Hw 5 - CS 120

Evan Jolley

October 2022

1

1.1

Code

1.2

Code

1.3

Code

1.4

Timeout length: 20 seconds

	lowest n failed	largest n passed
Exhaustive	30	20
ISET/BFS	20	80

In terms of big-O, our BFS/ISET algorithm runs in time $O(1.89^n)$ as discussed in lecture and given in our pset. As for Exhaustive-Search, in our worst case we have to verify the coloring across m edges for every color combination (3^n total). Thus, the ES algorithm runs in time $O(m * 3^n)$. Clearly this is slower than BFS/ISET, and this explains the large discrepancy in the "largest n passed" column.

I find it interesting that the lowest n failed is larger for ES, but I believe this can be explained as well. ES simply brute forces a solution, so if the time allotted is large enough to iterate through all possibilities, ES will always succeed. However, ISET/BFS is more dependent on ordering, the probabilities in our test cases, etc, and thus ended up failing that given $n = 20$ example.

2

2.1

In this problem, we are trying to construct a reduction from $IndependentSet-ThresholdDecision$ to $IndependentSet-OptimizationSearch$ to solve $IS-TD$ in time $O(T(n,m))$. Let's go through it step by step, analyzing runtime as we go.

Pre-process:
There is no pre-processing to do, as $IS-TD$ takes G and k as inputs and $IS-OS$ takes just G. There is nothing we need to manipulate G to prepare our original input for our oracle.

Oracle:
We run $IS-OS$ on our input G and return the largest independent set within it. As defined in the problem, this will run in time $T(n,m)$.

Post-process:
After running our oracle, we can simply check if the outputted independent set had length greater than or equal to k. If it is we return YES, and if not we return NO. This will run in constant time, as there is only one comparison to check, no matter the size of any input to our algorithm.

Conclusion:
In total, the algorithm will run in time $O(T(n,m)+1) = O(T(n,m))$ which is the desired runtime. We know this algorithm is correct because:

- $IS-OS$ is correct and will always output the largest possible independent set in the input graph.
- If the length of that set is greater than or equal to k, there exists one or more isets in G with length greater than or equal to k.

2.2

In this problem, we are trying to construct a reduction from $IS-OS$ to $IS-TS$ to solve $IS-OS$ in time $O((logn)T(n,m))$. Let's go through it step by step, analyzing runtime as we go.

Pre-process:
There is no pre-processing to do, as $IS-TS$ takes G and k as inputs and $IS-OS$ takes just G. There is nothing we need to manipulate G to prepare our original input for our oracle.

Oracle:
Our method for finding the largest independent set will employ binary search which runs in time $O(logn)$ (thank you Prof. Malan and your phone books).

First, we call $IS - TS$ using our inputted G and $k = n/2$. If an independent set of that size exists, we will again run $IS - TS$ using G and $k = 3n/4$. If one does not exist, we will run $IS - TS$ using G and $k = n/4$. We will continue this method, following Binary Search principles, until we find the k and $k+1$ where k results in an independent set and $k+1$ does not in $IS - TS$. Thus, that k value will be the length of the largest independent set possible in G.

We know that binary search runs in time $O(logn)$, so finding the largest possible k value will will run in time $O(logn)$ at every step of this process we must run $IS - TS$ which is defined in the problem to run in time $O(T(n, m)$. Thus, our oracle runs in time $O(logn * T(n, m))$.

Post-process:
We don't need to do anything in post-process, as the independent set that our oracle outputs will be the largest possible in G. This is exactly what $IS - OS$ outputs.

Conclusion:
In total the algorithm will run in time $O(logn * T(n, m))$ which is the desired runtime. We know this algorithm is correct because:

- $OS - TS$ is correct and will always output an independet set of size k if one exists.
- Binary search is always correct, and we will always find the largest possible k value in time $O(logn)$.

2.3

In this problem, we are trying to construct a reduction from $IS-TS$ to $IS-TD$ to solve $IS - TS$ in time $O(n * T(n, m))$. Let's go through it step by step, analyzing runtime as we go.

Pre-process:
In our pre-process, we will want to initialize an empty set which will come back later during our oracle calls. This can be created in constant time. No manipulation of our inputs is necessary.

Oracle:
First, we can simply run $IS - TD$ on our inputs. If "NO" is returned, then there is no independent set to return for $IS - TS$, and thus, there is no need to continue. If "YES" is outputted, however, we can use the process hinted at in the hint to solve this problem.

The hint states that "G has an independent set of size at least k containing vertex v iff $G - N(v)$ has an independent set of size at least $k - 1$, where

$G - N(v)$ denotes the graph obtained by removing v and all of its neighbors from G." The key realization we must have is that if v is in the independent set, we know that none of its neighbors are! Thus, if we remove v and all of its neighbors from G, the new independent set threshold will reduce by exactly 1.

The opposite is also true. If you have a graph with threshold k, and you add a group of j vertices with $j-1$ of these vertices connected to random parts of the graph and 1 connected only to the $j-1$ others, that single vertex can be added to the preexisting independent set. Thus, the new threshold would increment by 1.

We can iterate over all n vertices and do the following check:

1. Update the graph to remove the vertex and all of its neighbors.
2. Run $IS-TD$ using the new graph and $k-1$. If "YES" is returned, a) add the removed vertex to the set we initialized during pre-processing, b) re-save G as $G-N(v)$, and c) re-save k as $k-1$. If "NO" is returned, simply move to the next vertex with no reassignment. Each call to $IS-TD$ will run in time $O(T(n,m)$ and all re-assignment will run in constant time.

There are n vertices to iterate over, checking if they are in the independent set, meaning the total oracle runtime is $O(n * T(n,m))$.

Post-process:
In post we simply need to return the set we'd been populating across the course of our oracle calls. $IS-TS$ returns the set of vertices in the independent set of size at least k, and that is what we have collected in our set. Returning this set is clearly in constant time.

Conclusion:
In total the algorithm will run in time $O(1 + n * T(n,m) + 1) = O(n * T(n,m))$ which is the desired runtime. We know this algorithm is correct because:

- $IS-TD$ is known to be correct.
- We successfully reasoned that if a vertex v is in the independent set, removing it and all of its neighbors from the graph will lead to an independent set of size at least $k-1$, as no two vertices in an independent set are connected.
- Repeating this process until we have iterated over the whole graph will result in identifying and returning all vertices in the independent set.