Computer Science 373 – Analysis of Algorithms
Prof. Steven Skiena
Fall 2018

Homework 4 – Combinatorial Search
Due Thursday, November 8, 2018

The real reason we study algorithms is to find fast ways for solving problems. Analysis gives us a way of seeing how well we are doing, but it washes away questions about constants, difficulty of implementation, and performance in practice. For this assignment, you will be given an algorithmic problem requiring some form of combinatorial search, and your goal is to design and implement as fast a solution as possible. This is intended to be a competition - may the fastest program win! Modest prizes will be given for the fastest and slowest entries.

The *bandwidth problem* takes as input a graph G, with n vertices and m edges (ie. pairs of vertices). The goal is to find a permutation of the vertices on the line which minimizes the maximum length of any edge. This is better understood through an example. Suppose G consists of 5 vertices and the edges (v_1, v_2), (v_1, v_3), (v_1, v_4), and (v_1, v_5). We must map each vertex to a distinct number between 1 and n. Under the mapping $v_i \rightarrow i$, the last edge spans a distance of four (ie. 5-1). However, the permutation $\{2, 3, 1, 4, 5\}$ is better, for none of the edges spans a distance of more than two. In fact, it is a permutation which minimizes the maximum length of the edges in G, as is any permutation where 1 is in the center.

The bandwidth problem has a variety of applications, including optimizing memory usage in hypertext documents. What is known about it? The problem is NP-complete, meaning that it is *exceedingly* unlikely that you will be able to find an algorithm with polynomial worst-case running time. It remains NP-complete even for restricted classes of trees. However, since the goal of the problem is to find a permutation, a backtracking program which iterates through all the $n!$ possible permutations and computes the length of the longest edge for each gives an easy $O(n! \cdot m)$ algorithm. But the goal of this assignment is to find as practically good an algorithm as possible.

Rules of the Game

1. Everyone must do this assignment separately. Just this once, you are not allowed to work with your partner. The idea is to think about the problem from scratch.
2. If you do not completely understand what the bandwidth of a graph is, you don't have the *slightest* chance of producing a working program. *Don't be afraid to ask for a clarification or explanation!!!!!*
3. There will be a variety of different data files of different sizes. In a combinatorially explosive problem such as this, adding one to the problem size can multiply the running time by n, so test on the smaller files first. Do not be afraid to create your own test files to help debug your program.
4. The data files are available via the course WWW page. Each file has a name like "g-X-10-20", meaning that the file contains a graph of special type X, with 10 vertices and 20 edges. (The meaning of the type is irrelevant for your program, just that I might create a variety of different types of graphs of the same size). The first line of each file will contain the number of vertices, and the second line the number of edges. Each subsequent line contains a pair of vertices comprising an edge. For example, the star described above would be given:

```
5
4
1 2
1 3
1 4
1 5
```

Your program must output a permutation which minimizes the bandwidth, and what that bandwidth is.

5. Writing efficient programs is somewhat of an iterative process. Build your first solution so you can throw it away, and start it early enough so you can go through several iterations.
6. You will be graded on how fast and clever your program is, not on style. But please make it readable enough so we have a fighting chance to understand it. Incorrect programs will receive *no credit*. A program will be deemed incorrect if it does not find a permutation of smallest bandwidth for some graph.
7. If you feel your programming background is somewhat weak, be sure that you use a simple backtracking approach. Don't get fancy until you have something that works. As Clint Eastwood says, "a man's got to know his limitations." It should be possible to get a working program which is under 100 lines long.
8. You may use any programming language if you have a preference, but if you don't please use C for uniformity/efficiency. See my backtracking codes posted at `http://www.cs.stonybrook.edu/~skiena/392/programs/` as a starting point.
9. The programs are to run on the whatever computer you have access to, although it must be vanilla enough that I can run the program on something I have access to.
10. Don't forget to use the code optimizer on your compiler when you make the final run, to get a free 20% or so speedup. The system profiler tool, *gprof*, will help you tune your program.
11. You are to turn in a listing of your program, along with a brief description of your algorithm and any interesting optimizations, sample runs, and the time it takes on sample data files. Report the largest test file your program could handle in one minute or less of wall clock time.
12. The top five self-reported times / largest sizes will be collected and tested by me to determine the winner.
13. What kind of approaches might you consider? Instead of testing every permutation, you should be able to develop a backtracking algorithm which prunes partial solutions, ie. there will be no way to complete a given layout of k vertices which could reduce the cost of our best solution to date. There is room for cleverness in minimizing the time to compute the bandwidth for a particular permutation or partial permutation. Starting off with a good approximate solution may help achieve faster cutoffs. The choice of data structure for representing your graph and partial solution will be critical, should you use matrices or linked lists? Decide whether such paradigms as divide-and-conquer, dynamic programming, or randomized algorithms can be helpful. Can you devise any low-level hacks to further speed things up? Good luck!
14. Note that daily problems continue through this period. It is very important to do these to make sure you are keeping up with the material you will later get more problems on.