Week 03 Report (29 Jul, 2 Aug) - Guangyao Xu

1、Work

1.1 Read paper about algorithms of Graph Edit Distance to select a better quadratic time algorithm

In this paper : 'Improved quadratic time approximation of graph edit distance by combining Hausdorff matching and greedy assignment', the author lists three main quadratic time approximation of graph edit distance including HED(hausdroff Edit Distance), Greedy ED (Greedy Edit Distance) and BP2.

To compare this three algorithms, the author uses five database to measure the accuracy and time cost of these three algorithms. The result is:

Table 2
Accuracy on the test set in percentage. For each database, the best performing methods within a 95% confidence interval are highlighted.

Database	BP	HED	BP-G	BP-G2	BP2 (proposed)
Letters I	**99.7**	97.9	**99.7**	99.1	**99.7**
Letters II	**94.3**	86.9	**94.5**	92.9	**94.1**
Letters III	**89.9**	79.2	**89.2**	**88.7**	**89.6**
Fingerprints	80.3	**82.8**	79.9	76.8	80.2
Molecules	**99.6**	99.2	**99.5**	**99.6**	**99.5**

Table 3
Runtime results in seconds for $N \cdot M$ distances where N and M are the number of graphs in the training and test set, respectively. Results are reported with respect to a Java implementation and AMD Opteron 2354 nodes with 2.2 GHz CPU. The best results are highlighted.

Database	BP	HED	BP-G	BP-G2	BP2 (proposed)
Letters I	43.3	**12.3**	38.6	40.9	18.9
Letters II	36.5	**10.4**	29.0	40.3	20.0
Letters III	47.3	**11.4**	39.5	40.2	16.4
Fingerprints	372.0	**60.6**	297.6	310.4	107.6
Molecules	991.7	**40.1**	237.0	290.5	50.0

Table 4
Average μ and standard deviation σ of the edit distance difference to BP on the test set in percentage. The closest distances are highlighted.

Database	HED $\mu \pm \sigma$	BP-G $\mu \pm \sigma$	BP-G2 $\mu \pm \sigma$	BP2 $\mu \pm \sigma$
Letters I	−27.1 ± 14.4	5.9 ± 8.8	26.6 ± 29.6	**−0.3** ± 2.0
Letters II	−54.2 ± 15.5	9.6 ± 15.7	8.4 ± 15.2	**0.8** ± 13.4
Letters III	−32.7 ± 13.6	14.3 ± 23.6	9.4 ± 15.7	**1.3** ± 8.2
Fingerprints	−46.7 ± 39.0	4.4 ± 35.4	**1.8** ± 56.7	−2.9 ± 18.9
Molecules	−68.3 ± 16.5	11.8 ± 15.7	**0.8** ± 3.9	1.0 ± 5.2

Table 5
GED approximation error in percentage. The best results are highlighted.

Database	BP $\mu \pm \sigma$	HED $\mu \pm \sigma$	BP-G $\mu \pm \sigma$	BP-G2 $\mu \pm \sigma$	BP2 $\mu \pm \sigma$
Letters I	0.8 ± 1.9	−18.2 ± 13.2	6.6 ± 8.5	27.6 ± 29.8	**0.4** ± 1.4
Letters II	11.9 ± 14.3	−45.2 ± 13.0	15.2 ± 15.4	13.3 ± 15.7	**10.8** ± 14.2
Letters III	8.5 ± 11.8	−28.9 ± 11.5	23.2 ± 24.3	13.4 ± 16.0	**5.0** ± 8.2

By combining the HED and BP-Greedy, The BP2 method has been introduced to be the alternative to the previous cubic-time method BP and, surprisingly, has even outperformed BP in an empirical evaluation on the IAM graph database.

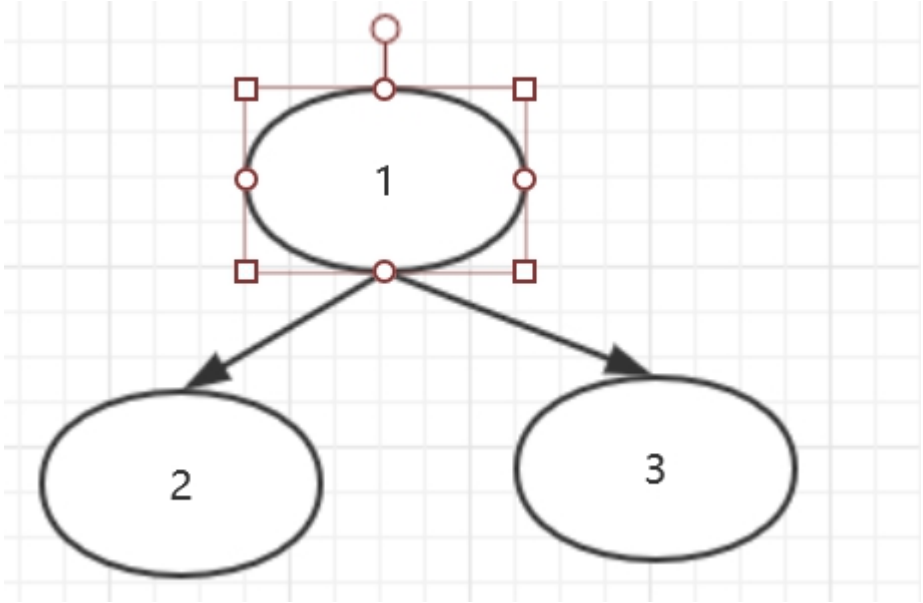
So I decided to use bipartite_graph_matching_2(BP2) in GMatch4py.

1.2 make comparison between quadratic time algorithm and cubic time algorithm

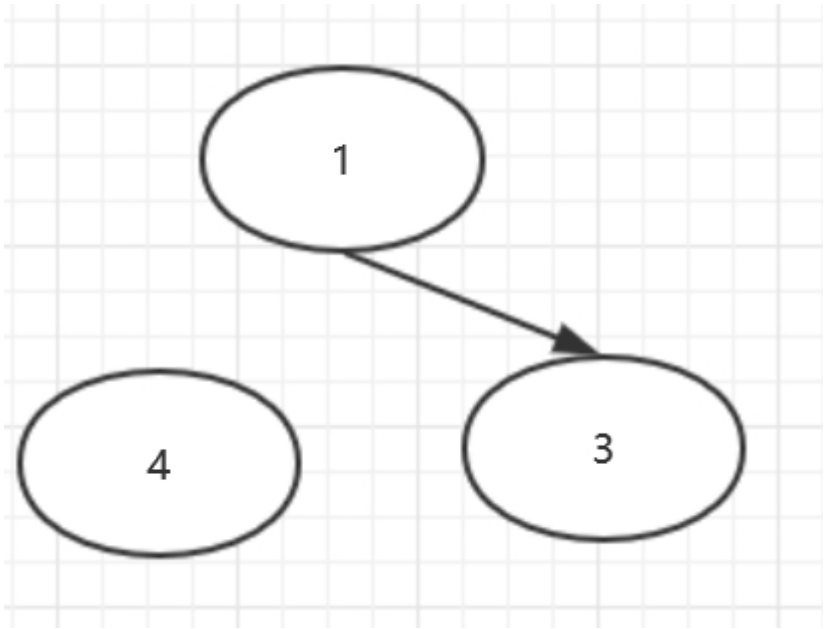
To verify the efficency of the quadratic time algorithm, I use cubic time algorithm to compare with the BP2 method. I write a python program called Comparison.py to evaluate the efficency of thse two algorithm. Nevertheless, the result is shocking. Using the python moudule GMath4py, the BP2 does not show better perform than cubic algorithm.

And There are some Quadratic-time approximation of Edit Distance like(Hausdorff Edit Distance, Greedy Edit Distance, BP2), and you can find the implement of these algorithm in open source project called GMatch4py. And the implement of Cubic-time approximation of Edit Distance can be found in a python module called networkx. And I tried this methods and found though the Quadratic-time approximation of Edit Distance can be faster than the Cubic-time one, the accuracy of Quadratic-time methods is bad. For example, I used a small graph to experiment the accuracy of these method.

The structure of the two graph is like:



and



The standard edit distance of these two graphs is 2, because you can change the graph one to graph two by deleting the edge(1,2) between node1 and node2 then relabeling the node 2 to node4.

So the cost is 2 including a delete operation and relabel operation.

And the cubic-time approximation of edit distance method's output is 2.

But the Quadratic-time methods' output is mostly wrong. For example, the output of BP2 method is 4 and the output of HED method is 1.

1.3 compile .c file from the real data set

To generate the real graphs from the real data set, I choose data set 'iterate' as the real data set from the Adelson's Github. First, I use python to write a program to iterate the data set 'problems' and I got 68 c code files from many students. Then, I compile these .c file with the comman call in Python and compile them to .bc file.

1.4 generate graphs from these .bc file

I run these .bc file and transer them to .dot file in their dirs.

c

1.5 write document about the algorithm I used to Neil and Angello.

I write a document to illustrate the algorithms I used to Neil and Angello:

The exact Graph Edit Distance is NP-complete problem, so it may cost a lot of time. So there are many methods to compute the approximated edit distance including (Hausdorff Edit Distance, Greedy Edit Distance, BP2, etc)

Standard Edit Distance algorithm:

We consider arbitrarily structured graphs with unconstrained labeling functions. Formally, a *graph* is a four-tuple $g = (V, E, \mu, \nu)$ where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, μ: $V \to L_V$ is the node labeling function, and ν: $E \to L_E$ is the edge labeling function. L_V and L_E are finite or infinite label sets for nodes and edges, respectively (numerical and/or symbolic labels are allowed).

Given graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$, the idea of graph edit distance [6,30] is to transform g_1 into g_2 using basic *edit operations*. For nodes $u \in V_1$ and $v \in V_2$, we denote *substitutions* with $(u \to v)$, *deletions* with $(u \to \epsilon)$, and *insertions* with $(\epsilon \to v)$, where ε refers to the empty "node". For edges, we use a similar notation.

An *edit path* $\lambda(g_1, g_2) = (e_1, \ldots, e_k)$ between g_1 and g_2 is a sequence of edit operations that transform g_1 into g_2. Note that only node edit operations have to be specified in the edit path. They imply the corresponding edge edit operations.

Finally, the *graph edit distance* $d_{\lambda_{\min}}(g_1, g_2)$ is defined as the minimum cost of all edit paths. Formally,

$$d_{\lambda_{\min}}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i) \tag{1}$$

where $\Upsilon(g_1, g_2)$ is the set of all edit paths and $c(e_i)$ is a non-negative edit operation cost.

Optimal algorithms for computing the edit distance $d_{\lambda_{\min}}(g_1, g_2)$ are typically based on combinatorial search procedures (such as A* search techniques). These procedures explore

the space of all possible mappings of the nodes and edges of g_1 to the nodes and edges of g_2 (i.e. the search space corresponds to $\Upsilon(g_1, g_2)$). Yet, considering m nodes in g_1 and n nodes in g_2, the set of possible edit paths $\Upsilon(g_1, g_2)$ contains $O(m^n)$ edit paths. Therefore, exact edit distance computation is exponential in the number of nodes of the involved graphs.

Approximation of graph edit distance(such as BP)(Cubic-time method):

With respect to the node sets $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$ of g_1 and g_2, respectively, the procedure computes an $(n+m) \times (n+m)$ cost matrix $\mathbf{C} = (c_{ij})$

$$\mathbf{C} = \left[\begin{array}{cccc|cccc} c_{11} & c_{12} & \cdots & c_{1m} & c_{1\varepsilon} & \infty & \cdots & \infty \\ c_{21} & c_{22} & \cdots & c_{2m} & \infty & c_{2\varepsilon} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\ c_{n1} & c_{n2} & \cdots & c_{nm} & \infty & \cdots & \infty & c_{n\varepsilon} \\ \hline c_{\varepsilon 1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \infty & c_{\varepsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \infty & \cdots & \infty & c_{\varepsilon m} & 0 & \cdots & 0 & 0 \end{array}\right] \tag{2}$$

where c_{ij} denotes the cost of a node substitution $(u_i \rightarrow v_j)$, $c_{i\varepsilon}$ denotes the cost of a node deletion $(u_i \rightarrow \varepsilon)$, and $c_{\varepsilon j}$ denotes the cost of a node insertion $(\varepsilon \rightarrow v_j)$.

By solving an LSAP based on this cost matrix, an optimal permutation $(\varphi_1, \ldots, \varphi_{n+m})$ of the integers $(1, 2, \ldots, (n+m))$ is found that minimizes the overall assignment cost $\sum_{i=1}^{(n+m)} c_{i\varphi_i}$. The LSAP solution corresponds to the assignment

$$\psi = ((u_1 \rightarrow v_{\varphi_1}), (u_2 \rightarrow v_{\varphi_2}), \ldots, (u_{m+n} \rightarrow v_{\varphi_{m+n}})) \tag{3}$$

of the nodes of g_1 to the nodes of g_2. It includes assignments of the form $(u_i \to v_j)$, $(u_i \to \varepsilon)$, $(\varepsilon \to v_j)$, and $(\varepsilon \to \varepsilon)$ (the latter can be dismissed, of course).

The node assignment ψ can be interpreted as an admissible edit path from Y(g_1, g_2). That is, the global edge structures from g_1 and g_2 can be edited with respect to the node operations captured in ψ. Eventually, the total cost of all edit operations (applied on both nodes and edges) can be interpreted as a distance $d_{\langle\psi\rangle}(g_1, g_2)$ between graphs g_1 and g_2. This edit path is admissible but not necessarily optimal and thus the distance $d_{\langle\psi\rangle}$ is – in the best case – equal to, or – in general – larger than the exact graph edit distance $d_{\lambda_{\min}}$.

In order to reduce the approximation error of $d_{\langle\psi\rangle}$, not only the node costs are included in the matrix entry c_{ij} but also estimated edge costs. They are computed with another LSAP between the set of edges $P = \{p_1, \ldots, p_k\}$ adjacent to u_i and the set of edges $Q = \{q_1, \ldots, q_l\}$ adjacent to v_j. The corresponding square cost matrix has $(k + l)$ rows and columns in analogy to Eq. (2). Consequently, to find an optimal node assignment ψ we solve $(n \cdot m + 1)$ LSAPs in total, first the $n \cdot m$ LSAPs for the edge assignments and then the final LSAP for the node assignment based on **C**. The latter typically dominates the computational complexity when considering a large number of nodes and a constrained edge degree.

1.6 generate graphs from real data set and compare them with each other

I use python to write a program to compare these graphs with each other.

It costs 2277.6217632792163 s to calculate 4624 pairs of graphs, about 0.5 s per pair. And I got a 68*68 matrix containing the edit distance of these graphs. data[i][j] represent the edit distance of i th graph to j graph is data[i][j]

I also ask Neil to create a user in the remote server(asedl.cs.illinois.edu), so I can run the grogram in the remote environment, which need a long time to compare a huge data set.

2 Solutions to problems last week

2.1 Which improved Edit Distance methods is the best?

The cubic time method is better than quadratic time methods, though the quadratic method can save time cost. And we need more accuracy rather than less time cost, because the grogram we use is sometimes really simple. So I choose the quadratic time methods after comparing the two mthods.

2.2 Whether should we take the label of nodes into consideration when we compare the similarity of graphs?

Currently, I don not put label into consideration, when I calculate the edit distance between two graphs.

2.3 How to combine the similarity of CFG,DD,CD into one similarity? How to choose the weight of each graph?

These problems need further discussion.

2.4 Are there any other factors taking into consideration to help to improve the accuracy of similarity?

Currently, I only consider the structure of graph, regradless of the node label and so on.

3 problems

3.1 How to combine the similarity of CFG,DD,CD into one similarity? How to choose the weight of each graph?

3.2 Whether we need to improve the time cost and whether we need to do n*n times comparison?