribution of sources.
- std::vector< std::vector< int > > **adj_list**

  Adjacency list for the given graph.
- std::vector< std::tuple< int, int, double > > **edges**

  List of edges of the graph.
- std::vector< std::vector< std::pair< double, int > > > **P**
- std::vector< std::vector< std::pair< double, int > > > **Cum_P**

  Transition Matrix and Cumulative Transition Matrix.
- std::map< std::pair< int, int >, double > **weightMap**

  Contains mapping from pair of nodes to their corresponding edge weight.
- std::map< std::pair< int, int >, std::vector< std::pair< double, int > > > **HittingTable**

  Stores the computed hitting table distributions for different pair of nodes.

3.1.1 Detailed Description

One-sink Laplacian Solver Using Random Walks.

Author

Dhananjay Kajla (kajla.dhananjay@gmail.com)

Version

0.5

Date

2021-11-22

Copyright

3.1.2 Function Documentation

3.1.2.1 bootstrap()

```
void bootstrap ( )
```

Bootstraps the chain and finds s.
Bootstrap runs the bootstrapping algorithm and finds the ideal vertex to start the chain from. < Default Initialization of the chain
< Represents the vertices we picked
< Picking a source
< No paths between source sink pair
< Generate a path between source and sink
< Pick a vertex from the given distribution
< Increment counter for the chosen vertex
< Keeps check of maximum freq
< Keeps track of best vertex
< Update new maximum
< Update new winner
< Assign chosen vertex to s

3.1.2.2 checkConnected()

```
bool checkConnected ( )
```

Check if the given graph is connected.
Checks if the input is connected.

Returns

true Input Graph is connected

false Input Graph is not connected

< This vector keeps track of visited nodes
< Counter to keep track of number of visited nodes
< DFS to update visited counter
< Return true if all nodes have been visited, false otherwise

3.1.2.3 cumDist()

```
template<typename T >
bool cumDist (
            const std::vector< std::pair< double, T > > & dist )
```

Checks if the given array is a Cumulative distribution or not.

Template Parameters

T	Type of labels on each entity

Parameters

dist	Candidate distribution

Returns

true Given candidate is a Cumulative distribution

false Given candidate is not a Cumulative distribution

< checks previous entry
< cumulative prob. distribution is monotonically increasing
< update previous entry
< last entry of the distribution should be 1

3.1.2.4 DFS()

```
void DFS (
            int node,
            std::vector< int > & visited,
            int & cnt )
```

Performs DFS on the graph storing connected counter in cnt.
Runs a Depth first search to mark all connected nodes.

Parameters

node	current node
visited	array indicating wether each node has been visited or not
cnt	counter indicating number of visited nodes

< increase counter indicating total number of connected vertices
< color of node is color_val
< Iterate through the neighbors of current node
< Node already visited
< Node not visited, apply DFS recursively

3.1.2.5 distSelector()

```
template<typename T >
T distSelector (
            const std::vector< std::pair< double, T > > & dist )
```

Sample from a given distribution.
Selects a given label from the distribution.

Template Parameters

T	Type of label in the disrtibution

Parameters

dist	Cumulative probability disrtibution

Returns

T

< d1 represents the chosen random value
< first entry is the selected entry
< Temp variables for binary search
< Binary search for the interval of distribution in which d1 falls

3.1.2.6 end()

```
void end ( )
```

Completion Formalities.
Completes exit formalities. <std::cout << s << std::endl;

3.1.2.7 generateHittingTable()

```
void generateHittingTable (
            int start,
            int end )
```

Generates Hitting Table between the two given vertices.
Generates Hitting table for vertices starting from start and ending at end.

Parameters

start	Starting vertex
end	Ending vertex

< Initial vertex
< start has been visited
< Total number of vertices in the walk
< Continue walking until we hit our target
< Select next vertex from the transition matrix
< increment occurences of i1
< increment vertex counter for the walk
< Vector to generate Hitting table distribution
< Generate Hitting Table Distribution

3.1.2.8 init()

```
void init ( )
```

Initializes the chain.
Initializes the variables, i.e.
Takes input < Temporary Int Variables
< Temporary Double Variables
< Input Number of nodes and Number of edges
< Initialize edges vector
< Initialize adjacency list vector
< Initialize Transition Matrix
< Initialize Cumulative Transition Matrix
< Initialize Total node weight tracker
< Input the edges
< Takes in the incident vertices and their edge-weight
< Setup adjacency list
< Setup adjacency list
< Map given edge weight
< Map given edge weight
< Add weight of given edges to total node weight
< Add weight of given edges to total node weight
< append given edge to the edges
< Check if the graph is connected
< Setting up the transition matrix for our markov chain
< Cumulative Probability counter
< Check if cumulative edge weight of a vertex is very low or 0
< Throw Error and exit
< Scan through all neighboring vertices
< Get the weight of the required edge
< Find relative weightage of this edge
< Add it's probability to the cumulative value
< If there's a chance to go from i to it, we push it in our transition matrix
< Update transition matrix
< Update the cumulative transition matrix
< Input the dimension of column vector, this should equal n
< Initialize b
< Initialize j
< Taking input bi
< Finding the sink
< Sum of all non-sink vertices

< Indentified u
< Cumulative j
< Sink vertex
< j_sink = 0 by definition
< Using definition of j
< Updating cumulative value
< Making cumulative probability distribution
< Culumative j
< Initializing P[u]
< Initializing Cum_P[u]
< Push back non-zero j value into transition matrix
< Push back non-zero j value into transition matrix
< Build cumulative transition matrix for sink
< Input the error parameter
<std::cout << eps << std::endl;

3.1.2.9 main()

```
int main ( )
```

Main function.

Returns

int Exit status of the program

<std::ios_base::sync_with_stdio(false); std::cin.tie(NULL);

3.1.2.10 runChain()

```
void runChain ( )
```

Run Phase two.
runs the chain

3.1.2.11 runChainSerial()

```
void runChainSerial ( )
```

Run the chain serially.
Runs the chain serially. < t = t+1
< Choosing a dimension to update
< Check whether the update is lazy
< Select new vertex from distribution
< Increment occupancy for new vertex
< Decrement occupancy for previous vertex
< Increment in mu for previous vertex
< Increment in mu for new vertex
< Update mu[previous vertex]
< Update mu[new vertex]
< Update state of multi-dim markov chain

3.1.2.12 throwError()

```
void throwError (
            std::string err )
```

Throws Error and exits the program.
A utility function to throw errors and exit the program.

Parameters

err	Error encoded as string

Index