Appendix A

Quantum Walk Similarity Algorithm Documentation

The coined and Szegedy quantum walk algorithms have been developed in *Fortran*. The functionality of each algorithm is split amongst a number of module files, and two program files. The first program takes a list containing pairs of graphs to determine the similarity between. The second program takes a list of graphs and outputs the similarity between the input graph and itself with a number of random changes made. The source code files can be found at github.com/C-Schofield/Quantum-Walk-Graph-Similarity.

A.1 Coined Quantum Walk Algorithm

A.1.1 Data_Types.f90

Source file used to define parameters and the structure (type) *node* used to define the space and coin basis states of the system $|\psi\rangle$. Each node is defined by its own copy of *node*.

A.1.2 Walk_Functions.f90

Collection of subroutines which provide the coined quantum walk implementation. These subroutines initiate and conduct the walk, some of which are:

- INIT_STATE: Initialises the state $|\psi\rangle$ in equal superposition.
- COPY_STATE: Copies the contents from one state to another, used to preallocate a state used during the flip operation.

- CLEAR_STATE: Used to deallocate the contents of each *node* structure in a state.
- FlipCoin: Applies the coin flip operation.
- Step: Conducts a single step in the coined quantum walk.
- ProbDist: Returns the probability that the walker is at each node for the given state.

A.1.3 QW_Similarity.f90

Provides subroutines which determine the similarity score between the two input graphs. There are two subroutines which compute the comparison score between two graphs, using a single and two reference node scheme respectively.

A.1.4 QW_Multiple.f90

This program provides an interface between the module files and the user. The input and parameter files are parsed, then the similarity between the input graphs are determined and outputted.

A.1.5 QW_Changes.f90

This program provides the resulting similarity score between each input graph and the set of graphs which are produced by making random changes.

A.2 Szegedy Quantum Walk Algorithm

A.2.1 Szegedy_Functions.f90

Collection of subroutines which provide the Szegedy quantum walk implementation. These subroutines initiate and conduct the walk, some of which are:

- Szeg_init: Initialises the state $|\psi\rangle$ in equal superposition and calculates the quantum step operator $\hat{U}$.
- colNormalise: Ensures the input adjacency (or transition) matrix is column normalised.

- SzegStep: Conducts a single step in the Szegedy quantum walk.
- MapProb: Returns the probability that the walker is at each node for the given state.

A.2.2 Szegedy_Similarity.f90

Provides subroutines which determine the similarity score between the two input graphs. There are two subroutines which compute the comparison score between two graphs, one which makes comparisons between the two graphs, and the other makes comparisons for the input graph to a predetermined set of probabilities (used in Szegedy_Changes so that the same operations are not repeated each iteration).

A.2.3 Szegedy_Multiple.f90

This program provides an interface between the module files and the user. The input and parameter files are parsed, then the similarity between the input graphs are determined and outputted.

A.2.4 Szegedy_Changes.f90

This program provides the resulting similarity score between each input graph and the set of graphs which are produced by making random changes.

A.3 Common Functionality

Below lists the source files which are utilised in both the coined and Szegedy quantum walk algorithms.

A.3.1 Common_Functions.f90

Contains miscellaneous functions which are not specific to either program but provide provide additional functionality. These include:

- identity: Returns the identity matrix of size n.
- init_random_seed: Creates a random seed using the system clock for the *random_number* function.

- MAT_INV: Returns the inverse of the input matrix.
- FILE_DIMENSIONS: Returns the number of lines of the input file.
- MATUSITA_DIST: Determines the Matusita distance between two vectors.
- PERMUTATION: Returns a random permutation vector of length n.

A.3.2 Graph_Functions.f90

Contains functions and subroutines which provide information or make modifications on adjacency (or transition) matrices. Each is interfaced to allow graphs defined using both integer and double types. Additionally, the functionality is separated into graphs which are undirected or directed, specified via an input variable.

- CHANGE_GRAPH: Makes *nChanges* random modifications to the input graph based on the input mode (1,2,3,4).
    1. *Add random edge*: Adds an edge to the graph randomly.
    2. *Remove random edge*: Removes an edge from the graph randomly.
    3. *Add or remove random edge*: Each change has a 50/50 chance to either add or remove an edge from the graph randomly.
    4. *Randomly modify weight*: An edge is chosen at random to be given a random weight. Note this is only available for graphs with type double.
- COUNT_EDGES: Returns the number of edges in the input graph. For directed graphs, edges entering and leaving nodes are counted as separate edges.

A.3.3 NodeAffinity_Functions.f90

Contains functions and subroutines which compute the similarity between graphs using the DELTACON method [20]. Functions are interfaced to allow integer or double graphs to be compared.

A.3.4 Sort.f90

Fortran quicksort code written by Michel Olagnon.

A.4 Usage

Each program has a corresponding parameters file which can be loaded to define various program variables. Additionally, some variables can be set using the command line (indicated below with their options via "-option_name"). Parameter files use "#" as a comment line. Note that loading the parameter file (using -p or -param on the command line) will override any options prior to it in the command line. The parameters are to be listed in the following orders:

A.4.1 QW_Multiple.f90

- -g, -graphs: List of graph locations. Each line contains the filepath of each graph to be compared, separated by "¡¿".
- -o, -out: Output file location.
- Number of steps for the walker to take.
- Number of reference nodes to use (can only currently use 0,1,2).
- -t, -threads: Number of threads to use for parallelisation.
- Value for ϵ (used for threshold).

A.4.2 QW_Changes.f90

- -g, -graphs: List of graph locations. Each line contains the filepath a single graph to be compared (to itself with modifications).
- -o, -out: Output file location.
- Number of steps for the walker to take.
- Number of reference nodes to use (can only currently use 0,1,2).
- -t, -threads: Number of threads to use for parallelisation.
- -b, -begin: Initial number of changes to make.
- -e, -end: Final number of changes to make.
- -s, -step: Step-size between each set of changes.
- -n, -trials: Number of trials to make for each set of changes.
- -m, -mode: Type of changes to be made (1 add only, 2 remove only,3 both)

- Value for ϵ (used for threshold).

A.4.3 Szegedy_Multiple.f90

- -g, -graphs: List of graph locations. Each line contains the filepath of each graph to be compared, separated by "<>".
- -o, -out: Output file location.
- Number of steps for the walker to take.
- -t, -threads: Number of threads to use for parallelisation.
- -w, -weight: Value for w (used for scaling). Note that if you want the algorithm to return only the distance, so you can determine the similarity using $sim = \frac{1}{1+wd}$ post completion, using a variety of w values, set $w \leq 0$.

A.4.4 Szegedy_Changes.f90

- -g, -graphs: List of graph locations. Each line contains the filepath a single graph to be compared (to itself with modifications).
- -o, -out: Output file location.
- Number of steps for the walker to take.
- -t, -threads: Number of threads to use for parallelisation.
- -b, -begin: Initial number of changes to make.
- -e, -end: Final number of changes to make.
- -s, -step: Step-size between each set of changes.
- -n, -trials: Number of trials to make for each set of changes.
- -m, -mode: Type of changes to be made (1 add only, 2 remove only,3 both)
- -d, -directed: Determines whether to treat edges as directed or undirected when making changes
- -w, -weight: Value for w (used for scaling). Note that if you want the algorithm to return only the distance, so you can determine the similarity using $sim = \frac{1}{1+wd}$ post completion, using a variety of w values, set $w \leq 0$.