Porównanie bibliotek Jung oraz Gephi

JUNG:

Interfaces:

- DirectedGraph<V,E>
- Graph<V,E>
- Hypergraph<V,E>
- MultiGraph<V,E>
- UndirectedGraph<V,E>
- Tree<V,E>

Classes:

- AbstractGraph<V,E>
- AbstractTypedGraph<V,E>
- DelegateTree<V,E>
- SparseGraph<V,E>
- SparseMultigraph<V,E>
- UndirectedSparseGraph<V,E>
- UndirectedSparseMultigraph<V,E>
- ObservableGraph<V,E>

Features:

- *Structurallly equivalent vertices - sets of structurally equivalent vertices in a graph (.algorithms.blockmodel.StructurallyEquivalent<V,E>)
    - Set<Pair<V>> getEquivalentPairs(Graph<V,?> g) – For each vertex pair v, v1 in G, checks whether v and v1 are fully equivalent: meaning that they connect to the exact same vertices.
- *Weak components - all weak components in a graph as sets of vertex sets. A weak component is defined as a maximal subgraph in which all pairs of vertices in the subgraph are reachable from one another in the underlying undirected subgraph. (.algorithms.cluster.WeakComponentClusterer<V,E>)
    - Set<Set<V>> transform(Graph<V,E> graph) – Extracts the weak components from a graph.
- *Edge betweenness clusters - computing clusters (community structure) in graphs based on edge betweenness. (algorithms.cluster.EdgeBetweennessClusterer<V,E>)
    - List<E> getEdgesRemoved() – Retrieves the list of all edges that were removed (assuming extract(...) was previously called).
- *Bicomponents - Finds all biconnected components (bicomponents) of an undirected graph. A graph is a biconnected component if at least 2 vertices must be removed in order to disconnect the graph.(algorithms.cluster.BicomponentClusterer<V,E>)
    - protected void findBiconnectedComponents(UndirectedGraph<V,E> g, V v, Set<Set<V>> bicomponents) – Stores, in bicomponents, all the biconnected components that are reachable from v.

- *(Edge) betweenness centrality - Computes betweenness centrality for each vertex and edge in the graph. (.algorithms.importance.BetweennessCentrality<V,E> oraz .algorithms.scoring.BetweennessCentrality<V,E>, jakieś inne: .algorithms.importance.RandomWalkBetweenness<V,E>, algorithms.importance.RandomWalkSTBetweenness<V,E>)
- *K-step Markov (?) - Algorithm variant of PageRankWithPriors that computes the importance of a node based upon taking fixed-length random walks out from the root set and then computing the stationary probability of being at each node. (.algorithms.scoring.KStepMarkov<V,E>, wersja algorithms.importance.KStepMarkov<V,E> podobno jest zła!)
- *Weighten NI Paths (?) - the importance of nodes based upon both the number and length of disjoint paths that lead to a given node from each of the nodes in the root set (algorithms.importance.WeightedNIPaths<V,E>)
    - protected void computeWeightedPathsFromSource(V root, int depth)
- *barycenter (zw. z sumą odległości dróg od wierzchołka do wszystkich pozostałych) - .algorithms.scoring.BarycenterScorer<V,E>
- *closeness centrality - mean distance to each other vertex. (algorithms.scoring.ClosenessCentrality<V,E>)
- *degree - .algorithms.scoring.DegreeScorer<V>
    - Integer getVertexScore(V v) – Returns the degree of the vertex.
- *distance centrality - scores to vertices based on their distances to each other vertex in the graph. This class optionally normalizes its results based on the value of its 'averaging' constructor parameter. If it is true, then the value returned for vertex v is 1 / (_average_ distance from v to all other vertices); this is sometimes called closeness centrality. If it is false, then the value returned is 1 / (_total_ distance from v to all other vertices); this is sometimes referred to as barycenter centrality. (.algorithms.scoring.DistanceCentralityScorer<V,E>)
    - Double getVertexScore(V v) – Calculates the score for the specified vertex.
- *eigenvector centrality - eigenvector centrality for each vertex in the graph. Assumes that the graph is strongly connected. (.algorithms.scoring.EigenvectorCentrality<V,E>)
- *pagerank - Assigns scores to each vertex according to the PageRank algorithm. (algorithms.scoring.PageRank<V,E>, algorithms.scoring.PageRankWithPriors<V,E>)
    - protected void collectDisappearingPotential(V v) – Collects the "disappearing potential" associated with vertices that have no outgoing edges.
- *HITS - Assigns hub and authority scores to each vertex depending on the topology of the network. (.algorithms.scoring.HITS<V,E> albo algorithms.scoring.HITSWithPriors<V,E>)
    - protected void collectDisappearingPotential(V v) – Collects the "disappearing potential" associated with vertices that have either no incoming edges, no outgoing edges, or both.
    - protected void normalizeScores() – Normalizes scores so that sum of their squares = 1.

GEPHI:

Interfaces:

- DirectedGraph
- GraphListener
- HierarchicalDirectedGraph
- MixedGraph
- UndirectedGraph

Classes:

- DirectedGraph
- UndirectedGraph

Algorithm:

- GraphDensity
  - double calculateDensity(org.gephi.graph.api.Graph graph, boolean isGraphDirected)
- GraphDistance
  - Map<String, Double[]> calculateDistanceMetrics(org.gephi.graph.api.Graph hgraph,HashMap<org.gephi.graph.api.Node,Integer> indicies, boolean directed, boolean normalized)
  - HashMap<org.gephi.graph.api.Node,Integer> createIndicesMap(org.gephi.graph.api.Graph hgraph)
- ClusteringCoefficient – the clustering coefficient (Watts-Strogatz), when applied to a single node, is a measure of how complete the neighborhood of a node is. When applied to an entire network, it is the average clustering coefficient over all of the nodes in the network.
  - HashMap<String, Double> computeClusteringCoefficient(org.gephi.graph.api.Graph hgraph, org.gephi.statistics.plugin.ArrayWrapper[] currentNetwork, int[] currentTriangles, double[] currentNodeClustering, boolean directed)
  - int createIndiciesMapAndInitNetwork(org.gephi.graph.api.Graph hgraph,HashMap<org.gephi.graph.api.Node,Integer> indicies, org.gephi.statistics.plugin.ArrayWrapper[] networks, int currentProgress)
- ConnectedComponents – On directed graphs: detect strongly and weakly connected components. On undirected graphs: detect only weakly connected components.
  - LinkedList<LinkedList<org.gephi.graph.api.Node>> computeWeeklyConnectedComponents(org.gephi.graph.api.Graph graph,HashMap<org.gephi.graph.api.Node,Integer> indicies)
  - HashMap<org.gephi.graph.api.Node,Integer> createIndiciesMap(org.gephi.graph.api.Graph hgraph)
  - LinkedList<LinkedList<org.gephi.graph.api.Node>> top_tarjans(org.gephi.graph.api.DirectedGraph graph,HashMap<org.gephi.graph.api.Node,Integer> indicies)
- Degree

- EigenvectorCentrality – A measure of node importance in a network based on a node's connections
  - double calculateEigenvectorCentrality(org.gephi.graph.api.Graph hgraph, double[] eigCentralities,HashMap<Integer,org.gephi.graph.api.Node> indicies,HashMap<org.gephi.graph.api.Node,Integer> invIndicies, boolean directed, int numIterations)
  - void fillIndiciesMaps(org.gephi.graph.api.Graph hgraph, double[] eigCentralities,HashMap<Integer,org.gephi.graph.api.Node> indicies,HashMap<org.gephi.graph.api.Node,Integer> invIndicies)
- Hits – Hyperlink-Induced Topic Search (HITS) (also known as Hubs and authorities) is a link analysis algorithm that rates Web pages, developed by Jon Kleinberg.
  - void calculateHits(org.gephi.graph.api.Graph hgraph, double[] hubValues, double[] authorityValues,Map<org.gephi.graph.api.Node,Integer> indicies, boolean isDirected, double eps)
  - HashMap<org.gephi.graph.api.Node,Integer> createIndiciesMap(org.gephi.graph.api.Graph hgraph)
- Modularity – Measures how well a network decomposes into modular communities.
- PageRank – An iterative algorithm that measures the importance of each node within the network. The metric assigns each node a probability that is the probability of being at that page after many clicks. The page rank values are the values in the eigenvector that has the highest corresponding eigenvalue of a normalized adjacency matrix A'. The standard adjacency matrix is normalized so that the columns of the matrix sum to 1
  - HashMap<org.gephi.graph.api.Node,Integer>createIndiciesMap(org.gephi.graph.api.Graph hgraph)
- WeightedDegree
  - double calculateAverageWeightedDegree(org.gephi.graph.api.Graph graph, boolean isDirected, boolean updateAttributes)