Asia Pacific Journal of Multidisciplinary Research
P-ISSN 2350-7756 | E-ISSN 2350-8442 | www.apjmr.com | Volume 2, No. 4, August 2014

A Hybrid Graph-drawing Algorithm for Large, Naturally-clustered, Disconnected Graphs

Toni-Jan Keith P. Monserrat,[1,2] Jaderick P. Pabico,[1] and Eliezer A. Albacea[1,3]

[1]*Institute of Computer Science, University of the Philippines Los Baños, Philippines*
[2]*NUS-HCI Laboratory, National University of Singapore, Singapore*
[3]*National Academy of Science and Technology, DOST, Philippines*

{tjkpmonserrat, jppabico,eaalbacea}@uplb.edu.ph

Date Received: July 7, 2014; Date Published: August 15, 2014

Abstract *– In this paper, we present a hybrid graph-drawing algorithm (GDA) for laying out large, naturally-clustered, disconnected graphs. We call it a hybrid algorithm because it is an implementation of a series of already known graph-drawing and graph-theoretic procedures. We remedy in this hybrid the problematic nature of the current force-based GDA which has the inability to scale to large, naturally-clustered, and disconnected graphs. These kinds of graphs usually model the complex inter-relationships among entities in social, biological, natural, and artificial networks. Obviously, the hybrid runs longer than the current GDAs. By using two extreme cases of graphs as inputs, we present the derivation of the time complexity of the hybrid which we found to be* $O(|V|^3)$*, where V is the set of nodes in the graph.*

Keywords **–** graph drawing, hybrid algorithm, large disconnected graph, clustered graph

I. INTRODUCTION

Information that abstractly describes the interrelationships among entities in most complex systems is usually mathematically represented using graphs. Graphs, as tools, are an intuitive approach for visualizing entities because they make it easier for humans to understand the relationships between different entities. Because of this, graph visualizations of entities, as well as that of processed data, are used in many types of applications. For example, computer science concepts are usually easier to understand with the use of visualization concepts such as data flow diagrams, subroutine-call graphs, program nesting trees, object-oriented class hierarchies, entity-relationship diagrams, organization charts, circuit schematics, knowledge- representation diagrams, logic trees, and networks. Other fields of sciences also use graph visualization to represent information like concept lattices, evolutionary trees, molecular drawings, and maps and map schematics [1].

Because of the utility of graph visualization for viewing data that can be understood by the user in a vast number of applications, many techniques were devised for drawing graphs efficiently and beautifully. Since the first paper by Knuth in 1963 on drawing flowcharts for visualization purposes [1, 2], there are now about 300 existing algorithms on graph drawing itself, some of these have improved the existing ones by utilizing the research advances made in topological and geometrical graph theory, graph algorithms, data structures, computational geometry, visual languages, graphical user interfaces, and software visualization [1]. However, given the numerous available algorithms, there is no one-size-fits-all graph drawing algorithm for any given graph. It is also important to identify the class to which a certain graph belongs. This is because several graph-drawing algorithms can only make effective visualizations on certain graph classes. Additionally, there are several approaches that exist in drawing graphs. Some of these approaches are drawing conventions, aesthetics, constraints, and efficiency. These approaches include topology-shape-metrics, hierarchical, visibility, augmentation, divide and conquer, and force-directed.

In the current effort, we developed a hybrid force-directed approach algorithm based on Kamada and Kawai's work [3]. We used the Markov clustering algorithm to group the original vertices into sub-graphs and then used the original Kamada-Kawai (KK) force-directed algorithm to draw the vertices in each sub-graph. We considered each sub-graph as a big "*phantom*" vertex and applied the Iterative Kamada-Kawai (IKK) algorithm to draw the respective locations of the non-uniform-sized phantom vertices.

In this paper, we analyze the runtime of our hybrid graph drawing algorithm (HGDA). We illustrate our derivation by considering input graphs in extreme cases: a fully connected graph $G_a(V, E_a)$ and a graph with no edges $G_b(V, \emptyset)$.. With these input graphs, we found out that HGDA has $O(|V|^3)$ runtime complexity.

II. REVIEW

Recent research worked around visualization of graphs to be more aesthetic, more understandable, and more pleasing to the viewer. Certain criteria were used to meet these needs. By including the type and properties of the graph to be drawn, several graph drawing algorithms emerged to cater to certain types of graphs. It is essential to know that no optimum drawing for any graph can be done, as human perception of aesthetic and ability to digest a visual image changes from every individual. It should be noted that although the product of a graph-drawing algorithm may be subjective, it also has objective criteria such as drawing convention, aesthetic, and constraints.

For a graph drawing to be admissible, it should follow certain drawing conventions such as: having polyline for edges, using planar mathematics for laying out these edges, and using grids to locate the vertices. A certain type of convention that is often used in graph drawing theories [1] is the straight-line drawing. To objectively evaluate the aesthetics of a graph drawing, it specifies graphic properties of drawing that adds readability at the least. Some common aesthetic evaluation includes minimization of the total number of edge crossings and minimization of the drawing area. These two efficiently use the drawing space without sacrificing the readability of the relationship between vertices [4–6]. Additionally, constraints must also be considered, specifically when drawing sub-graphs. Creating certain constraints on position and space provides how each subgraph should be drawn. An example of a common constraint would have a given vertex be drawn at the center of the drawing area. Another example is to have some of vertices be clustered or enclosed within a predefined shape [7, 8].

Because of these criteria, several approaches in graph drawing were established. One of these approaches is through the use of force-directed algorithms (FDA). Due to their flexibility, ease of implementation and often-pleasant drawings, FDA are often used and improved [9]. Conventionally, FDA use straight-line drawings to draw edges in undirected graphs. FDA simulate some "force" that is directed to each vertex. When the minimal energy of the whole system is already achieved, the position of the vertices in the graph are said to be in its balanced state. To find the balanced state of the graph, FDA incorporate two main functions: (1) The force model that simulates the forces acting on each of the vertex; and (2) An iterative algorithm to find the local minimal energy configuration [1].

The KK algorithm takes in a connected graph $G(V, E)$ and uses the graph theoretic distance (GTD) between each pair of vertices $u \in V$ and $v \in V$ as its force model. GTD between vertices u and v is calculated as the number of edges on a shortest path from vertex u to vertex v. The aim of the FDA that uses GTD as a force model is to find the Euclidean distance between u and v to be approximately proportional to their GTD. KK includes an energy or spring view in the GTD [1, 3]. Because of this, KK was able to create symmetric drawings with relatively few edge crossings, which is practically similar to drawing isomorphic graphs [3]. It should be noted, however, that KK only focused on fairly simple graphs. Originally, it was intended to solve undirected, non-weighted, simple, and fully connected graphs [10]. An obvious problem for KK is its inability to scale to handle large graphs. This inability is common also for other FDA. FADE [9], a fast algorithm for two-dimensional drawing of large undirected graphs, was one of the more successful implementations of FDA that scale to larger graphs. It uses clustering before applying FDA, although primarily to lessen the computational time, and secondarily for maintaining the visualization better [9].

There are many ways to cluster large graphs into manageable sub-graphs. Examples of these are the graph theoretic clustering [11] and the geometric clustering [12] procedures like the ones being used in FADE, and the Markov Cluster Algorithm (MCL) [13]. One of the advantages of MCL is that it does not have any high level procedural rules for splitting or joining groups. The idea of MCL is to simulate a system of "*current*" C flowing inside the graph, promote that system when C is strong, or demote the system when C is weak. The computational paradigm is that C between natural groups in the graph will wither away, revealing the cluster or sub-graph [13].

Clustering a graph into sub-graphs defines the structure and natural clusters within the graph. By doing so, it arranges the vertices in the adjacency matrix A by creating blocks of "1s" diagonally in A where the clusters are formed. This makes it easy for the FDA to find the equilibrium by re-ordering the vertices according to their connections within and between the

P-ISSN 2350-7756 | E-ISSN 2350-8442 | www.apjmr.com

clusters, as opposed to the original procedure of randomly arranging vertices in G [14].

III. THEORETICAL FRAMEWORK

Before discussing our hybrid algorithm, we start off with the framework that would help us discuss our derivation of our analysis.

A. Preliminary

The definition of a graph G is composed of a pair of sets (V, E), where V is the set of vertices and E is the set of edges in the graph. An edge (u,v) connects two vertices $u \epsilon V$ and $v \epsilon V$. The number of vertices $n=|V|$ is called the *order* of the graph while the cardinality $|E|$ of the edge set is called the *size* of the graph. In an undirected graph, each edge is an unordered pair (v,w). A vertex w is adjacent to a vertex v if and only if $(v,w) \epsilon E$. In an undirected graph, the abstract relationship represented by (v,w) is the same as that of (w, v).

A path in a graph is a sequence of vertices $w_1, w_2, ..., w_n$ such that there exists an edge (w_i, w_{i+1}) where $1 \leq i < n$. The length of the path is equal to number of edges $(n - 1)$, where n is the number of vertices that runs along that selected path. A simple path is a path such that all vertices are distinct. A cycle is a path with a distinction that the first and last vertex are the same [15]. A graph $G'(V', E')$ is a sub-graph of $G(V, E)$ if $V' \subset V$ and $E' \subset E \bigcap (V' \times V)$.

A graph $G(V, E)$ with $n = |V|$ vertices can be described by an $n \times n$ adjacency matrix A whose rows and columns correspond to vertices. The matrix elements $A_{u,v} = 1$ if (u, v) is part of E. $A_{u,v} = 0$ otherwise. A graph is connected if there is a path between u and v for each pair of vertices u and v.

B. Clustered and disconnected graphs

Graphs that are of small-world, scale-free characteristics are naturally clustered with some disconnected components. Small-world graphs are characterized by a very small network diameter, which usually values within six for naturally-occurring social networks SN [16, 17]. The degree Δ_i of a vertex v_i counts the number of incident edges of v_i. A symmetric matrix $A_{i,j}$ represents an undirected graph G, where $A_{i,j} = A_{j,i} = 1$ if v_i is incident to v_j. Thus, $\Delta i = \sum_{j=1..n} A_{i,j}$. For most SN , the frequency distribution $\rho(\Delta)$ of the degree in G has been found by various researchers [18–20] to asymptotically follow the power law distribution of the form $\rho(\Delta) = \alpha \times \Delta^{\varphi}$. For social networks, and all other biological networks, the power usually takes the value $-3 \leq \varphi \leq -2$. Having $\rho(\Delta) \sim \alpha \times \Delta^{\varphi}$ makes SN scale-free [18]. Figure 1 shows an example of a small-world, scale-free graph that is naturally clustered and disconnected.

C. Connected components

The **connected components** of an undirected graph G are the maximal disjoint sets $V_1, V_2, ..., V_n$ such that $V = V_1 \cup V_2 \cup \cdots \cup V_n$, and the vertices $u, v \in V_i$ if and only if u is reachable from v and v is reachable from u [22, 23]. Two methods are generally used to identify the connected components of G: (1) The breadth-first-search (BFS) and (2) The depth-first-search (DFS). We can use any of these two to see if a certain path from u to v exists for each vertex pair of (u, v) [24]. Given a starting vertex v_0, BFS systematically searches a given graph of vertices that has a path from v_0. First, BFS lists all vertices that are adjacent to v_0. Then, it starts again with another vertex v_i in the list that is directly connected with the previous vertex. The usual convention is to take the first vertex in the list as the v_i. BFS again does the listing of vertices that are directly connected to v_i. The algorithm stops when there are no more vertices that have a path from v_0. Now, if there still exist vertices that are not listed after the BFS has been done, then the said graph is considered disconnected. The complexity of a BFS algorithm that returns all connected components is $0(|V| \times |E|)$.

In DFS, the traversal is done in a depth-first fashion, wherein the outcome is a forest of depth-first trees. Each tree in the forest contains vertices that belong to a different subgraph. The correctness of DFS as a test for graph connectivity follows directly from the definition of a spanning tree, and from the fact that the graph is undirected. This means that a depth-first tree is also a spanning tree of a graph induced by the set of vertices in the depth-first tree. Assuming that the graph is stored using a sparse representation, the run time of the DFS is $\theta(|E|)$.

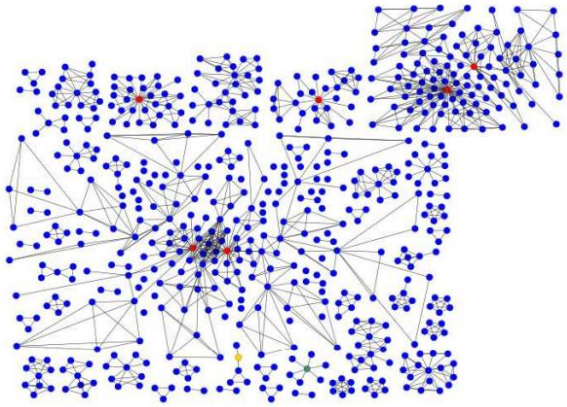

Figure 1. An example large, naturally-clustered and disconnected graph G drawn using KK. This graph is based on the co-authorship network of Filipino computer scientists created by Pabico [21] with $|V| =$

605. Notice that there exist some pronounced vertex

D. MCL

The MCL starts from a random starting vertex $v_0 \in G$ and walks to other vertices connected to v_0. Here, G maybe described using a similarity matrix. The traversal usually does not leave the graph's cluster until many of the cluster's vertices have been visited. The idea of the algorithm is that it simulates "*flow*" within a graph. The flow is done iteratively wherein after each step, MCL demotes the edges within the distant nodes and promotes the edges of the nearby nodes. To do this, MCL takes the corresponding $n \times n$ adjacency matrix A of the graph G and normalizes each column to obtain a stochastic matrix M. This includes adding the diagonal elements in the adjacency matrix to include self-loops for all nodes. After initializing the matrix, the algorithm uses two alternating functions: (1) **expansion**, which is used to flatten the stochastic distributions in the columns and causes the edges and paths of the random walker to become evenly spread; and (2) **inflation**, which contracts them to favor paths. It is said that the MCL algorithm's complexity is $O(n^3)$, where $n = |V|$ is the number of vertices of the input graph. This is the same as the cost of multiplying two matrices of dimension n. It is also noted that the **inflation** step of the algorithm has a complexity of $O(n^2)$. The mathematical analysis on the time complexity of MCL is discussed in detail by van Dongen [13].

E. Kamada-Kawai

The KK algorithm [3] is commonly described as a "*spring-embedder*" where the vertices $v_1, v_2, ..., v_n \in V$ are considered particles that are mutually connected by springs in a dynamic system. Each vertex $v_i \in V$ is initially located within the canvass with its two-dimensional coordinates (x_i, y_i). The human-readable layout of vertices in the canvass is directly related to the dynamic balance of the energy $\mathcal{E}$ in the spring system. In other words, $\mathcal{E}$ is modeled as a system of springs with a degree of elasticity wherein a desired resting length is achieved when the system reaches an equilibrium. This physical fact is described mathematically in Equation 1. The best layout for a given graph G is at minimum $\mathcal{E}$.

$$\mathcal{E} = 0.5 \times \sum_{i=1..n-1} \sum_{j=i+1..n} k_{ij} (D_{i,j} - L \times d_{ij})^2 \quad (1)$$

$$L = L_0 \times (\max_{i<j} d_{ij})^{-1} \quad (2)$$

$$k_{ij} = K \times (d_{ij})^{-2} \quad (3)$$

In Equation 1, $D_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{0.5}$ is the Euclidean distance between vertices v_i and v_j while d_{ij} is the graph theoretic shortest path [25]. L is the desired clusterings in each connected component. length of the canvass edge. However, when the size of the canvass edge is already constrained, say as L_0, L now (Equation 2) depends on the graph theoretic diameter [25], which is the distance between the farthest pair of vertices in a graph. The coefficient k_{ij} (Equation 3) quantifies the strength of the spring that connects v_i and v_j. In Equation 3, K is a constant.

Given an initial location for each vertex, KK first calculates the "*energy*" or the sum of spring tension for each vertex. The initial vertex location is usually randomly assigned within the canvass. Some implementations of KK randomly initialize the vertices along the diameter of a circle. Whichever vertex initialization procedure is used, KK first finds the vertex $v*$ with the highest energy. It then uses a modified Newton-Raphson procedure [26] to compute the new positions of $v*$ until the energy in the graph is minimum, or below a certain threshold ϵ. The necessary condition to find the minimum is,

$$\partial \mathcal{E} / \partial x_m = \partial \mathcal{E} / \partial y_m = 0, \forall 1 \leq m \leq |V|.$$

The above condition can be calculated by taking the derivatives of Equation 1 with respect to x_m and y_m:

$$\partial \mathcal{E} / \partial x_m = \sum_{i \neq m} k_{mi} \{(x_m - x_i) - l_{mi} (x_m - x_i)/D_{mi}\}, \quad (4)$$

$$\partial \mathcal{E} / \partial y_m = \sum_{i \neq m} k_{mi} \{(y_m - y_i) - l_{mi} (y_m - y_i)/D_{mi}\}. \quad (5)$$

After this, KK looks again for the vertex with the next highest energy and begins moving it. When all vertices have been moved, KK stops and the graph drawing is completed. The complexity of the original KK algorithm is $0(|V|^3)$ which is just equivalent to finding the distances of all pairs of vertices in G (i.e., the simple shortest-path algorithm of Floyd). After that, KK requires $0(|V|^2)$ to compute the Newton-Raphson iteration for all high-energy vertices. A thorough complexity analysis of KK can be found on the authors' original work [3].

Because of the ease of using the KK algorithm for drawing graphs, several modifications have been made to it. One of them is the modification for input graphs with non-uniformed vertex sizes. This modification uses an iterative KK (IKK) where a layout for a graph with arbitrarily sized-vertices is found by iteratively finding a nice layout of a similar graph with weighted edges and dimensionless vertices [27].

F. Drawing constraints

The literature is not lacking on methodologies that allow one to visualize graph structures. In some of these methods, positioning of vertices are restricted to some location within the drawing canvass. For example,

vertices could be located on grid points [8, 28], within concentric circles [29], or along parallel lines [6, 26]. Edges, on the other hand, maybe drawn as straight lines, polygonal lines, or curves. In our drawings, we did not put a constraint on the location of the vertices, while we have drawn the edges as straight lines. The main task of our algorithm is, therefore, to find a location for the vertices of a given graph such that the number of edge crossings is minimized, and at the same time, vertices and edges are uniformly distributed within the canvass for easier readability by humans.

IV. HYBRID DRAWING

We discuss the procedure for our HGDA using the graph shown in Figure 2 as an illustrative example. The procedure is as follows:

1. On an input $G(V, E)$, run DFS to output n subgraphs $\{G_1(V_1, E_1), G_2(V_2, E_2), ..., G_n(V_n, E_n)\}$. Here, $V = V_1 \cup V_2 \cup \cdots \cup V_n$, $E = E_1 \cup E_2 \cup \cdots \cup E_n$, and $n \leq |V|$. As discussed above, this step has a complexity of $\Theta(|E|)$.
2. For each sub-graph G_i, run MCL to find the clusters in each G_i, $\forall i = 1, 2, ..., n$. The output of the ith MCL is m clusters $\{C_{i,1}, C_{i,2}, ..., C_{i,m}\}$. Each cluster $C_{i,j}$ has an associated set of vertices $v_{i,j}$. Here, $V_i = v_{i,1} \cup v_{i,2} \cup \cdots \cup v_{i,m}$ and $m \leq |V_i|$. This step has a complexity of $O(n \times (\max_{i=1\,..\,n}(|V_i|))^3)$.
3. For each cluster $C_{i,j}$, run KK to rearrange the vertices in $v_{i,j}$. This step has a time complexity of $O(m \times n \times (\max(|v_{1,1}|, ..., |v_{m,n}|))^3)$.
4. Consider each cluster $C_{i,j}$ as one big "*phantom vertex*" for a temporary subgraph G'_i. If there is at least one edge going from one vertex in the current cluster $C_{i,j}$ to another vertex in another cluster $C_{i,k}$, create a "*phantom edge*" connecting $C_{i,j}$ and $C_{i,k}$. The complexity of this step is $O(n \times m)$ to connect the m phantom vertices with $m - 1$ phantom edges.
5. Run IKK on G_i to rearrange the clusters within the sub-graph G_i. It should be noted that because clusters are now considered as a vertex for the sub-graph G_i, the phantom vertex has already gained its own size. Because of this, IKK is able to draw nice layouts for graphs with vertices that have different sizes. The complexity of this step is $O(n \times m^3)$ because there are only n subgraphs with m phantom vertices each.
6. Consider each G_i as one phantom vertex for a temporary graph $G*$. Since all subgraphs are disconnected from each other, make each G_i be connected to at most two other phantom vertices only and no two phantom vertices have at least one same vertex connected into it to avoid creating a cyclic graph. As in step 4, the complexity of this step is $O(n)$ to connect the n phantom vertices with $n - 1$ phantom edges.
7. Run IKK on $G*$ to rearrange the sub-graphs. Again, using IKK is useful here because sub-graphs, which are now considered as phantom vertices, will be of different sizes and has dimensions. The complexity of this step is $O(n^3)$ because there are only n phantom vertices corresponding to n subgraphs.

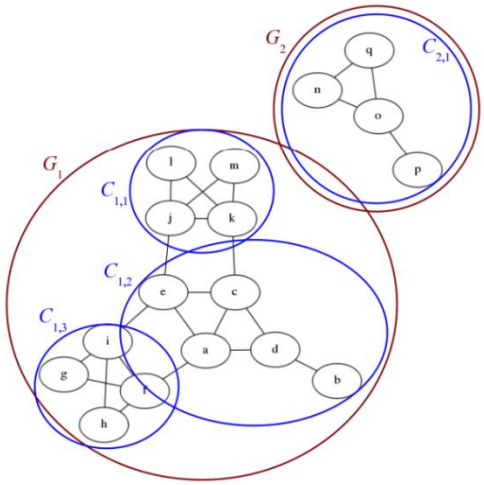

Figure 2. An example graph G with two unconnected subgraphs G_1 and G_2 (colored in the digital format of this paper). The red circles identify the two subgraphs G_1 and G_2 found by the first step. The blue circles identify the four clusters $C_{1,1}$, $C_{1,2}$, $C_{1,2}$ and $C_{2,1}$ found by the second step. There are three clusters in G_1 and only one in G_2. The nodes in each cluster were drawn in step 3. Clusters $C_{1,1}$, $C_{1,2}$ and $C_{1,3}$ were considered as "phantom vertices" and were arranged using IKK in steps 4 and 5. Subgraphs G_1 and G_2 were considered as "phantom vertices" and were arranged using IKK in steps 6 and 7.

A. Fully-connected graphs as input

On an input of a fully-connected graph $G(V, E)$, HGDA will have a complexity of $\theta(|V| \times (|V| - 1)/2) = O(|V|^2)$ in step one. Step two, however, will have $O(|V|^3)$ since there is only one subgraph and the lone subgraph has $|V|$ vertices. Since G is fully connected, only one cluster will be created from MCL and thus step three will have a time complexity of $O(|V|^3)$. Each remaining steps will only run in $\theta(1)$ because the number of clusters found is one, while the number of subgraphs created is also one. Thus, for a fully-connected G as an

input, HGDA will run in $O(|V|^2 + O(|V|^3) + O(|V|^3) + \theta(1) + \theta(1) = O(|V|^3)$.

B. Graphs with E = ∅ as input

On an input of a graph $G(V, \emptyset)$ with no edge, this means that there are $|V|$ sub-graphs, each with only one vertex. Step one will have a zero time complexity. However, step two is exactly $\theta(|V|)$, while step three is $O(1)$. Steps five and seven will run $O(|V|)$ and $O(|V|^3)$, respectively. Thus, for an input of $G(V, \emptyset)$, HGDA will run in $\theta(|V|)+O(1)+O(|V|)+O(|V|^3) = O(|V|^3)$.

V. Parallel Implementation

Our proposed HGDA needs to be run on parallel processors in order to efficiently draw large, naturally-clustered, disconnected graphs. In this section, we present our implementation of the HGDA over a parallel random access machine (PRAM) architecture and derive the corresponding parallel complexities per step. We assume here that our PRAM has p processing units (PUs) that can compute in parallel.

A. Parallel DFS

The search for connected components of the input graph G can be parallelized by partitioning the adjacency matrix A into p parts and then assigning each part to one of p PUs. Each PU P_i has an associated subgraph G_i of G, where $G_i(V, E_i)$ and E_i are the set of edges that correspond to the portion of A assigned to P_i. We implement the following steps:

1. Each P_i computes the depth-first spanning forest of G_i to construct p spanning forests; and then
2. We merge the spanning forests pairwise until only one spanning forest remains.

The remaining spanning forest has the property that two vertices u and v are in the same connected component if they are in the same tree. Step 1 above can be computed sequentially by using any of the Kruskal [30], Prim [31], or Sollin [32] algorithms. However, a parallel algorithm exists that uses $p = |V|^2$ on a concurrent-read, exclusive-write PRAM to solve the problem in time $\Theta(\log^2|V|)$ [33]. To implement step 2 above efficiently, our parallel implementation uses disjoint sets of edges. We assumed that each tree $t_{i,j}$ in the spanning forest T_i of a subgraph G_i of G is represented by a set and that all pairwise sets for different trees are disjoint. We used the following functions to be applied on these disjoint sets:

find(x): This function finds the $U \ni x$ and returns a pointer to the unique representative element $u \in U$.

union(x, y):This function returns a pointer to $X \cup Y$, where $X \ni x$, $Y \ni y$, and $X \cap Y = \emptyset$.

We merge the two spanning forests T_i and T_j with at most $|V| - 1$ edges of T_i with the edges of T_j. For each edge $(u, v) \in T_i$, a **find** operation is performed for each u and v to determine if they are in T_j. If not, then the two trees containing u and v are united by the **union** function. We can see here that merging T_i and T_j requires at most $2(|V| - 1)$ **find** calls and $(|V| - 1)$ **union** calls. Thus, the cost of merging is $O(|V|)$. However, this parallel DFS has a parallel complexity of $\theta((\log^2 |V|)$ because it is dominated by step 1 above.

B. Parallel MCL

Since MCL is based on the simulation of stochastic "*current*" flow in graphs, an analytical method cannot be performed for implementing the parallel MCL over PRAM. However, several implementations, such as those by Olman, *et al*. [34] and Bustamam, *et al*. [35], have been performed over a message-passing architecture wherein the respective runtimes were experimentally determined.

VI. Conclusion

We developed a hybrid graph drawing algorithm by incorporating in series:

1. DFS to find the n connected components G_i ($\forall i$ = 1,...,n) of an input graph G,
2. MCL to find the m clusters of vertices in each connected component,
3. KK to layout the vertices in jth cluster,
4. IKK to layout the clusters as *phantom vertices*, and
5. Another IKK to layout the components as another *phantom vertices*.

We derived the runtime complexity of our hybrid algorithm by considering input graphs in extreme cases: a fully connected graph $G(V, E)$ and a graph with no edge $G(V, \emptyset)$. With these input graphs, we found out that HGDA has $O(|V|^3)$ runtime complexity, where V is the set of vertices of the input graph G. Although we found that the runtime of HGDA is slower than that of the KK or IKK, our purpose here is not to improve the runtime of the drawing algorithm, but instead to hopefully "*nicely*" draw large, naturally-clustered, and disconnected graphs that usually model the complex inter-relationships among entities in social, biological, natural, and artificial networks. We designed an implementation of parallel DFS over a PRAM and found its parallel runtime to be $\Theta(\log^2 |V|)$ if $p = |V|^2$. As an extension of this work, at least two of our concurrent efforts are already underway to objectively define and

provide measurement for what is subjectively called "*nicely drawn*" graphs.

ACKNOWLEDGEMENTS

This research effort is funded by the Institute of Computer Science (ICS) of the University of the Philippines Los Baños (UPLB) under the research programs *Structural Characterization and Temporal Dynamics of Various Natural, Social and Artificial Networks in the Philippines* and *Evaluating Approaches for Modeling and Simulating the Structure and Dynamics of Artificial Societies*.

Ms. Fatima M. Moncada, University Research Associate of the PCAARRD-funded R&D Program *National Pork Traceability System*, is credited for her valuable suggestions in making this paper readable. We also thank the nameless reviewers of this journal whose suggestions vastly contributed to the improvement of the paper.

REFERENCES

[1] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. 1998. **Graph Drawing: Algorithms for the Visualization of Graphs**. Prentice-Hall Inc. :New Jersey, USA, pp 397 (ISBN: 0133016153).

[2] D.E. Knuth. 1963. *Computer-drawn flowcharts*. **Communications of the ACM** 6(9): 555–563 (DOI: 10.1145/367593.367620).

[3] T. Kamada and S. Kawai. 1989. *An algorithm for drawing general undirected graphs*. **Information Processing Letters** 31(1):7–15 (DOI: 10.1016/0020-0190(89)90102-6).

[4] C. Batini, L. Furlani, and E. Nardelli. 1985. *What is a good diagram? A pragmatic approach*. In **Proceedings of the 4th International Conference on the Entity Relationship Approach**, Chicago, IL, IEEE Computer Society Press, pp 312–319 (ISBN: 0818606452)

[5] H.C. Purchase, R.F. Cohen, and M. James. 1996. *Validating graph drawing aesthetics*. In F.-J. Brandenburg (ed) **Proceedings of the 3rd International Symposium on Graph Drawing** (GD 95), Passau, Germany, (Lecture Notes in Computer Science volume 1027) pp 435–446 (ISBN: 3540607234).

[6] K. Sugiyama, S. Tagawa, and M. Toda. 1981. *Methods for visual understanding of hierarchical system structures*. **IEEE Transactions on Systems, Man, and Cybernetics** 11(2):109–125 (DOI: 10.1109/ TSMC.1981.4308636).

[7] C. Kosak, J. Marks, and S.M. Shieber. 1994. *Automating the layout of network diagrams with specified visual organization*. **IEEE Transactions on Systems, Man, and Cybernetics** 24(3):440–454 (DOI: 10.1109/21.278993).

[8] R. Tamassia, G. Di Battista, and C. Batini. 1988. *Automatic graph drawing and readability of diagrams*. **IEEE Transactions on Systems, Man, and Cybernetics** 18 (1):61–79 (DOI: 10.1109/ 21.87055).

[9] A. Quigley and P. Eades. (2001). *FADE: Graph drawing, clustering and visual abstraction*. In J. Marks (ed) **Proceedings of the 8th International Symposium on Graph Drawing** (GD 2000), Colonial Williamsburg, Virginia, USA, (Lecture Notes in Computer Science volume 2265) pp 197–210 (ISBN: 3-540-41554-8).

[10] S. Bender-deMoll and D.A. McFarland. 2006. *The art and science of dynamic network visualization*. **Journal of Social Structure** 7(2) (http://www.cmu.edu/joss).

[11] G. Karypis and V. Kumar. 1998) *A fast and high quality multilevel scheme for partitioning irregular graphs*. **SIAM Journal on Scientific Computing** 20(1):359–392 (DOI: 10.1137/S1064827595287997).

[12] G.L. Miller, S.-H. Teng, and S.A. Vavasis. 1991. *A unified geometric approach to graph separators*. In **Proceedings of the 32nd Annual Symposium on Foundations of Computer Science**, IEEE Computer Society Press, pp 538-547 (DOI: 10.1109/SFCS.1991.185417).

[13] S.M. van Dongen. 2007. **Graph Clustering by Flow Simulation**. PhD thesis, University of Utrecht, 2000.

[14] S.E. Schaeffer. 2007. *Graph clustering*. **Computer Science Review** 1(1):27–64 (DOI: 10.1016/ j.cosrev.2007.05.001).

[15] M.A. Weiss. 1996. **Data Structures and Algorithm Analysis in C, 2nd Edition**. Addison Wesley: Boston, MA, USA, pp 600 (ISBN: 0201498405).

[16] J. Travers and S. Milgram. 1969. *An experimental study of the small world problem*. **Sociometry** 32(4):425–443.

[17] D.J. Watts and S.H. Strogatz. 1998. *Collective dynamics of small world networks*. **Nature** 393:440–442 (DOI: 10.1038/30918).

[18] R. Albert and A.L. Barabasi. 2002. *Statistical mechanics of complex networks*. **Reviews of Modern Physics** 74:47–97 (DOI: 10.1103/RevModPhys.74.47).

[19] A.L. Barabasi and E. Bonabeau. 2003. *Scale-free networks*. **Scientific American** 288(5):50–59 (DOI: 10.1038/scientificamerican0503-60).

[20] A.L. Barabasi and R. Albert. 1999. *Emergence of scaling in random networks*. **Science** 286(5439):509–512 (DOI: 10.1126/science.286.5439.509).

[21] J.P. Pabico. 2010. *Authorship patterns in computer science research in the Philippines*. **Philippine Computing Journal** 5(1):1–13 (ISSN: 1908-1995).

[22] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. 2009. **Introduction to Algorithms, 3rd Edition**. MIT Press: Oxford, MA, USA, 1312 pp (ISBN: 0262033844).

[23] J. Woo and S. Sahni. 1989. *Hypercube computing: Connected components*. **Journal of Supercomputing** 3(3):209–234 (DOI: 10.1007/BF00127829).

[24] S. Even and R.E. Tarjan. 1975. *Network flow and testing graph connectivity*. **SIAM Journal on Computing** 4(4):507–518 (DOI: 10.1137/0204043).

[25] G. Chartrand, L. Lesniak, and P. Zhang. 2010. **Graphs and Digraphs, 5th Edition**. Chapman and Hall/CRC: UK, 598 pp (ISBN: 1439826277).

[26] L.A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan. 1987. *A browser for directed graphs*. **Software: Practice and Experience** 17(1): 61–76 (DOI: 10.1002/spe.4380170107).

[27] D. Harel and Y. Koren. 2002. *Drawing graphs with non-uniform vertices*. In **Proceedings of the Working Conference on Advanced Visual Interfaces** (AVI 2002), Trento, Italy, pp 157–166 (DOI: 10.1145/1556262.1556288).

[28] C. Batini, E. Nardelli, and R. Tamassia. 1986. *A layout algorithm for data flow diagrams*. **IEEE Transactions on Software Engineering** 12(4):538–546 (DOI: 10.1109/TSE.1986.6312901).

[29] M.J. Carpano. 1980. *Automatic display of hierachized graphs for computer-aided decision analysis*. **IEEE Transactions on Systems, Man, and Cybernetics** 10:705–715 (DOI: 10.1109/TSMC.1980.4308390).

[30] J.B. Kruskal. 1956. *On the shortest spanning subtree of a graph and the traveling salesman problem*. **Proceedings of the American Mathematical Society** 7(1):48–50 (ISSN: 0002-9939).

[31] R.C. Prim. 1957. *Shortest connection network and some generalizations*. **Bell Systems Technical Journal** 36(6):1389–1401 (DOI: 10.1002/j.1538-7305.1957.tb01515.x).

[32] M. Sollin. 1977. *An algorithm attributed to Sollin*. In S.E. Goodman and S.T. Hedetniemi (eds.) **Introduction to the Design and Analysis of Algorithms.** McGraw-Hill: Cambridge, MA, USA (ISBN: 0070237530).

[33] C. Savage and J. Jaja. 1981. *Fast, efficient parallel algorithms for some graph problems*. **SIAM Journal of Computing** 10(4):682–690 (DOI: 10.1137/0210051).

[34] V. Olman, F. Mao, H. Wu, and Y. Xu. 2009. *Parallel clustering algorithm for large data sets with applications in bioinformatics*. **IEEE/ACM Transactions on Computational Biology and Bioinformatics** 6(2):344–352 (DOI: 10.1109/TCBB.2007.70272).

[35] A. Bustamam, M.S. Sehgal, N.A. Hamilton, S. Wong, M.A. Ragan, and K. Burrage. 2009. *An efficient parallel implementation of Markov clustering algorithm for large-scale protein-protein interaction networks that uses MPI*. In I.M. Arnawa, M. Muhafzan, and S. Bahri (eds.) **Proceedings of the 5th IMT-GT International Conference on Mathematics, Statistics and Their Applications** (ICMSA 2009), Bukittinggi, Indonesia, pages 94–101 (ISSN: 2085-6369).