Articulation Point Detection

Michael Verdicchio
Arizona State University
07 December 2007
CSE 550 Semester Project
Goran Konjevod, Instructor

Abstract

Articulation points, or cut-vertices, are vertices in an undirected graph which when deleted disconnect the graph. In other words, they separate the biconnected components of a graph. Their detection is of interest in the context of disrupting network communication, for example, in a gene regulatory network. If a drug is to target a particular gene or gene product, it is tempting to focus on those with maximal degree in the network of interest, but perhaps the genes holding the whole network together, the articulation points, deserve attention as well. This semester project serves to 1) implement an efficient articulation point detection algorithm, and 2) compare its performance against a trivial and inefficient articulation point detection algorithm. As would be expected, the $O(|V|+|E|)$ algorithm performed much better than the trivial $O(|V||E|)$.

Introduction

In an undirected graph G = (V, E), where V denotes the set of vertices (or nodes) and E the set of edges (or arcs), a biconnected component is a subgraph of G in which cannot be separated by deleting any single vertex. For the entire graph G, the articulation points (or cut-vertices) are vertices which, if removed, disconnect the graph. These articulation points are of interest as they represent volatile points in the network communication structure.

Finding Articulation Point Detection Algorithms

There are many algorithms that detect articulation points with various efficiencies (1). A literature survey(2, 3, 4, 5, 6, 7) began with those references from (1) and began looking for algorithms that were designed more with implementation in mind rather than complex, yet minor improvements in efficiency. Eventually a variation of Tarjan's LOWPOINT method (8) demonstrated by Harold Gabow in (9) was chosen. It's running time is listed at $O(|V|+|E|)$, and for comparison's sake, its performance was contrasted with a trivial $O(|V||E|)$ algorithm described crudely on Wikipedia (10)).

Gabow's Algorithm

Gabow's algorithm is a recursive design using a stack and depth-first search (DFS). Basically, it numbers the vertices as they are seen in a DFS traversal and maintains a stack of possible articulation points. The algorithm is reproduced from the original paper here:

procedure ART(G)
1. empty stack S;
2. **for** $v \in V$ **do** $C[v] = I[v] = 0$;
3. $i = 0$;
4. **for** $v \in V$ **do if** $I[v] = 0$ **then** DFS$(v, 0)$;

procedure DFS(v, u)
1. **if** $u \neq 0$ **then** PUSH(u, S); increase i by 1; $I[v] = i$; /* add v to the end of P */
2. **for** edges $\{v, w\} \in E$ **do**
3.   **if** $I[w] = 0$ **then** DFS(w, v)
4.   **else while** $I[w] < I[\text{TOP}(S)]$ **do** POP(S);
5. **if** $u \neq 0$ **then** {
6.   increase $C[v]$ by 1;
7.   **if** $u = S[\text{TOP}(S)]$ **then** { increase $C[u]$ by 1; POP(S) } };

Figure 1: Gabow's Algorithm

Trivial O(|V||E|) Algorithm

Strictly for the sake of comparison with Gabow, the following trivial algorithm was implemented. Its correctness comes from the fact that the removal of an articulation point will disconnect a portion of the graph. Thus, after determining the number of components, delete vertices one at a time and count the number of components after the deletion. If it increases, the deleted vertex is an articulation point.

procedure ART_TRIVIAL(G)
1. a = number of components in G (found using DFS/BFS)
2. **for** each i in V with incident edges **do**
3.   Remove i from V
4.   b = number of components in G with i removed
5.   **if** b > a **then**
6.   i is a cut vertex
7.   restore i

Figure 2: Trivial Algorithm

Implementation

Both algorithms were implemented using the Java programming language and embedded into a usable GUI. The GUI operates in two main modes, namely "interactive", and "diagnostic". In "interactive" mode, the user is presented with a blank canvas where he or she may build a graph by adding vertices with a left-click, and then connect them by dragging a line from one to another. All graph layout and rendering is done by JUNG, the Java Universal Network/Graph Framework, and Open Source project from UC Irvine (11). Once the user is done creating the graph, the articulation points can be located with the button. A user can also choose to upload a graph that has been saved in the NEATO graph format (12). This is a very rudimentary format that can be created on the fly, and an example NEATO file is given below. Finally, a user has a choice to generate a random graph to use for detecting articulation points. Each choice opens a new graph canvas which operates independently of the others. While editing mode is still available whether one has loaded a NEATO file, generated a random graph, or not, one should not attempt to augment a loaded or generated graph. Only build graphs from scratch.

```
graph G{
   1--2;
   1--3;
   2--3;
   3--4;
   3--5;
   4--5;
}
```

Figure 3: Sample NEATO file

In "diagnostic" mode, a user specifies a text file to save the output, and then enters a number of iterations for each of 12 randomly generated, increasingly large graphs. The software then compares Gabow's algorithm against the trivial version. As the simulation runs, a text output window alerts the progress, and once completed, the results are easily viewed in the text file.

Accessing the Software

The main executable JAR file, source code, and sample graphs can all be accessed at http://www.public.asu.edu/~mverdic/articulation.

Diagnostics

For this report, the software was run on diagnostic mode with 60 iterations over each of 12 randomly generated graphs of increasing size. For each of the 12 trials, statistics were kept on each algorithm's execution time (in milliseconds). After the 60 iterations, the iteration-by-iteration execution times were compared and wins, losses and ties were reported. The average times over all 60 iterations were also compared. For full details, please see the appendix for the diagnostic output file. A chart showing the average execution times across the 12 increasingly large graphs is shown in the figure below.

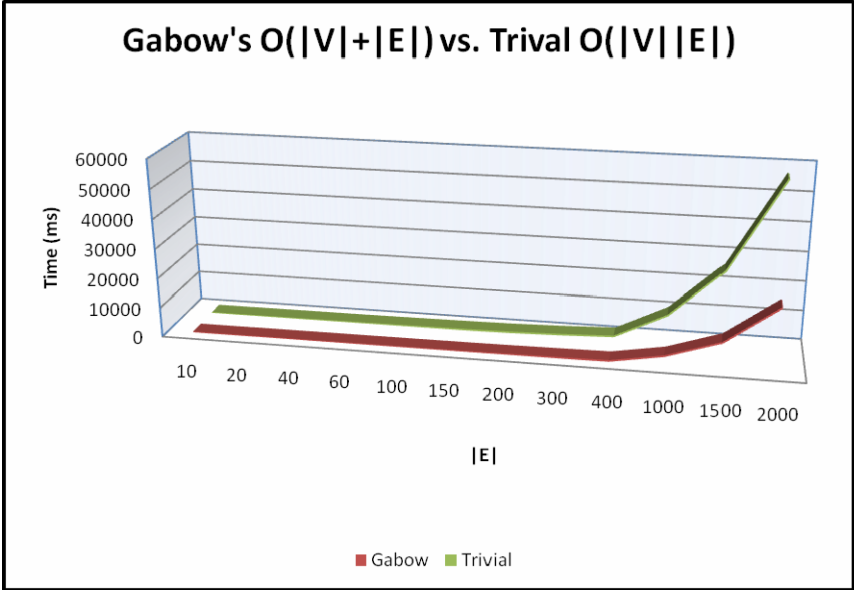

Figure 4: Average Execution Times

Conclusions

Obviously, these diagnostic results are not surprising. However, with the goals being the understanding of the topic, implementation, usability and future expansion, this project was highly successful.

References

1. Schrijver,A. and Schrijver,A. (2003) Combinatorial Optimization. Springer-Verlag.

2. Cheriyan,J. and Thurimella,R. (1991) Algorithms for parallel k-vertex connectivity and sparse certificates. *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing,* 391.

3. Even,S. (1973) An algorithm for determining whether the connectivity of a graph is at least k. *SIAM J. Comput*.

4. Even,S. and Tarjan,R.E. (1975) Network flow and testing graph connectivity. *SIAM Journal on Computing,* **4,** 507.

5. Gabow,H. (2006) Using expander graphs to find vertex connectivity. *Journal of the ACM,* **53,** 800.

6. Henzinger,M., Rao,S. and Gabow,H. (2000) Computing vertex connectivity: New bounds from old techniques. *Journal of Algorithms,* **34,** 222.

7. Tarjan,R.E. and Vishkin,U. (1984) Finding biconnected componemts and computing tree functions in logarithmic parallel time. *Foundations of Computer Science, 1984. 25th Annual Symposium on,* .

8. Tarjan,R.E. (1972) Depth-first search and linear graph algorithms. *SIAM Journal on Computing,* **1,** 146.

9. Gabow,H. (2000) Path-based depth-first search for strong and biconnected components. *Information Processing Letters,* **74,** 107.

10. Wikipedia contributors. Cut vertex. **2007**.

11. O'Madadhain,J., Fisher,D., White,S. and Boey,Y.B. (2003) The JUNG (java universal Network/Graph) framework. **Technical Report UCI-ICS 03-17**.

12. North,S.C. (1992) NEATO user's guide. *AT&T Bell Laboratories*.

Appendices

Following this page in the hard copy of this report is 1) the diagnostic report followed by 2) Java source code for the algorithm-implementing class. The rest of the source code can be downloaded from the aforementioned website.