Graph Similarity Algorithm Evaluation

Tallman Nkgau
Department of Computer Science
University of Botswana
P/Bag UB 00704
Gaborone, Botswana
Email: nkgautz@mopipi.ub.bw

George Anderson
Department of Computer Science
University of Botswana
P/Bag UB 00704
Gaborone, Botswana
Email: andersong@mopipi.ub.bw

***Abstract*—This paper considers the problem of Graph Similarity. We propose a benchmark framework which can be used to study and evaluate graph matching algorithms. Each benchmark in the framework generates similarity scores for a pair of graphs, according to how similar or dissimilar they are. Many algorithms exist, but each implementation uses a different emphasis on graph properties emphasised in the similarity scores. Some place more emphasis on number of nodes, while others place more emphasis on number of edges, etc. Our proposed benchmark framework is a set of graph pairs; each pair contains graphs which are similar/dissimilar according to certain properties, as identified by experts. Graph matching algorithms being evaluated take this benchmark as input and, according to the similarity scores generated for each pair in the benchmark, it can be determined what a particular algorithm focuses on and is biased towards. Thus a classification system for graph similarity algorithms is produced.**

***Keywords*—Graph Similarity; Pattern Recognition; Algorithms**

I. Introduction

In many applications, objects can be modelled as graphs; discrete structures with vertices and edges linking pairs of vertices. There is often a need to compare two objects, or an object and a model to which the object could be related. Graph matching involves finding a correspondence between the nodes and edges of two graphs [1]. Graph similarity involves calculating a score which expresses how similar two graphs are [2]. One approach is, given two graphs, G_A with n vertices and G_B with m vertices, to generate a similarity $n \times m$ matrix,where each cell has a score for how similar the corresponding vertices are to one another, and then calculate an overall score for the two graphs, making use of the best matching combination.

A. Graph Matching in Pattern Recognition

Graph matching has received much coverage in pattern recognition literature [1]. The statistical approach which makes use of feature vectors suffers from the requirement that all feature vectors be the same length, and an inability to capture relationships between components of the feature vector. When objects are represented as graphs, the graph size is in proportion to the complexity of the object and can vary, and relationships between components can be capture using edges [3]. There is often a need to compare graphs. Exact matching is used for exact comparisons of entire graphs or subgraphs. However, in pattern recognition data is often noisy, making it impractical to use exact matching algorithms, therefore *inexact matching algorithms* must be used. This typically generates a score which expresses similarity between two graphs. This is used, for example, to match an input graph, representing an unknown object derived from an image, with a database of graphs derived from known images [4]. The unknown object is classified with the same class as the most similar graph in the database. Other areas of application include synonym extraction [5], computer network analysis, bioinformatics, and web content mining.

B. Research Contribution

To the best of the authors' knowledge, there is no database available for analysing graph similarity algorithms with the purpose of determining their sensitivities and biases. In this study, we propose a set of benchmarks and a framework based on well-known, classic graphs, which can be used to gauge the sensitivity of graph similarity algorithms to differences in various graph properties, given two graphs, such as number of nodes, number of edges, and regularity.

C. Structure of Paper

The remainder of this paper is organised as follows. Section II serves as a literature review related to graph benchmarks and related areas; Section III specifies our notation; Section IV describes the algorithms we consider in our study; Section V discusses the benchmarks in our proposed framework; Section VI describes our experiments and discusses results; Section VII summarizes our framework; and Section VIII has our conclusion.

II. Background

There is a dearth of literature on benchmarking for analysis of graph similarity algorithms. In this section we cover related benchmarking efforts. There are various datasets available for Pattern Recognition and Machine Learning, such as the one described by Lichman [6] and hosted at the University of California. This contains over 300 data sets for various classification, regression problems, and clustering problems, but are not targeted for the same use as ours, which is for analysing graph similarity algorithms. Foggia et al. [7] and De Santo et al. [8] produced a database of synthetically generated graphs for the purpose of comparing graph isomorphism algorithms. Foggia et al. [9] had also evaluated graph isomorphism algorithms using the database. However, their focus was not graph similarity algorithm analysis.

III. NOTATION

An directed graph is an ordered pair $G = (V, E)$ where V is its set of vertices and E is its set of edges/arcs. In keeping with [10], an edge $e = (i, j) \in E$ is also identified with its source $s(e) = i$ and target $t(e) = j$. Vertex i is called the *in-neighbor* of vertex j, and vertex j is called the *out-neighbor* of vertex i. For a vertex $i \in V$, $id(i)$ and $od(i)$ refers to the number of in-neigbours and out-neghbors of i, respectively. The degree of a vertex $i \in V$ is given by $d(i) = id(i) + od(i)$. For a vertex $i \in V$, $N_G(i)$ is the set of neighbors of i in G. $\|X\|_2$ is the usual euclidean norm of matrix X. the matrix $[1]^{n\times n}$ is the all-ones matrix of size $m \times n$. Given two graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$ their modular product is defined as the graph $G = (V, E)$ where $V = V_A \times V_B$ and $((u, v), (x, y)) \in V$ whenever $(u, x) \in E_A$ and $(v, y) \in E_B$ or $(u, x) \notin E_A$ and $(v, y) \notin E_B$. It is known that the largest clique (maximal complete induced subgraph) in the modular product graph of two graphs corresponds to the largest induced common subgraph of the graphs.

IV. GRAPH SIMILARITY ALGORITHMS

In this section, we give overviews of the three graph similarity algorithms we consider: Jaccard Coefficient (Jaccard) and Neighbor Matching graph similarity (NM), and a third, the Zager-Verghese Similarity Algorithm.

A. Jaccard Coefficient

Hattori et al. [11] presented an approach for the use of the Jaccard Coefficient in matching of chemical compound graphs. The algorithm is listed in Figure 1. They made use of a modular product of the two input graphs, which produces a structure that matches each node in G_A with each node in G_B. Edges join nodes from both graphs that are similar. A maximum common subgraph is then computed, which is equivalent to computing the maximum clique in the modular product. The size of this maximum common subgraph is used as a measure of the amount of "overlap" the two graphs have. A simple equation then produces a similarity score. An important issue to note is that finding the maximum clique of a graph is an NP-complete problem, with not efficient solutions known. We make use of the Bit-Board Maximum Clique algorithm (BBMC) as described by Prosser [12]. He used a Java implementation however, while we used a C++ one, for performance reasons. Bunke [13] also discussed use of the maximal common subgraph as a graph distance metric.

Input: Undirected graphs G_A, G_B
Output: Jaccard Similarity Score
1: $M \leftarrow GetModularProduct(G_A, G_B)$
2: $maxClique \leftarrow GetMaxClique(\mid M \mid, M)$
3: $sizeMCS \leftarrow \mid maxClique \mid$
4: $n_a = \mid G_A \mid, n_b = \mid G_B \mid$
5: **return** $sizeMCS/(n_a + n_b - sizeMCS)$

Fig. 1. Jaccard Graph Similarity Algorithm [11]

B. Neighbor Matching Graph Similarity

Nikolić [10] presented an approach to calculating graph similarity scores based on matching of vertex neighborhoods. The idea of neighborhood-matching stems from the fact that often objects are evaluated for their similarity based on their features [14]. Given two directed graphs, $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$, Nikolić [10] neighbor matching algorithm works by computing vertex similarity between the two graphs. It iteratively maintains a similarity matrix $\mathbf{X} = [\mathbf{x_{ij}}]$ during computations.The algorithm's conceptual design is based on the following principles: two vertices $i \in V_A$ and $j \in V_B$ are similar if their neighbors can be similarly matched. To accomplish this, the neighbor matching algorithm computes similarities between in-neigbors and between out-neighbors of $i \in V_A$ and $j \in V_B$. To match in-neighbors (and out-neighbors), the algorithm builds a weighted bipartite graph using in-neighbors of i as one partition and in-neighbors of j as the other partition. The weight of each edge, $e = (u, v)$, at the $k + 1$st iteration, is taken as $w(u, v) = x^k_{uv}$, which is the similarity score between vertex u and v at the end of the kth iteration. We take $x^0_{uv} = 1$ for all $u \in V_A$ and $v \in V_B$. We solve weighted bipartite matching problem (maximization) using the Hungarian algorithm [15]. Figure 2 gives the neighbor matching algorithm modified for undirected graphs. Since the graphs are undirected, we only consider the in-neighbors of vertices $i \in V_A$ and $j \in V_B$. The procedure $MWBMatching(N_i, N_j, w)$ computes the maximum weight matching problem between the sets N_i and N_j with edge weights given by w. The functions f and g represents the mapping between the enumeration of the matched edges and the two sets. More specifically, for a matching of size $k = \min\{|N_i|, |N_j|\}$, $(f(l), g(l))$ is a matched edge for all $l = 1, 2, \ldots, k$. $n_{ij} = \min\{|N_i|, |N_j|\}$ and $m_{ij} = \max\{|N_i|, |N_j|\}$.

Input: Undirected graphs G_A, G_B, Maximum number of iterations N, Error level ϵ
Output: Node similarity matrix
1: $n_a = \mid V_A \mid, n_b = \mid V_B \mid$
2: $X^{n_a \times n_b} \leftarrow [1]^{n_a \times n_b}$
3: $k \leftarrow 0$
4: **repeat**
5: **for all** $(i, j) \in V_A \times V_B$ **do**
6: $N_i \leftarrow N_{G_A}(i)$
7: $N_j \leftarrow N_{G_B}(j)$
8: $(M, f, g) \leftarrow MWBMatching(N_i, N_j, w)$
9:

$$x_{ij}{}^{k+1} \leftarrow \frac{1}{m_{ij}} \sum_{l=1}^{n_{ij}} x^k_{f_{ij}(l) g_{ij}(l)}$$

10: **end for**
11:

$$X^{k+1} \leftarrow \frac{X^{k+1}}{\|X^{k+1}\|_2}$$

12: $k \leftarrow k + 1$
13: **until** $k > N$ or $\mid x^k_{ij} - x^{k-1}_{ij} \mid \leq \epsilon \ \forall (i, j) \in V_A \times V_B$
14: **return** X^{k-1}

Fig. 2. Nikolić [10] Neighbor Matching algorithm modified for undirected graphs

C. Zager-Verghese Similarity Algorithm

Zager and Verghese [2] discussed an approach to computing graph similarity scores, as well as graph matching. In their approach, like that of Nikolič [10] in the previous section, takes advantage of the concept of a neighborhood of a node. Here, an edge or node in G_A is considered similar to another one in G_B if their respective neighborhoods are similar. This leads to an iterative approach in which similarity scores propagate to neighboring elements at each time step, with the process complete when effective changes no longer take place. Their work builds on work by Blondel et al. [5]. They make use of a source edge matrix and target edge matrix, constructed as shown in Equations 1 and 2 respectively.

$$[A_S]_{ij} = \begin{cases} 1 & S_A(j) = i \\ 0 & otherwise \end{cases} \quad (1)$$

$$[A_T]_{ij} = \begin{cases} 1 & S_T(j) = i \\ 0 & otherwise \end{cases} \quad (2)$$

The algorithms relies on matrix iteration and the X_k and Y_k matrices which keep track of vertex and edge similarities respectively. After convergence, a maximum weighted bipartite algorithms is used, as in Neighbor Matching, to match vertices and edges. Figure 3 shows the algorithm. $MWBMatching2()$ matches rows and columns of, first of all X_k, then Y_k, and produces similarity scores for each. we modified the Zager-Verghese algorithm to work with undirected graphs, by doubling the number of edges such that, for each undirected edge from node a to b, there are two directed edges, $a \rightarrow b$ and $b \rightarrow a$.

Input: Undirected graphs G_A, G_B
Output: Zager-Verghese Graph Similarity Score
1: Initialize A_S, A_T, B_S, and B_T matrices.
2: $k = 0$
3: $r_{as} =\mid A_S rows \mid, r_{bs} =\mid B_S rows \mid$
4: $c_{as} =\mid A_S columns \mid, c_{bs} =\mid B_S columns \mid$
5: $X_k^{r_{bs} \times r_{as}} \leftarrow [1]^{r_{bs} \times r_{as}}$
6: $Y_k^{c_{bs} \times c_{as}} \leftarrow [1]^{c_{bs} \times c_{as}}$
7: **repeat**
8: $k = k + 1$
9: $Y_k \leftarrow B_S^\top X_{k-1} A_S + B_T^\top X_{k-1} A_T$
10: $X_k \leftarrow B_S Y_{k-1} A_S^\top + B_T Y_{k-1} A_T^\top$
11: $$Y_k \leftarrow \frac{Y_k}{\|Y_k\|}$$
12: $$X_k \leftarrow \frac{X_k}{\|X_k\|}$$
13: **until** Convergence with X_k and Y_k
14: $NodeSimilarity \leftarrow MWBMatching2(X_k)$
15: $EdgeSimilarity \leftarrow MWBMatching2(Y_k)$
16: $Similarity \leftarrow \frac{NodeSimilarity + EdgeSimilarity}{2}$
17: **return** $Similarity$

Fig. 3. Zager-Verghese Graph Similarity Algorithm [2]

V. Benchmarks

We made use of the python-based open source graph library NetworkX [16], [17] to generate the following categories of graphs, which serve as benchmarks to evaluate graph similarity. Most graphs are based on the circular ladder graph, CL_n, which is a Cartesian product of a cycle of length $n \geq 3$ and an edge [18]. It has $2n$ nodes and $3n$ edges.CL_n consists of two concentric $n-$cycles in which each of the n pairs of concentric nodes are joined by an edge. Node labels are the integers 0 to n-1. [16]. CL_n is one of the classic graphs. A decision was made to use a classic graph for our benchmarks since users will be familiar with it; out of the commonly known classic graphs, CL_n has a good balance of nodes and edges, hence the decision to use it for some of the applicable benchmarks. It is also uniform across all regions. The circular ladder graphs were modified to remove two random edges in order that they do not have the regularity property, since all circular ladder graphs are 3-regular.

A. Experiment 1 Benchmark: Identical Graphs

Experiment 1 compares identical graphs, in order to discover the scores various algorithms produce when comparing pairs of graphs of identical sizes, over a range of graph sizes and structures. This experiment is very important in order to determine the highest score an algorithm can produce, since the data set is the best possible scenario (identical graphs). We generate circular ladder graphs, starting from 6 vertices, up to 44 vertices.

B. Experiment 2 Benchmark: Isomorphic Graphs

Experiment 2 compares isomorphic graphs, in order to discover the scores various algorithms produce when comparing pairs of isomorphic graphs; this is done to determine how sensitive an algorithm is to isomorphism. We generate circular ladder graphs, starting from 6 vertices, up to 44 vertices. For each graph, two random edges are removed, to generate the first graph in the pair, then a random relabelling is performed in order to generate the second graph in the pair.

C. Experiment 3 Benchmark: Different Number of Nodes

Experiment 3 compares graphs with different numbers of nodes. This experiment determines an algorithm's sensitivity to node size difference of a pair of graphs. We generate circular ladder graphs, from 6 nodes up to 46 nodes. We compare a graph with 6 nodes, with a graph with 8, then 8 with 10, and so on.

D. Experiment 4 Benchmark: Different Number of Edges

Experiment 4 compares graphs with different numbers of edges. This experiment determines an algorithm's sensitivity to edge size difference of a pair of graphs. We generated a circular ladder graph with 30 nodes, then, for each iteration, randomly selected two unconnected vertices, and joined them to form a new graph. This process continued for 20 graphs.

E. Experiment 5 Benchmark: Different Regularity

Experiment 5 compares graphs with the same number of nodes (26) but different regularity. This experiment determines an algorithm's sensitivity to differences in regularity. We generated random regular graphs of 26 nodes starting with 3 regular, 4 regular, up to 23 regular graphs. Comparison was done between 3 and 4-regular versions, 4 and 5-regular versions, etc.

F. Experiment 6 Benchmark: Complete Graphs with Different Numbers of Nodes

Experiment 6 compares complete graphs with different numbers of nodes. This experiment determines an algorithm's sensitivity to completeness of a pair of graphs. We generated complete graphs from 3 up to 23 nodes.

VI. Experiments, Results and Discussion

Our benchmark graphs were saved in the open and portable GraphML format [19]. The graph similarity algorithms were implemented originally in Python, using the NetworkX library, but later reimplemented in C++, using the C++ Boost Graph Library [20]. This was due to very poor performance with Python and NetworkX. In addition, the Zager-Verghese algorithm was implemented with version 6.600.4 of the Armadillo C++ linear algebra library [21] and the Dlib C++ machine learning and optimization library [22], mainly for the maximum weighted bipartite matching algorithm (Hungarian algorithm). Compilation was done using the GNU g++ 5.4.0 compiler with the highest level of optimization (-O3), for best execution performance. Experiments were run on an Ubuntu Server 16.04 Linux operating system, running on an Oracle Sun X3-2 server with 2 CPUs, 16 cores and 64GB of RAM. The CPUs were Intel Xeon E5-2660 running at 2.2GHz.

Each experiment involved comparing 20 pairs of graphs. Comparison of each pair was run as a separate process. Comparisons were run in batches of 10, in order to take advantage of the parallel capability of a machine with 16 cores, but at the same time allowing careful monitoring and timing.

A. Experiment 1 Benchmark Results: Identical Graphs

As evident in the Experiment 1 results in Table I, Jaccard and NM yield scores of 1 and close to 1 respectively for each pair of identical graphs that are compared. This allows us to determine what maximum score is output for identical graphs. Which is 1 for each algorithm. It also determines that both algorithms are sensitive to pairs of graphs with equal structures. The values for ZV are much smaller, indicating that ZV was mainly developed for neighbor matching, not similarity comparisons. However, the magnitude is less important than relative similarity scores. ZV produces lower scores for larger graphs.

B. Experiment 2 Benchmark Results: Isomorphic Graphs

As evident in the Experiment 2 results in Table II, Jaccard produced a score of 1, for all 20 pairs of isomorphic graphs, which means it is not sensitive to isomorphism, that is, it recognizes that both graphs in a pair are have basically identical structures. Neigbor matching however, is sensitive to isomorphism. This could be partly due to Neighbor matching using the Hungarian algorithm to map nodes of graph A to nodes of graph B, making use of a approximation involving conversion of real numbers to long integers. ZV has low scores for all graph pairs, but identical to Experiment 1, indicating no sensitivity to isomorphism.

C. Experiment 3 Benchmark Results: Different Number of Nodes

As evident in the Experiment 3 results in Table III, Jaccard is more sensitive to differences in number of nodes, having scores ranging from 0.4 to 0.869, while NM has a consistent score of 1. Since NM examines neighborhoods, the neighborhoods might appear similar given a pair of nodes to be matched, leading to the score of 1. One can also see that as the sizes of the graphs increases, Jaccard scores approach 1 because there is less relative difference between the pair of graphs. However, while ZV is sensitive to number of nodes, its similarity scores decrease as the number of vertices goes up.

D. Experiment 4 Benchmark Results: Different Number of Edges

This experiment relied on a relatively large pair of graphs, which caused the Jaccard processes to be terminated by the operating system due to memory exhaustion. There are therefore no scores for Jaccard. As evident in the Experiment 4 results in Table IV, NM is almost insensitive to edge differences. This is due to the focus on vertices by the algorithm. ZV is sensitive to edge differences, and more so as the sizes of graph pairs increases.

E. Experiment 5 Benchmark Results: Different Regularity

As evident in the Experiment 5 results in Table V, Jaccard is more sensitive to differences in regularity. With NM, the larger the graph pairs, the more similar they are, so less difference reported by the algorithm. ZV is sensitive to regularity, but not as much as it is sensitive to edge differences and node differences. This is evident when observing the slight difference in scores as one goes down the table.

F. Experiment 6 Benchmark Results: Complete Graphs with Different Numbers of Nodes

As evident in the Experiment 6 results in Table VI, Jaccard and NM have similar similarity scores. In fact, for a given pair, NM shows the same similarity as the previous experiment for Jaccard. This is because NM is more slightly sensitive to differences in number of edges. The results also validate our Jaccard implementation, because for each pair Graph A is an induced subgraph of Graph B, so the maximum clique algorithm returns the subgraph, leading to the scores observed. ZV is sensitive to differences in size of complete graph pairs. As it is sensitive to number of edges and nodes.

G. Execution Time Performance for Benchmarks (Jaccard and Neighbor Matching)

Figure 4 shows the performance of the benchmarks with regard to execution time when run using Jaccard and NM.

Each experiment is run in two batches: the first batch has the 10 smallest pairs of graphs, followed by the second batch with the 10 largest pairs of graphs. Each batch is run for a single similarity algorithm at a time, but in parallel, making use of the multicore architecture of our compute server. The figures in the chart represent the makespan of a batch i.e. the maximum time of all graph pairs in the batch, since they are all started at the same time. Neighbor matching has far better performance, with a speedup over Jaccard (Jaccard Max Batch Time/NM Max Batch Time) ranging from 17 to 520617. The speedup over ZV (ZV Max Batch Time/NM Max Batch Time) ranges from 0.736 to 1458. The speedup of ZV over Jaccard (Jaccard Max Batch Time/ZV Max Batch Time) ranges from 0.37 to 47042. This is due to the Maximum Common Clique algorithm which Jaccard depends on, which requires exponential time. There is no Jaccard time for Experiment 4 since it ran out of memory due to the sizes of the pair of graphs.

TABLE I. EXPERIMENT 1 RESULTS (IDENTICAL GRAPHS). ALL GRAPHS ARE MODIFIED CIRCULAR LADDER GRAPHS WITH NUMBER OF NODES SPECIFIED IN THE GRAPH A AND GRAPH B COLUMNS. J=JACCARD, NM=NEIGHBOR MATCHING, ZV=ZAGER-VERGHESE

A	B	J	Time	NM	Time	ZV	Time
6	6	1	1481	0.777778	10700	0.159516	7781
8	8	1	3374	0.729167	1623	0.127676	271748
10	10	1	4819	0.933333	25515	0.07091	6683
12	12	1	17840	0.777778	15465	0.0586745	51439
14	14	1	26945	0.809524	2366	0.0502632	121079
16	16	1	46870	0.875	2764	0.0535419	636217
18	18	1	97152	0.851852	11556	0.0383347	402194
20	20	1	286147	0.9	14701	0.0362018	943292
22	22	1	420918	0.878788	16839	0.031436	717481
24	24	1	2345884	0.944444	24716	0.0289042	1477257
26	26	1	9153440	0.897436	12870	0.026124	2686626
28	28	1	1217007	0.904762	14175	0.0247157	4405626
30	30	1	14064560	0.933333	13063	0.0226021	5579105
32	32	1	3798443	0.916667	17724	0.021209	6299463
34	34	1	1873051	0.921569	7149	0.0198399	553943
36	36	1	18265274	0.925926	6350	0.0187875	8521219
38	38	1	251346448	0.929825	8310	0.0177522	8657167
40	40	1	36929565	0.95	6653	0.0170549	9654093
42	42	1	7829478	0.936508	11805	0.0160127	1125745
44	44	1	32045675	0.954545	20119	0.0152727	26966964

VII. FRAMEWORK

Our framework consists of the following components.

1) A collection of graph pairs based on well known graphs. Currently most of our pairs are based on the circular ladder graph. The graph pairs are saved in the GraphML format.
2) The graph pairs have predetermined properties, such as numbers of nodes, edges, and regularity.
3) A collection of graph similarity algorithm implementations which output four pieces of data: graph A identifier, graph B identifier, similarity score, and execution time in microseconds. We use C++ because of its desirable properties of performance and software reuse, and availability of template libraries and high performance linear and graph manipulation libraries.

TABLE II. EXPERIMENT 2 RESULTS (ISOMORPHIC GRAPHS). ALL GRAPHS ARE MODIFIED CIRCULAR LADDER GRAPHS WITH NUMBER OF NODES SPECIFIED IN THE GRAPH A AND GRAPH B COLUMNS. ISOMORPHISM IS ACHIEVED BY A RANDOM RELABELLING OF NODES. J=JACCARD, NM=NEIGHBOR MATCHING, ZV=ZAGER-VERGHESE

A	B	J	Time	NM	Time	ZV	Time
6	6	1	1481	0.777778	10700	0.159516	7781
8	8	1	3374	0.729167	1623	0.127676	271748
10	10	1	4819	0.933333	25515	0.07091	6683
12	12	1	17840	0.777778	15465	0.0586745	51439
14	14	1	26945	0.809524	2366	0.0502632	121079
16	16	1	46870	0.875	2764	0.0535419	636217
18	18	1	97152	0.851852	11556	0.0383347	402194
20	20	1	286147	0.9	14701	0.0362018	943292
22	22	1	420918	0.878788	16839	0.031436	717481
24	24	1	2345884	0.944444	24716	0.0289042	1477257
26	26	1	9153440	0.897436	12870	0.026124	2686626
28	28	1	1217007	0.904762	14175	0.0247157	4405626
30	30	1	14064560	0.933333	13063	0.0226021	5579105
32	32	1	3798443	0.916667	17724	0.021209	6299463
34	34	1	1873051	0.921569	7149	0.0198399	553943
36	36	1	18265274	0.925926	6350	0.0187875	8521219
38	38	1	251346448	0.929825	8310	0.0177522	8657167
40	40	1	36929565	0.95	6653	0.0170549	9654093
42	42	1	7829478	0.936508	11805	0.0160127	1125745
44	44	1	32045675	0.954545	20119	0.0152727	26966964

TABLE III. EXPERIMENT 3 RESULTS (DIFFERENT NUMBER OF NODES). ALL GRAPHS ARE MODIFIED CIRCULAR LADDER GRAPHS, WITH NUMBER OF NODES SPECIFIED IN THE GRAPH A AND GRAPH B COLUMNS. J=JACCARD, NM=NEIGHBOR MATCHING, ZV=ZAGER-VERGHESE

A	B	J	Time	NM	Time	ZV	Time
6	8	0.4	1895	1	6593	0.096225	6504
8	10	0.5	4885	1	11659	0.0745356	1408
10	12	0.571429	11484	1	12056	0.0608581	1582
12	14	0.625	21319	1	14407	0.0514344	2000
14	16	0.666667	41104	1	14811	0.0445435	2252
16	18	0.7	124198	1	14729	0.0392837	10914
18	20	0.727273	170189	1	6278	0.0351364	5276
20	22	0.75	327513	1	2672	0.0317821	3733
22	24	0.769231	459632	1	6834	0.0290129	6582
24	26	0.785714	816846	1	10578	0.026688	3008
26	28	0.8	1278030	1	2880	0.0247083	5596
28	30	0.8125	2187679	1	5644	0.0230022	5657
30	32	0.823529	2919854	1	4165	0.0215166	15631
32	34	0.833333	4676517	1	5995	0.0202113	16786
34	36	0.842105	6081398	1	4476	0.0190554	9349
36	38	0.85	9949459	1	5356	0.0180246	10891
38	40	0.857143	12257890	1	5469	0.0170996	10552
40	42	0.863636	18805388	1	5831	0.016265	15821
42	44	0.869565	22801762	1	6061	0.0155081	13593
44	46	0.875	34941606	1	6678	0.0148185	34499

4) An execution environment to run all similarity programs against all graph pairs. We currently have a bash shell scripting implementation.
5) A strategy for evaluating and comparing graph similarity algorithms.

TABLE IV. EXPERIMENT 4 RESULTS (DIFFERENT NUMBER OF EDGES). ALL GRAPHS ARE BASED ON CIRCULAR LADDER GRAPH WITH 30 NODES, WITH ONE ADDITIONAL EDGE ADDED FOR EACH SUCCESSIVE GRAPH E.G. P2 MEANS 30 NODES PLUS 2 EDGES ADDED. J=JACCARD, NM=NEIGHBOR MATCHING, ZV=ZAGER-VERGHESE

A	B	J	Time	NM	Time	ZV	Time
p1	p2	-	-	0.997727	85951	0.0334458	412912
p2	p3	-	-	0.997955	98728	0.0307193	364332
p3	p4	-	-	0.997727	100276	0.0297811	109624
p4	p5	-	-	0.997727	163798	0.0297714	218809
p5	p6	-	-	0.997727	160737	0.0298484	242002
p6	p7	-	-	0.997955	82315	0.0302806	287515
p7	p8	-	-	0.997727	83990	0.0315741	402710
p8	p9	-	-	0.997727	91426	0.0322117	509423
p9	p10	-	-	0.997955	96421	0.0324618	596683
p10	p11	-	-	0.997727	91540	0.0239835	440803
p11	p12	-	-	0.997727	78510	0.0245946	773929
p12	p13	-	-	0.997727	77713	0.0234753	648419
p13	p14	-	-	0.997727	77051	0.0235959	529640
p14	p15	-	-	0.997727	78299	0.0225148	563062
p15	p16	-	-	0.997727	76823	0.0226236	1033329
p16	p17	-	-	0.997727	77930	0.0221664	1786713
p17	p18	-	-	0.998333	80112	0.0273693	2311478
p18	p19	-	-	0.998214	76180	0.0270741	1997826
p19	p20	-	-	0.997727	75892	0.0246582	2786992
p20	p21	-	-	0.997727	79753	0.0241008	2334520

TABLE V. EXPERIMENT 5 RESULTS (DIFFERENT REGULARITY). ALL GRAPHS HAVE 26 NODES WITH REGULARITY SPECIFIED IN THE GRAPH A AND GRAPH B COLUMNS. J=JACCARD, NM=NEIGHBOR MATCHING, ZV=ZAGER-VERGHESE. J & NM SIMILARITY SCORES ARE ROUNDED TO 3 DECIMAL PLACES TO SAVE SPACE. ZV ROUNDED TO 4 DECIMAL PLACES

A	B	J	Time	NM	Time	ZV	Time
3	4	0.529	8739093386	0.75	10210	0.0248	15038
4	5	0.485	2121027035	0.8	10613	0.0235	16379
5	6	0.444	578318650	0.833	10454	0.0227	160691
6	7	0.405	284220100	0.857	13171	0.0222	102774
7	8	0.368	178261261	0.875	14735	0.0218	141816
8	9	0.368	121506171	0.889	16786	0.0215	146173
9	10	0.333	92051645	0.9	14210	0.0213	68447
10	11	0.333	58666275	0.909	13931	0.0211	124111
11	12	0.333	47549056	0.917	13097	0.0209	185772
12	13	0.333	43596365	0.923	16447	0.0208	121030
13	14	0.333	51683519	0.929	7760	0.0207	321927
14	15	0.333	64917310	0.933	10803	0.0206	638683
15	16	0.333	81330602	0.938	13668	0.0205	660137
16	17	0.368	81406933	0.941	15317	0.0204	531851
17	18	0.368	143604981	0.944	15689	0.0203	548443
18	19	0.405	181703085	0.947	17071	0.0203	697488
19	20	0.444	366669203	0.95	22662	0.0202	686791
20	21	0.486	504108256	0.952	21269	0.0202	699542
21	22	0.529	1229512695	0.955	15799	0.0201	665247
22	23	0.529	9465644875	0.957	25045	0.0201	738265

TABLE VI. EXPERIMENT 6 RESULTS (COMPLETE GRAPHS WITH DIFFERENT NUMBERS OF NODES). ALL GRAPHS ARE COMPLETE, WITH NUMBER OF NODES SPECIFIED IN THE GRAPHA AND GRAPHB COLUMNS. J=JACCARD, NM=NEIGHBOR MATCHING, ZV=ZAGER-VERGHESE

A	B	J	Time	NM	Time	ZV	Time
3	4	0.75	428	0.666667	17706	0.203263	840
4	5	0.8	909	0.75	17915	0.144078	980
5	6	0.833333	960	0.8	10396	0.1117	1174
6	7	0.857143	1546	0.833333	22607	0.0912376	1483
7	8	0.875	2503	0.857143	23998	0.0771251	2666
8	9	0.888889	6453	0.875	20545	0.0667998	2914
9	10	0.9	6338	0.888889	22297	0.0589159	4572
10	11	0.909091	15594	0.9	3622	0.0526983	4224
11	12	0.916667	15762	0.909091	9026	0.0476688	6291
12	13	0.923077	21874	0.916667	29530	0.0435164	22101
13	14	0.928571	31750	0.923077	3373	0.0400298	211693
14	15	0.933333	44063	0.928571	4865	0.0370608	132088
15	16	0.9375	60889	0.933333	10102	0.034502	185509
16	17	0.941176	84753	0.9375	9028	0.0322739	172692
17	18	0.944444	111149	0.941176	7150	0.0303162	328707
18	19	0.947368	147540	0.944444	8881	0.0285825	423623
19	20	0.95	201256	0.947368	9484	0.0270364	329724
20	21	0.952381	247351	0.95	11460	0.0256491	440964
21	22	0.954545	304378	0.952381	11417	0.0243972	328928
22	23	0.956522	355172	0.954545	13453	0.0232618	516830

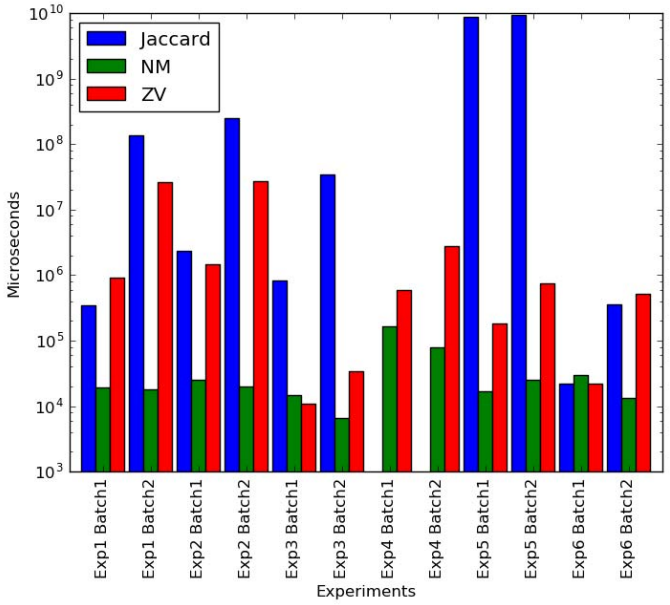

Fig. 4. Execution time for the various batches of experiments, for Jaccard and Neighbor Matching. Expx Batchy is Experiment x Batch y e.g. Exp2 Batch1 is Experiment 2 Batch 1.

VIII. CONCLUSION

Our proposed benchmark framework lists classic graphs which can be used to determine sensitivity of graph similarity algorithms to various properties of graphs, such as number of nodes, number of edges, completeness and regularity. Our case study makes use of a Maximum Common Graph-based algorithm (Jaccard), a neighborhood-based algorithm (NM), and another algorithm based on hubs and authorities (ZV), and discusses their implied differences. We found that NM is more localised in the way it works; that is if the local neighborhood of the vertices are similar, the similarity scores are high, even though the graphs being compared are largely different in number of nodes and number of edges. ZV tends to be sensitive to differences in numbers of basic structures such as numbers of nodes and edges, but not other properties such as regularity and isomorphism. NM is also much faster than Jaccard and ZV. Our planned future work involves growing our framework

with a wider variety of algorithms and a corresponding detailed analysis of the new algorithms in order to come up with more insightful analysis with regards to how benchmark results could determine properties of the graph similarity algorithms. We also intend to reduce the memory usage of Jaccard possibly by converting its recursive approach to an iterative one, when it comes to finding the Maximum Common Subgraph.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International journal of pattern recognition and artificial intelligence*, vol. 18, no. 03, pp. 265–298, 2004.

[2] L. A. Zager and G. C. Verghese, "Graph similarity scoring and matching," *Applied Mathematics Letters*, vol. 21, no. 1, pp. 86–94, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893965907001012

[3] K. Riesen and H. Bunke, "Iam graph database repository for graph based pattern recognition and machine learning," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2008, pp. 287–297.

[4] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865597000603

[5] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. V. Dooren, "A measure of similarity between graph vertices: Applications to synonym extraction and web searching," *SIAM Review*, vol. 46, no. 4, pp. 647–666, 2004. [Online]. Available: http://dx.doi.org/10.1137/S0036144502415960

[6] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[7] P. Foggia, C. Sansone, and M. Vento, "A database of graphs for isomorphism and sub-graph isomorphism benchmarking," in *Proc. of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*, 2001, pp. 176–187.

[8] M. De Santo, P. Foggia, C. Sansone, and M. Vento, "A large database of graphs and its use for benchmarking graph isomorphism algorithms," *Pattern Recogn. Lett.*, vol. 24, no. 8, pp. 1067–1079, May 2003. [Online]. Available: http://dx.doi.org/10.1016/S0167-8655(02)00253-2

[9] P. Foggia, C. Sansone, and M. Vento, "A performance comparison of five algorithms for graph isomorphism."

[10] M. Nikolić, "Measuring similarity of graph nodes by neighbor matching," *Intell. Data Anal.*, vol. 16, no. 6, pp. 865–878, Nov. 2012. [Online]. Available: http://dx.doi.org/10.3233/IDA-2012-00556

[11] M. Hattori, Y. Okuno, S. Goto, and M. Kanehisa, "Heuristics for chemical compound matching," *Genome Informatics*, vol. 14, pp. 144–153, 2003.

[12] P. Prosser, "Exact algorithms for maximum clique: A computational study," *Algorithms*, vol. 5, no. 4, pp. 545–587, 2012.

[13] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern recognition letters*, vol. 19, no. 3, pp. 255–259, 1998.

[14] Z. Lin, M. R. Lyu, and I. King, "Matchsim: a novel neighbor-based similarity measure with maximum neighborhood matching," in *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1613–1616.

[15] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. [Online]. Available: http://dx.doi.org/10.1002/nav.3800020109

[16] A. Hagberg, D. Schult, and P. Swart, *NetworkX Reference*, September 2014. [Online]. Available: http://networkx.github.io/

[17] Networkx Development Team. (2015) Networkx website. [Online]. Available: http://networkx.github.io/

[18] Y. Chen, J. L. Gross, and T. Mansour, "Total embedding distributions of circular ladders," *Journal of Graph Theory*, vol. 74, no. 1, pp. 32–57, 2013.

[19] G. Team, "The graphml file format," *Homepage: http://graphml.graphdrawing.org*, 2015.

[20] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *Boost Graph Library: User Guide and Reference Manual, The*. Pearson Education, 2001.

[21] C. Sanderson and R. Curtin, "Armadillo: a template-based c++ library for linear algebra," *Journal of Open Source Software*, 2016.

[22] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.