Network Simulator

Generated by Doxygen 1.15.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 AdjGraph Struct Reference

graph representation using an adjacency list.

Data Fields

- int32_t n_nodes
- int32_t n_edges
- EdgeNode ** head

3.1.1 Detailed Description

graph representation using an adjacency list.

3.1.2 Field Documentation

3.1.2.1 head

```
EdgeNode** AdjGraph::head
```

array of pointers to the head of the linked lists for each node

3.1.2.2 n_edges

```
int32_t AdjGraph::n_edges
```

num of edges in the graph

3.1.2.3 n_nodes

```
int32_t AdjGraph::n_nodes
```

num of nodes in the graph

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

- simulator.c

3.2 CSRGraph Struct Reference

graph representation using compressed sparse row/CSR.

Data Fields

- int32_t n_nodes
- int32_t n_edges
- int32_t * row_ptr
- int32_t * col_ind
- double * val
- double * val_unsigned

3.2.1 Detailed Description

graph representation using compressed sparse row/CSR.

3.2.2 Field Documentation

3.2.2.1 col_ind

```
int32_t* CSRGraph::col_ind
```

column indices (size n_edges)

3.2.2.2 n_edges

```
int32_t CSRGraph::n_edges
```

num of edges

3.2.2.3 n_nodes

```
int32_t CSRGraph::n_nodes
```

num of nodes

3.2.2.4 row_ptr

```
int32_t* CSRGraph::row_ptr
```

row pointers (size n_nodes + 1)

3.2.2.5 val

```
double* CSRGraph::val
```

edge weights (size n_edges)

3.2.2.6 val_unsigned

```
double* CSRGraph::val_unsigned
```

unweighted (size n_edges)

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

- simulator.c

3.3 EdgeNode Struct Reference

a node in the adjacency list linked list.

Data Fields

- int32_t dst
- double w
- double uw
- struct EdgeNode * next

3.3.1 Detailed Description

a node in the adjacency list linked list.

3.3.2 Field Documentation

3.3.2.1 dst

```
int32_t EdgeNode::dst
```

destination node id

3.3.2.2 next

```
struct EdgeNode* EdgeNode::next
```

pointer to the next edge in the list

3.3.2.3 uw

```
double EdgeNode::uw
```

unweighted value used for calculations

3.3.2.4 w

```
double EdgeNode::w
```

weighted value of the edge

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

- simulator.c

3.4 EdgeRaw Struct Reference

temporary structure for reading edges before graph construction.

Data Fields

- int src
- int dst
- double w
- double uw

3.4.1 Detailed Description

temporary structure for reading edges before graph construction.

3.4.2 Field Documentation

3.4.2.1 dst

```
int EdgeRaw::dst
```

3.4.2.2 src

```
int EdgeRaw::src
```

3.4.2.3 uw

```
double EdgeRaw::uw
```

3.4.2.4 w

```
double EdgeRaw::w
```

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

- simulator.c

3.5 HeapNode Struct Reference

represents a single element in the priority queue.

Data Fields

- int32_t node
- double dist

3.5.1 Detailed Description

represents a single element in the priority queue.

3.5.2 Field Documentation

3.5.2.1 dist

```
double HeapNode::dist
```

the priority value: distance/cost

3.5.2.2 node

```
int32_t HeapNode::node
```

node id

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

- simulator.c

3.6 MinHeap Struct Reference

a binary min-heap structure for efficient priority queue operations for Dijkstra's algorithm during global spillover calculation.

Data Fields

- HeapNode * data
- int32_t size
- int32_t capacity

3.6.1 Detailed Description

a binary min-heap structure for efficient priority queue operations for Dijkstra's algorithm during global spillover calculation.

3.6.2 Field Documentation

3.6.2.1 capacity

```
int32_t MinHeap::capacity
```

maximum capacity of the heap

3.6.2.2 data

```
HeapNode* MinHeap::data
```

array of heap nodes

3.6.2.3 size

```
int32_t MinHeap::size
```

current number of elements in the heap

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

- simulator.c

3.7 Node Struct Reference

represents a single entity in the simulation network.

Data Fields

- int32_t id
- double initial
- double target
- double indicator
- double alloc
- double gap
- double spill_weight

3.7.1 Detailed Description

represents a single entity in the simulation network.

3.7.2 Field Documentation

3.7.2.1 alloc

```
double Node::alloc
```

budget allocated to this node

3.7.2.2 gap

```
double Node::gap
```

difference between target and indicator

3.7.2.3 id

```
int32_t Node::id
```

unique node id

3.7.2.4 indicator

```
double Node::indicator
```

current state indicator

3.7.2.5 initial

```
double Node::initial
```

initial value

3.7.2.6 spill_weight

```
double Node::spill_weight
```

weight based on uniform, local, or global strategy

3.7.2.7 target

```
double Node::target
```

target value

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

- simulator.c

3.8 RankPair Struct Reference

helper structure for sorting nodes by spillover weight.

Data Fields

- int index
- double value

3.8.1 Detailed Description

helper structure for sorting nodes by spillover weight.

3.8.2 Field Documentation

3.8.2.1 index

```
int RankPair::index
```

original index of the node

3.8.2.2 value

```
double RankPair::value
```

value to sort by spillover weight

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

- simulator.c

Chapter 4

File Documentation

4.1 simulator.c File Reference

A simulator of network using OpenMP for parallel implementation.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
#include <omp.h>
```

Data Structures

- struct HeapNode

  represents a single element in the priority queue.
- struct MinHeap

  a binary min-heap structure for efficient priority queue operations for Dijkstra's algorithm during global spillover calculation.
- struct EdgeNode

  a node in the adjacency list linked list.
- struct AdjGraph

  graph representation using an adjacency list.
- struct CSRGraph

  graph representation using compressed sparse row/CSR.
- struct Node

  represents a single entity in the simulation network.
- struct RankPair

  helper structure for sorting nodes by spillover weight.
- struct EdgeRaw

  temporary structure for reading edges before graph construction.

Macros

- #define MAX_LINE_LEN 4096
- #define DIST_INF 1e300

Typedefs

- typedef struct EdgeNode EdgeNode

Functions

- MinHeap ∗ heap_create (int32_t capacity)

  allocates and initializes a new MinHeap.
- void heap_free (MinHeap ∗h)

  frees the memory associated with a MinHeap.
- void heap_push (MinHeap ∗h, int32_t node, double dist)

  pushes a new node into the min-heap.
- HeapNode heap_pop (MinHeap ∗h)

  pops the node with the smallest distance from the heap.
- AdjGraph ∗ create_adj_graph (int32_t n_nodes)

  creates an empty adjacency graph.
- void add_edge_adj (AdjGraph ∗g, int src, int dst, double w, double uw)

  adds an edge to the adjacency graph.
- void free_adj (AdjGraph ∗g)

  frees all memory associated with an adjacency graph.
- void free_csr (CSRGraph ∗g)

  frees all memory associated with a csr graph.
- void reset_nodes (Node ∗nodes, int n)

  resets node states for a new simulation run.
- int compare_rank (const void ∗a, const void ∗b)

  comparator function for qsort to sort RankPairs descending.
- int load_nodes (const char ∗path, Node ∗∗out_nodes, int ∗out_n)

  loads node data from a csv file.
- int load_graphs (const char ∗path, int n_nodes, CSRGraph ∗∗csr_f, CSRGraph ∗∗csr_b, AdjGraph ∗∗adj_f, AdjGraph ∗∗adj_b)

  loads graph data and builds both csr and adjacency list representations.
- void compute_local_spillover_csr (const CSRGraph ∗g, Node ∗nodes, int n)

  computes local spillover weights using csr representation.
- void compute_global_spillover_csr (const CSRGraph ∗g, Node ∗nodes, int n)

  computes global spillover weights using CSR representation.
- void compute_local_spillover_adj (const AdjGraph ∗g, Node ∗nodes, int n)

  computes local spillover weights using adj list representation.
- void compute_global_spillover_adj (const AdjGraph ∗g, Node ∗nodes, int n)

  computes global spillover weights using adj list representation.
- double compute_l2_gap (Node ∗nodes, int n)

  computes the l2/Euclidean dist of the gap.
- void simulation_iteration (int rep_type, const char ∗rep_name, const char ∗fac_name, const void ∗bwd_↩ graph, Node ∗nodes, int n, double budget, double lambda, double gamma, int max_iter, double epsilon, int output_mode, const char ∗output_dir, int data_val, int num_candidates, int num_snapshots, int ∗out_steps, double ∗out_l2)

  executes the simulation loop for budget allocation.
- int main (int argc, char ∗∗argv)

  entry point of the simulation.

4.1.1 Detailed Description

A simulator of network using OpenMP for parallel implementation.

The main funtion contains spillover computation and iteration computation. It supports two graph representations: adjacency list and compressed sparse row. It uses three allocation strategies: uniform, local spillover, and global spillover.

Note

- Compile: mpicc -fopenmp -fopenmp simulator.c -o simulator -lm
- Usage: srun -n #rank -c #thread ./simulator [gamma] [lambda] [budget] [output_mode] [nodes_file] [edges_file] ...

4.1.2 Macro Definition Documentation

4.1.2.1 DIST_INF

```
#define DIST_INF 1e300
```

representation of infinity for distance calculation

4.1.2.2 MAX_LINE_LEN

```
#define MAX_LINE_LEN 4096
```

maximum length for line buffer reading

4.1.3 Typedef Documentation

4.1.3.1 EdgeNode

```
typedef struct EdgeNode EdgeNode
```

4.1.4 Function Documentation

4.1.4.1 add_edge_adj()

```
void add_edge_adj (
            AdjGraph * g,
            int src,
            int dst,
            double w,
            double uw)
```

adds an edge to the adjacency graph.

Parameters

g	pointer to the graph.
src	source node id.
dst	destination node id.
w	edge weight.
uw	unweighted value.

4.1.4.2 compare_rank()

```
int compare_rank (
            const void * a,
            const void * b)
```

comparator function for qsort to sort RankPairs descending.

4.1.4.3 compute_global_spillover_adj()

```
void compute_global_spillover_adj (
            const AdjGraph * g,
            Node * nodes,
            int n)
```

computes global spillover weights using adj list representation.

Parameters

g	the adj graph.
nodes	the array of nodes to update.
n	num of nodes.

4.1.4.4 compute_global_spillover_csr()

```
void compute_global_spillover_csr (
            const CSRGraph * g,
            Node * nodes,
            int n)
```

computes global spillover weights using CSR representation.

Parameters

g	the csr graph.
nodes	the array of nodes to update.
n	num of nodes.

4.1.4.5 compute_l2_gap()

```
double compute_l2_gap (
            Node * nodes,
            int n)
```

computes the l2/Euclidean dist of the gap.

Parameters

nodes	array of nodes.
n	num of nodes.

Returns

the square root of the sum of squared gaps.

4.1.4.6 compute_local_spillover_adj()

```
void compute_local_spillover_adj (
            const AdjGraph * g,
            Node * nodes,
            int n)
```

computes local spillover weights using adj list representation.

Parameters

g	the adjacency graph.
nodes	the array of nodes to update.
n	num of nodes.

4.1.4.7 compute_local_spillover_csr()

```
void compute_local_spillover_csr (
            const CSRGraph * g,
            Node * nodes,
            int n)
```

computes local spillover weights using csr representation.

Parameters

g	the csr graph.
nodes	the array of nodes to update.
n	num of nodes.

4.1.4.8 create_adj_graph()

```
AdjGraph * create_adj_graph (
            int32_t n_nodes)
```

creates an empty adjacency graph.

Parameters

n_nodes	number of nodes.

Returns

pointer to the new AdjGraph.

4.1.4.9 free_adj()

```
void free_adj (
            AdjGraph * g)
```

frees all memory associated with an adjacency graph.

Parameters

g	pointer to the graph.

4.1.4.10 free_csr()

```
void free_csr (
            CSRGraph * g)
```

frees all memory associated with a csr graph.

Parameters

g	pointer to the graph.

4.1.4.11 heap_create()

```
MinHeap * heap_create (
            int32_t capacity)
```

allocates and initializes a new MinHeap.

Parameters

capacity	The maximum number of elements the heap can hold.

Returns

pointer to the created MinHeap.

4.1.4.12 heap_free()

```
void heap_free (
            MinHeap * h)
```

frees the memory associated with a MinHeap.

Parameters

h	pointer to the heap to free.

4.1.4.13 heap_pop()

```
HeapNode heap_pop (
            MinHeap * h)
```

pops the node with the smallest distance from the heap.

Parameters

h	pointer to the heap.

Returns

the HeapNode with the minimum distance.

4.1.4.14 heap_push()

```
void heap_push (
            MinHeap * h,
            int32_t node,
            double dist)
```

pushes a new node into the min-heap.

Parameters

h	pointer to the heap.
node	the node identifier.
dist	the priority/distance value.

4.1.4.15 load_graphs()

```
int load_graphs (
            const char * path,
            int n_nodes,
            CSRGraph ** csr_f,
            CSRGraph ** csr_b,
            AdjGraph ** adj_f,
            AdjGraph ** adj_b)
```

loads graph data and builds both csr and adjacency list representations.

Parameters

	path	path to the edges csv file.
	n_nodes	total num of nodes.
out	*csr_f*	pointer to forward csv graph.
out	*csr_b*	pointer to backward csv graph.
out	*adj_f*	pointer to forward adjacency graph.
out	*adj_b*	pointer to backward adjacency graph.

Returns

0 on success, -1 on failure.

4.1.4.16 load_nodes()

```
int load_nodes (
            const char * path,
            Node ** out_nodes,
            int * out_n)
```

loads node data from a csv file.

Parameters

	path	path to the csv file.
out	*out_nodes*	pointer to the array of nodes.
out	*out_n*	pointer to the integer storing node count.

Returns

0 on success, -1 on failure.

4.1.4.17 main()

```
int main (
            int argc,
            char ** argv)
```

entry point of the simulation.

Parameters

argc	argument count.
argv	argument vector.

Returns

0 on success, 1 on error.

4.1.4.18 reset_nodes()

```
void reset_nodes (
            Node * nodes,
            int n)
```

resets node states for a new simulation run.

Parameters

nodes	array of nodes.
n	num of nodes.

4.1.4.19 simulation_iteration()

```
void simulation_iteration (
            int rep_type,
            const char * rep_name,
            const char * fac_name,
            const void * bwd_graph,
            Node * nodes,
            int n,
            double budget,
            double lambda,
            double gamma,
            int max_iter,
            double epsilon,
            int output_mode,
            const char * output_dir,
            int data_val,
            int num_candidates,
            int num_snapshots,
            int * out_steps,
            double * out_l2)
```

executes the simulation loop for budget allocation.

Parameters

	rep_type	the graph representation (0 for AdjList, 1 for CSR).
	rep_name	string name of representation for logging.
	fac_name	string name of the spillover factor strategy.
	bwd_graph	void pointer to the backward graph (cast based on rep_type).
	nodes	array of Node structures containing state.
	n	total number of nodes.
	budget	total budget to distribute per iteration.
	lambda	impact factor for spillover weights.
	gamma	step size for indicator updates.
	max_iter	max num of iterations allowed.
	epsilon	convergence threshold or tolerance for l2 gap.
	output_mode	if 1, writes detailed csv logs; if 0, runs silently.
	output_dir	directory path for output files.
	data_val	the dataset scale identifier.
	num_candidates	num of top candidates to track in logs.
	num_snapshots	num of snapshots to record in the log file.
out	*out_steps*	pointer to store the final iteration count.
out	*out_l2*	pointer to store the final l2 error.