Data Structure (CS206) Project 4

An Algorithm Measuring Planar Graph Similarity for Recognition of Multi-font Chinese Characters

20050010 강성훈 (Seonghoon Kang)

20050145 김준기 (Joon-gi Kim)

20050546 차호정 (Hojeong Cha)

Leading professor : Otfried Cheong

The given problem

The goal of this project is proposing or implementing an algorithm for recognition of multi-font Chinese characters represented in planar undirected graph with coordinates and edge information. (It is allowed to use other people's idea, but implementation should be done by ourselves)

The given conditional information is numeric values, at the integer range of [0, 255], of each vertex and edges which connects vertices. Also, 4 thousands of sample graph of Chinese characters, the first 2,000 graphs are sorted by same characters with different 4 fonts, while the rest are shuffled.

Our approach to the problem

We first searched existing papers in the field of pattern recognition. Most of recent papers proposed methods based on statistical approach. After surveying thesis papers, we concluded that it is better to make simple algorithms by our hands, because many of them includes methods difficult to implement in our level, in our short time. Of course, our idea is affected by these papers.

The followings are the papers we searched:

- Ewa Kubicka, Grzegorz Kubicki, Ignatios Vakalis (1990), 'Using graph distance in object recognition', ACM Conference on Computer Science, ACM Press
- Chen-Tsun Chuang, Lin Yu Tseng (1995), 'A Stroke Extraction Method for Multifont Chinese Characters Based on the Reduced Special Interval Graph', IEEE Transactions on Systems, Man, and Cybernetics, Vol. 25, No. 7, pp.1171-1178
- Linda G. Shapiro (1978), 'Data Structures for Picture Processing', International Conference on Computer Graphics and Interactive Techniques, pp.140-146
- Osamu Fujimura, Ryohei Kagaya (1969), 'Structural patterns of Chinese characters', International Conference On Computational Linguistics, pp.1-17

Through free brain-storming with team members, 3 algorithms are proposed by each member. The details of them are explained below.

Proposed algorithms in our team

1st : Measuring distances between center points of each edges

Find the center points of all edges (let's call them 'edge points') in the source graph and the destination graph, and the center of the two graphs. Adjust the coordinates of edge points with the center coordinates. Then repeating with all the edge points, find the shortest distance to edge points of the destination graph. Finally, calculate the average of those distances and sum it and the result of the same process with the reversed the source and the destination.

This method can avoid some special cases (such as graph 31, 32) that the same edge is the part of different subgraphs (shown in the figure 1).

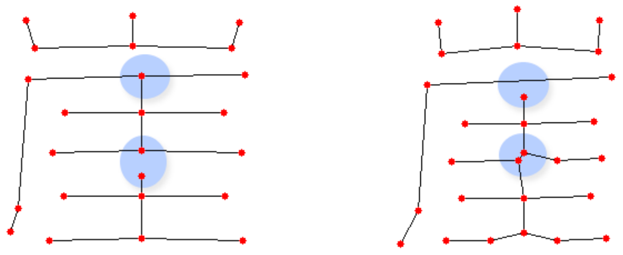


Figure 1. Different composition of same edges

2nd : Recursive divide and conquer

If there is an algorithm which gives the similarity of 2 graphs or subgraphs, suppose it's name A. Execute A on the whole character and take 1/2 of its result. And, divide the character graph into 2 parts. For the number of cases is 2 (top-bottom, left-right), execute the algorithm A on each 4 subgraphs, and take 1/8 of each result. Finally, sum up all the values, to get the 'whole' calculation. $(1/2+4\times 1/8=1)$

This proposal is based on an assumption that if the two graph is same character with different fonts then the sub-part of them is similar to each other.

3rd : Shortest distance from the source vertex to the destination edge

For all vertices of the source graph, find the shortest distance between the source's vertex and the destination's edge (orthogonal projection onto the nearest edge line segment). Sum up them and do same process on reversed source and destination. Fuse these two results and take average of them.

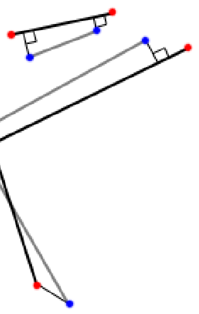


Figure 2. Shortest distance from source's vertex to destination's edge

This method uses the given information (vertex and edges) fully.

Furthermore…

- How about using the feature of Chinese characters, which consist of the basic regular components? But there are too many exceptional cases the subgraph of different components are fused/separated to others. (It seems to be a good idea, but unfortunately we have not enough time to implement them)

- Rendering vertex-edge graphs into a bitmap can give us many other methods, but this idea is refused because it may be out of the objectives of this problem. (The given graphs might be made from bitmap data)

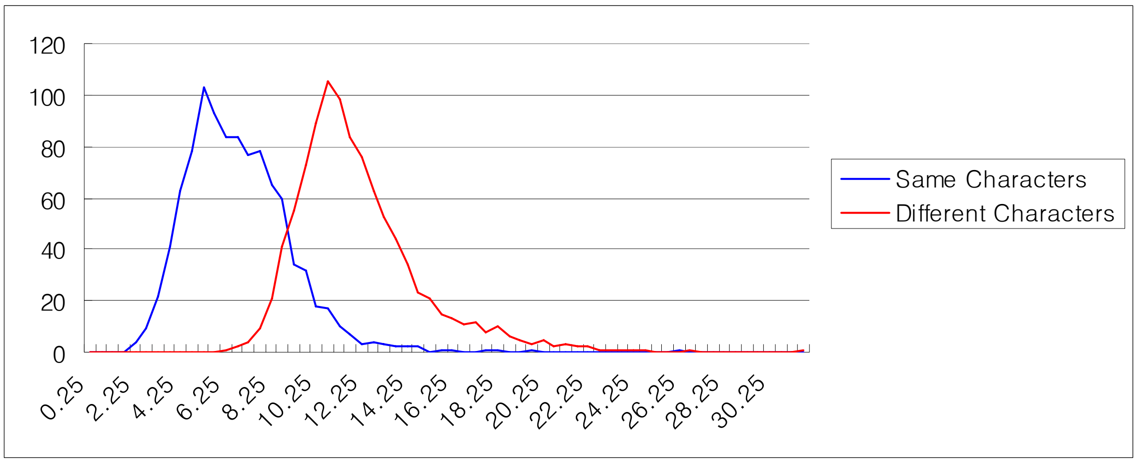
Final selection and code

We finally selected the 3rd algorithm. We tested for the first and third algorithm by Python prototype programs. The reason we selected this is that the common range of this algorithm between maximum of similarity of same characters and minimum of different characters is smaller than the first algorithm. The next is psuedo-code for this algorithm:

```
function adjustToCenter(graph)
   cx := Sum of the x-coordinates of all vertices / # of vertices
   cy := Sum of the y-coordinates of all vertices / # of vertices
   for v in vertices
     v := (v.x - cx, v.y - cy)

function split(graph, factor)
  for e in edges
    for i from 1 to factor
      split e into factor number of fragments

function _similarity(src, dest)
  dists := []
  for v in src.vertices
    for e in dest.edges
      make orthogonal projection from v to e, restricted in line segment
      get the distance between v, e
    dists.append(minimum of distances)
  return dists

function similarity(src, dest)
  adjustToCenter(src), adjustToCenter(dest)
  split(src, 3), split(dest, 3)
  dists := []
  dists.append(_similarity(src, dest))
  dists.append(_similarity(dest, src))
  return sum(dists) / len(dists)
```

Result of experiment & Conclusion

We have tested our algorithm on about 1,000 sample graphs, and got the following result. The horizontal axis of the graph is the value of similarity, and the vertical axis means the frequency of each similarity value range. The yellow line is comparisons between different characters while the pink line is comparisons between same characters. The similarity range seems to be well separated.



If the similarity is under 7, the two characters are almost surely same. If the similarity is over 8, the possibility that the two characters are different gets increasing.

Discussion

If we have invested more time on this project, we could make more interesting ideas and test them. This is the most unfortunate situation.

However, we could make a unique idea of measuring similarity, though it is not perfect. This projects gave us many interesting experience.