

Take home Final

Instructions:

You may consult your books, notes and the web. However, you are not to discuss these problems with anyone except me. You may send answers by email or put in HW box Please write clearly and succinctly; be sure to give an overview before plunging into details. Note that if an algorithm is requested, the most efficient one is desired. Open Book and Notes.

Your name:

On problem	you got	out of
1		30
2		10
3		20
4		24
5		16
Σ		100

1 Vertex Cover

[30 points]

In problem set 4 we considered various heuristics for random graphs. When the graphs were dense (e.g. $p = 70$) all the covers found were quite large (over 99% of the vertices). Here we explore getting some lower bounds to show this is not far from the optimal cover. It is obvious that if we exclude a vertex v from the cover, we must use all of v 's neighbors in the cover. Thus the minimum degree vertex in the graph provides a lower bound on C^* the optimal cover (e.g. if all vertices have degree 500 or more, $|C^*| \geq 500$).

More generally, for any set $S \subset V$, to **exclude** S from the cover there must be no edges connecting two vertices in S and the cover must **include** all vertices u with an edge touching something in S (i.e. an edge $((u, v), v \in S)$). Thus we can get a lower bound on C^* by considering the minimum number of included vertices over all sets S of a small size k . If we want a cover of size less than $N - k$, we need to use at least the number of required vertices for the best excluded set of size k .

(a) Your task is to get a good lower bound for random graphs of size 1000 with $p = 70$. You should consider all sets S of size 3 and, if its legal to exclude S from a cover, compute the number of vertices with arcs touching S (which must be included in a cover). Your lower bound is the minimum size set of vertices which must be included over all legal excluded sets of size 3 (e.g. if each such set requires including at least 900 vertices, you know that $|C^*| \geq 900$).

Specifically, you should write a program which takes a random graph generated by the program used in PS2,4 and calculates a good lower bound on its min-cover size using the method above. Warning: it will pay to think carefully before coding. Also, you may find it useful to make the diagonal elements zero in the A matrix (they are currently unset).

If you can't do this, partial credit for considering all sets of size 2. Extra credit for doing all sets of size 4 (or even 5?)

(b) Try to optimize your program so it works quickly (this would be important in a branch-bound solution using this). Describe your approach and improvements at a high level. Report the lower bound you found and give the run time for you program (my not super optimized version takes under a minute for sets of size 3, under two seconds for size 2 sets). Email me your final code.

2 Hard/Easy problems

[10 points]

i) Let B be an NP-hard problem and C be an NP-complete problem. Does a polynomial time algorithm for B give you a polynomial time algorithm for C ? Justify your answer.

3 Search timing**[20 points]**

Suppose we have a sorted array $A[1..n]$. We can use either binary search or sequential search to look for an element T in the array (and report its position, or zero if its not in the array). For small arrays sequential search will be faster. Determine for what values of n sequential search will be faster than binary search. You may either do this via estimation or actual timing tests.

Your answer will depend on several factors: How often the item T is in the array, when T is in the array where it is, whether the array A is in cache. Discuss how each of these affects your answer. As a baseline, (for experiments if done), you may assume that each item in the array is equally likely to be looked up, and T will not be in the array 10 % of the time. For experiments/analysis use good implementations of each strategy, but not the super optimized binary search given in the text in 9.3.

4 Space**[24 points]**

For each of the following assume you are storing items of the following type, and your goal is to store them using the least space possible (even if this increases the access time) assume the operation you need to support is LIST: which lists all the data items (in an arbitrary order). Describe you method and estimate the total number of bytes used.

- i) 100,000 Dates: month, day, year (you may assume no year is larger than 9999 or negative)
- ii) 100,000 Social Security Numbers
- iii) An undirected graph with no edge weights, 10^6 vertices and 10^{11} edges.

5 Local Search**[16 points]**

The traveling-salesman problem (TSP) consists of a complete directed graph with edge weights. We want the best tour, among the $(n - 1)!$ possible tours. Here we examine using local search for TSP. We can start with an arbitrary tour τ (just a random permutation of the vertices). A neighbor is a new permutation which is similar to our old one.

- i) Describe a method to convert the current tour τ to a new tour τ' such that the two tours differ in only $O(1)$ edges (a picture may be a useful part of your description).
- ii) Why is it helpful to have the two permutations differ by a small number of edges?
- iii) describe how to pick a random neighbor using your strategy of part i)