CSEP521 : Applied Algorithms: Homework 4

Karuna Sagar Krishna

December 10, 2022

Question 1

Idea

Thanks to our class TA for clarifying the question as it asks us to solve a search problem and not a decision problem. The question asks us to provide an polynomial time algorithm to find a factor for given n bit number N, if $P = NP$.

To search for a factor, we would have to test all the integers $1 < a < \sqrt{N}$. This is not polynomial in terms of the length of the input given to us i.e. n bits. As discussed in class, problem classes P and NP contain decision problems. So, the question hints at using a decision problem to solve the factoring search problem. Further, a search problem hints at using binary search algorithm.

Combining the hints, the idea is to use binary search algorithm to divide the search space by half on each iteration/recursion. To determine which half to continue the search on, we need to answer the question - is there factor in that half, which is a decision problem. We can show that this decision problem belongs to NP by providing a poly-time certifier. And finally, if $P = NP$, we can solve the overall factoring search problem in polynomial time.

Algorithm

```
procedure FindFactor(N)
    return BinarySearchFactor(N, 2, √N)
procedure BinarySearchFactor(N, l, h)
    if l > h then
        return 0
    m = (l + h)/2
    if N mod m = 0 then
        return m
    if ContainsFactor(N, l, m − 1) then
        return BinarySearchFactor(N, l, m − 1)
    else
        return BinarySearchFactor(N, m + 1, h)
```

Correctness

Factors by definition are between 1 and N, inclusive. However, the above algorithm limits the search space between 2 and $\sqrt{N}$. The lower bound is 2 because the question asks for non-trivial factors ($\neq 1$ and $\neq N$). The upper limit is $\sqrt{N}$, because factors occur in pairs: $a * b = N$ and $\sqrt{N} * \sqrt{N} = N$ hence we cannot have both a and b greater than $\sqrt{N}$. Since we are interested in finding only one factor, it is sufficient to search between 2 and $\sqrt{N}$.

BinarySearchFactor performs a binary search between l and h, inclusive. It pick the middle number in the search space and checks if it divides N and hence a factor. If not, it divides this search space into two halves, checks if there is a factor in any of the half by calling into *ContainsFactor*. If it finds a factor in any half, it recuses on that half. Note, it is possible to have factors in both halves, since we are interested in only one factor we pick one half to continue the search. This recursion ends when we find that the middle element is a factor or when our search space becomes empty i.e. $l > h$, at which point there can be no factor of N hence returning 0. This establishes the correctness of *BinarySearchFactor*

FindFactor finds the factor using *BinarySearchFactor*. Since we are interested in non-trivial factors, we restrict the search space as discussed above.

The procedure for our decision problem *ContainsFactor* is missing above. Currently, we don't know any polynomial time algorithm to solve this. We could iterate through the entire search space, but as noted above this is not polynomial in terms of the input length. We can prove that this decision problem is in NP by providing a poly-time certifier as shown below. This certifier is straightforward, it takes the inputs to the decision problem N, l, h and a certificate a. It verifies the certificate a is between l and h and then verifies if a divides N. If $P = NP$, then *ContainsFactor* has a polynomial time algorithm.

```
procedure ContainsFactorCertifier(N, l, h, a)
    if a < l ∨ a > h then
        return False
    if N mod a = 0 then
        return True
    else
        return False
```

Analysis

As provided by question, assuming $P = NP$, then *ContainsFactor* has a polynomial time algorithm say $O(p)$.

BinarySearchFactor runtime can be expressed as $T(n) = T(n/2) + p$ and $T(1) = O(1)$. Expanding this recurrence shows that the runtime is $O(p \log N)$.

Hence, *FindFactor* runs in $O(p \log N)$ time.

Question 2

Lets write the given linear program in standard form:

$$\begin{aligned} &\textbf{maximize} \\ &\quad c^T x \\ &\textbf{subject to} \\ &\quad Ax \leq b \\ &\quad x \geq 0 \end{aligned}$$

So, the given linear program has the following values for $A = \begin{bmatrix} 5 & 3 & 0 \\ 4 & -1 & 0 \\ 0 & -1 & 3 \end{bmatrix}$, $b = \begin{bmatrix} 0 \\ 3 \\ 2 \end{bmatrix}$ and $c = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix}$

The dual is computed as:

$$\begin{aligned} &\textbf{minimize} \\ &\quad b^T y \\ &\textbf{subject to} \\ &\quad A^T y \geq c \\ &\quad y \geq 0 \end{aligned}$$

So, the dual linear program is:

minimize

$$3y_2 + 2y_3$$

subject to

$$5y_1 + 4y_2 \geq 1$$
$$3y_1 - y_2 - y_3 \geq -3$$
$$3y_3 \geq 4$$
$$y_1 \geq 0$$
$$y_2 \geq 0$$
$$y_3 \geq 0$$

Or the dual can be stated as:

maximize

$$-3y_2 - 2y_3$$

subject to

$$-5y_1 - 4y_2 \leq -1$$
$$-3y_1 + y_2 + y_3 \leq 3$$
$$-3y_3 \leq -4$$
$$y_1 \geq 0$$
$$y_2 \geq 0$$
$$y_3 \geq 0$$

Question 3

The question asks us minimize the maximum error which is defined as $|b_i - (\alpha * a_i) - \beta|$ across the 4 given points. Note, here α and β are variables that we want our linear program to find, given a_i and b_i constants. There is no additional constraints mentioned in the question for our variable. So, our linear program would be:

minimize

$$max_{i=1,2,3,4}(|b_i - (\alpha * a_i) - \beta|)$$

However, this doesn't look like a linear program and it is missing constraints. Lets introduce variables e_i to denote the error $|(b_i - \alpha)(a_i - \beta)|$. Then we have,

minimize

$$max_{i=1,2,3,4}(e_i)$$

subject to

$$e_i = |b_i - (\alpha * a_i) - \beta|$$

We can expand the constants a_i and b_i to get:

$$\begin{aligned}
&\textbf{minimize} \\
&\qquad max(e_1, e_2, e_3, e_4) \\
&\textbf{subject to} \\
&\qquad e_1 = |3 - 1\alpha - \beta| \\
&\qquad e_2 = |7 - 2\alpha - \beta| \\
&\qquad e_3 = |5 - 3\alpha - \beta| \\
&\qquad e_4 = |-1 - 4\alpha - \beta|
\end{aligned}$$

The objective function looks different compared to other linear programs we have discussed in class. We can fix this by hiding the max function under a new variable f i.e. $f = max(e_1, e_2, e_3, e_4)$. Also, we can see that f is one of e_i, specifically it is the maximum of e_i or $f \geq e_i$. Using this we can replace e_i to get:

$$\begin{aligned}
&\textbf{minimize} \\
&\qquad f \\
&\textbf{subject to} \\
&\qquad f \geq |3 - 1\alpha - \beta| \\
&\qquad f \geq |7 - 2\alpha - \beta| \\
&\qquad f \geq |5 - 3\alpha - \beta| \\
&\qquad f \geq |-1 - 4\alpha - \beta|
\end{aligned}$$

This looks more like a linear program. $|q|$ is either q or $-q$, so a constraint expression $p \geq |q|$ can be replaced with two constraint expressions: $p \geq q$ and $p \geq -q$ i.e. p would have to greater than or equal to both q and $-q$ at the same time. Hence, our linear program is the following with variables f, α and β:

$$
\begin{aligned}
&\textbf{minimize} \\
&\qquad f \\
&\textbf{subject to} \\
&\qquad f \geq (3 - 1\alpha - \beta) \\
&\qquad f \geq -(3 - 1\alpha - \beta) \\
&\qquad f \geq (7 - 2\alpha - \beta) \\
&\qquad f \geq -(7 - 2\alpha - \beta) \\
&\qquad f \geq (5 - 3\alpha - \beta) \\
&\qquad f \geq -(5 - 3\alpha - \beta) \\
&\qquad f \geq (-1 - 4\alpha - \beta) \\
&\qquad f \geq -(-1 - 4\alpha - \beta)
\end{aligned}
$$

Question 4

Idea

Following the hints provided, we'll construct a bipartite graph $G = (C \cup V, E)$ where C represents the clauses and V represents variables in the 3-SAT problem. Both positive and negative variables are represented by the same vertex in V. If a clause c_i contains a variable v_j, an edge (c_i, v_j) is added in bipartite graph G. Next, we'll find the maximum matching M using Ford-Fulkerson algorithm in polynomial time. For each edge $(c_i, v_j) \in M$, we'll set v_j accordingly to ensure clause c_i is satisfied; remaining variables are set arbitrarily. This forms a satisfying assignment to the given 3-SAT problem.

Algorithm

```
procedure SOLVE3SAT(3sat)
    G = {C ∪ V, E}
    add vertex c_i to C for each clause in 3sat
    add vertex v_j to V for each variable (positive or negative) in 3sat
    add edge (c_i, v_j) to E if variable v_j occurs in clause c_i in 3sat


    M = MaxMatching(G)
    for (c_i, v_j) ∈ M do
        if v_j occurs in positive form in c_i then
            v_j = True
        else
            v_j = False
    return V
```

Correctness

The construction of graph G given $3sat$ is pretty straightforward as described above. Note, that all edges in G are between clauses and variables, which implies that G is indeed bipartite.

Next, $Solve3SAT$ finds the maximum matching as discussed in class. We need to prove that maximum matching finds the correct variable assignment that satisfy $3sat$. To prove this, we use the hint to show that there is a matching that covers all vertices in C. And then we find this matching using maximum matching, which implies that we need to prove that every maximum matching cover all vertices in C. And finally, to answer the question we need to show that assigning these matched variables accordingly satisfies each clause and hence satisfies $3sat$.

The marriage theorem discussed in class, states that there is a perfect matching for bipartite graph $G' = (L' \cup R', E')$ with $|L'| = |R'|$ iff $|N(S')| \geq |S'|$ for all subsets $S' \subseteq L'$. We proved this by first establishing a matching that covers all vertices in L' iff $|N(S')| \geq |S'|$. And then since $|L'| = |R'|$ it implies the matching covers all vertices in $|R'|$ as well. Clearly, $|L'| = |R'|$ condition doesn't hold for our 3-SAT bipartite graph. We only require the first half of the theorem proof i.e. there exists a matching that covers all vertices in L' iff $|N(S')| \geq |S'|$.

Consider $S \subseteq C$ and the subgraph induced by S and its neighbors $N(S)$, we can say that each clause contains exactly 3 variable placeholders by definition of 3-SAT that needs to be filled in by variables available in $N(S)$. The total number of variable placeholders is $3|S|$ and there are $|N(S)|$ variables. So, the average number of placeholders that a variable would have to fill is $\frac{3|S|}{|N(S)|}$. This average cannot exceed 3 since each variable can appear in atmost 3 clauses as given in the question i.e. $\frac{3|S|}{|N(S)|} < 3$. Assume, $|N(S)| < |S|$ for some subset S,

so we get $\frac{|S|}{|N(S)|} > 1$ or $\frac{3|S|}{|N(S)|} > 3$. Hence our assumption is wrong i.e. there is no subset $S \subseteq C$ for which $|N(S)| < |S|$. In other words, for every subset $S \subseteq C$, $|N(S)| \geq |S|$ holds. Now, by marriage theorem, we can say that G has a matching that covers all vertices in C.

This matching cannot have a size greater than maximum matching by definition. Further, we can claim that every maximum matching has to cover all vertices in C. If this were not true, then there is a maximum matching M' which doesn't cover atleast one vertex $c_i \in C$. Say this maximum matching covers $C_m \subset C$. Consider subset $S = C_m \cup \{c_i\}$, as proved above $|N(S)| \geq |S|$ holds for S. So by applying marriage theorem to the subgraph induced by S and its neighborhood $N(S)$ there must exist a matching in this subgraph that covers all vertices in S and clearly this matching has larger size than M' contradicting that M' is maximum matching. Hence, every maximum matching covers all vertices in C. And as discussed in class, a maximum matching can be found using Ford-Fulkerson max flow algorithm which is encoded by $MaxMatching$ procedure call in the above algorithm.

Now, that we have a matching that covers all clauses in C, we use this to assign values to variables such that each clause is satisfied. Each clause can be satisfied by having atleast one literal satisfied i.e. having atleast one literal evaluate to $True$. By definition of matching, each clause is matched to a separate variable. Hence we can assign value to this variable such that the corresponding literal evaluate to $True$. Specifically, if the literal is the positive variable, we assign $True$ to this variable otherwise we assign $False$. Other variables that are not covered by this matching does not matter to satisfy the clause or the 3-SAT, hence we can assign values arbitrarily. $Solve3SAT$ returns this variable assignment that satisfies each clause and hence 3-SAT.

Analysis

Say, $3sat$ has p clauses and q variables, then in the bipartite graph G, we have $|C| = p$, $|V| = q$ and $|E| = 3p$. So to construct this graph, we need $O(p + q + 3p) = O(p + q)$ time.

The max flow algorithm runs in $O(m^2 \log C)$, where m is the number of edges in flow network and C is the maximum capacity. $MaxMatching$ internally construct a flow network based on the bipartite graph with $p + q + 2$ vertices, $p + 3p + q = 4p + q$ directed edges and assigns unit capacity to all these edges. It then runs max flow algorithm on this flow network to find the maximum matching. So, $MaxMatching$ runs in $O((4p + q)^2 \log C) = O((4p + q)^2) = O((p + q)^2)$.

The last for loop goes over all the edges in M assigning values to corresponding variables. Since, there are $3p$ edges, $|M| \leq 3p$. Hence the for loop takes $O(p)$ time.

Overall, $Solve3SAT$ takes $O(p + q) + O((p + q)^2) + O(p) = O((p + q)^2)$.