CS 120: Intro to Algorithms and their Limitations

Problem Set 5

Due: October 25, 2023 11:59pm **Denny Cao**

Collaborators:
No. of late days used on previous psets: 2
No. of late days used after including this pset: 2

§1 Exponential-Time Coloring

(d) Within the time limit of 10 seconds:

- `Exhaustive-Search 3-Coloring`: For Line of Rings, no combination of parameters resulted in the algorithm completing within the time limit. For Randomized Cluster Connections, the largest instance it can solve is when there is a probability of keeping an edge of 0.8 with 2 clusters of size 10 (n = 20, m = 83). The smallest instance it cannot solve within a reasonable time is when there is a probability of keeping an edge of 0.2 with 3 clusters with a size of 10 (n = 30, m = 55).
- `ISET-BFS Coloring`: The largest instance it can solve is when there are 200 rings with a size of 4. (n = 800, m = 999). The smallest instance it cannot solve within a reasonable time limit is when there are 100 rings with a size of 3. (n = 300, m = 399). The largest instance it can solve is when there is a probability of keeping an edge of 0.8 with 2 clusters of size 18 (n = 36, m=248). The smallest instance it cannot solve within a reasonable time is when there is a probability of keeping an edge of 0.2 with 4 clusters with size 10 (n =- 40, m =123).

There is no instance where `Exhaustive-Search 3-Coloring` finishes but `ISEF-BFS Coloring` does not; either `ISEF-BFS Coloring` is the only of the two that finishes within the time limit, or both do. Thus, relatively, `ISEF-BFS Coloring` is faster than `Exhaustive-Search 3-Coloring`. This is makes sense, as `Exhaustive-Search 3-Coloring` takes time $O(m3^n)$, as there are 3^n ways to pick a color for each of the n vertices with m edges which must be checked to see if there are conflicting colors, whereas `ISEF-BFS Coloring` takes time $O(1.89^n)$ as shown from the SRE.

§2 Reductions Between Variants of IndependentSet

(a) *Proof.* We first describe the algorithm:

- Make an oracle call `IndependentSet-OptimizationSearch` on the input graph G. Let the independent set returned be S.
- Postprocess: Find size of S. Let this be n.
- If $n \geq k$, return `YES`. Otherwise, return `NO`.

We now want to prove our reduction algorithm:

1. has the desired runtime and
2. is correct.

- There is a single oracle call to `IndependentSet-OptimizationSearch` that runs at most $T(n,m)$ time. Finding the size of S of at most n elements takes time $O(n)$, Thus:

$$T_{\texttt{IndependentSet-ThresholdDecision}} \leq O(n) + T(n,m)$$

- The proof of correctness is that, if there exists an independent set of size at least k in G, then the size of the largest independent set returned by `IndependentSet-OptimizationSearch` will either be greater than or equal to k, as it is the largest independent set. Thus, returning `YES` is a correct solution to the `IndependentSet-ThresholdDecision` problem. If k is greater than the size of the independent set returned from `IndependentSet-OptimizationSearch`, then there does not exist an independent set of size at least k in G, as the set S returned by `IndependentSet-OptimizationSearch` is the largest independent set possible in G. In this case, the algorithm correctly returns `NO`. Thus, the reduction algorithm is correct.

We have shown that the reduction is correct and runs at $O(T(n,m))$. □

(b) *Proof.* We first describe the algorithm:

- Set `left` $= 1$ and `right` $= n$.
- While `left < right`:
  - Set `mid` $= \left\lceil \frac{\texttt{left}+\texttt{right}}{2} \right\rceil$.
  - Make an oracle call to `IndependentSet-ThresholdSearch(`G`, mid)`.
  - If the oracle does not return $\perp$, set `left` $=$ `mid`.
  - If the oracle returns $\perp$, set `right` $=$ `mid`.
- Return the set returned from the oracle.

We now want to prove our reduction algorithm:

1. has the desired runtime and
2. is correct.

- For each iteration of the loop, we execute basic operations as well as one oracle call to `IndependentSet-ThresholdSearch` which runs in time at most $T(n,m)$ and thus the runtime for each iteration is $O(1) + O(T(n,m)) = O(T(n,m))$. As we implement binary search, there are at most $\log n$ iterations, and thus the total run time is

$$T_{\texttt{IndependentSet-OptimizationSearch}} = O((\log n) \cdot (T(n,m)))$$

- We will prove that the reduction algorithm is correct. Initially, `left` $= 1$ and `right` $= n$, as the size of an independent set is at least 1 and cannot be greater than the amount of vertices in G. The algorithm then enters a loop that performs binary search for the largest independent set size. The loop invariant is that `left` $<$ `right`, and the loop will exit when `left` $=$ `right`. From the loop invariant, with each iteration, we make an oracle call to `IndependentSet-ThresholdSearch` with inputs G and `mid`, which is the average value in the current range from `left` to `right`. If the oracle returns a set, then it follows that there exists an independent set that is of size `mid` or greater, and thus we can shift our range to `mid` to `right`. If the oracle returns $\perp$, then it follows that there does not exist an independent set that is at least size `mid`, and thus we can shift our range to `left` to `mid`. We repeat until we exit the loop. At this point, there does not exist an independent set of size greater than `left` = `right`, or an independent set of size less than `left` = `right`, which means that the set obtained when `left` = `right` is the largest size for an independent set in G.

We have shown that the reduction algorithm is correct and runs at $O((\log n) \cdot T(n, m))$. $\square$

(c) *Proof.* We first describe the algorithm:

- Initialize a set S to store the independent set and set `remaining_size` $= k$, where k is the input threshold value for the algorithm.
- For each vertex v in the graph G:
  - If `remaining_size` $= 0$, return S.
  - Create a copy of the graph G' and remove v and its neighbors.
  - Make an oracle call to

    `IndependentSet-ThresholdDecision(`G`, remaining_size` $- 1$`)`.
  - If the oracle returns `YES`, add v to S and update `remaining_size` $=$ `remaining_size` $- 1$.
- If `remaining_size` $= 0$, then return S. Otherwise, return $\perp$.

We now want to prove our reduction algorithm:

1. has the desired runtime and
2. is correct.

- In the worst case, we iterate over all the vertices in G which takes time $O(n)$. For each vertex, we create a copy of the graph and remove its neighbors, which takes at most $O(n + m + m) = O(n + m)$. For each vertex, we make an oracle call to `IndependentSet-ThresholdDecision` which runs in $T(n, m)$ time. Thus, in total, the algorithm runs in time $O(n \cdot (m + T(n, m)))$, and as $T(n, m) > m$, it follows that

$$T_{\texttt{IndependentSet-ThresholdSearch}} = O(n \cdot (T(n, m)))$$

- We will first prove the biconditional statement 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 $k - 1$.

  ($\implies$) Suppose G has an independent set S of size k and let v be a vertex in S. Then, if we remove v along with the neighbors of v to obtain $G - N(v)$, the remaining vertices in $G - N(v)$ are not connected to v or each other, and thus they form an independent set. The size of this set is $k - 1$, as we removed a single vertex v and we retain the original independent set besides v, and thus there exists an independent set of size at least $k - 1$.

  ($\impliedby$) Suppose $G - N(v)$ has an independent set S of size at least $k - 1$. If we add the vertex v to the graph, then v will not be adjacent to any vertex in

S, and thus an independent set $S' = S \cup \{v\}$ will also be an independent set in G. The size of S' will be k, as $|S| = k - 1$ and we add an additional vertex v.

We will now use this to prove the correctness of our algorithm. If, in any iteration, the oracle returns `YES` for $G - N(v)$ with `remaining_size` $- 1$, there exists an independent set of size at least `remaining_size` $- 1$ in the graph after removing v. This, in turn, implies that there might exist an independent set of size at least `remaining_size` in the original graph G containing vertex v. We start at `remaining_size` $= k$ and decrement for each v that are in an independent set of size `remaining_size`, and thus if we reach `remaining_size` $= 0$, then all vertices we removed are part of an independent set of size at least k. Otherwise, we iterate until all vertices are exhausted and if `remaining_size` $\neq 0$, in which it would return $\perp$, there are not enough vertices available to form an independent set of size at least k.

We have shown that the reduction algorithm is correct and runs at $O(n \cdot (T(n, m)))$.
$\square$