Homework 6
UCLA-CS180-S18

Quentin Truong

June 8, 2018

1 Question 1

1) Algorithm

```
FIND-IS (G, k)
    if not INDEPENDENT-SET(G, k)
        return {}

    independent_set = {}
    for vertex in G.vertices
        G2 = G - vertex // remove vertex and associated edges
        if INDEPENDENT-SET(G2, k)
            G = G2
        else
            independent_set += vertex

    return independent_set
```

2) Proof
Suppose there does not exist an independent set of size k in G
Then FIND-IS will return the empty set
Otherwise, there exist an independent set of size k in G
Iterating through each vertex x of G, we remove x from G and check if the resulting graph still has an independent set of size k. If it does, then we may safely remove that vertex from the graph, because there still exist some independent set of size k. If it does not, then we know that vertex x is necessary for an independent set. Thus, we keep the vertex in the graph and we add it to our independent_set. Since we only remove vertices when we are guaranteed an independent set still exist, and we only add vertices when an independent set ceases to exist, we are guaranteed to end with an independent set.

2 Question 2

1) Transformation
Transform ϕ into the following described graph.
For each variable x in ϕ, represent x as vertex x connected by an edge to another vertex $-x$. (Will refer to these vertices and edges as the top row)
Below those vertices, for each variable $x1, x2, x3$ in each clause in ϕ, represent $x1, x2, x3$ as vertices and fully connect them by edges. (Will refer to these vertices and edges as the bottom row)
Then, connect each vertex $x1, x2, x3$ from each clause to their respective vertices from the top row.
Set vertex cover size k = 2 * number of variables + 3 * number of clauses.
2) Proving =>
Suppose we have a satisfying assignment for ϕ.
Then, from the top row of vertices, select vertices that are true variables. For example, if x were assigned true, select vertex x; otherwise, select -x. This will result in all top row edges having at least one end in the selected set. This will be a total of n vertices, where n is the number of variables in ϕ.
Then, from the bottom vertices, select the two non-true variables. This will result in bottom row edges having at least one end in the selected set. Note that because we selected the non-true variables in the bottom, and true variables in the top, the edges connecting bottom and top will still have at least one end in the selected set (true variables in the top are connected to true variables in the bottom, so need to select true in top and false in bottom, otherwise we would be covering both ends of a single edge and not covering at least one end of edges connected to non-true variables). This will be a total of $2 * m$ vertices, where m is the number of clauses in ϕ.
Then, we have a vertex cover with exactly $k = n + 2 * m$.
3) Proving <=
Suppose we have a vertex set of size at least $k = n + 2 * m$.
Then, n of those vertices must belong to the top row vertices, otherwise we would not have a vertex cover.
Allowing those n vertices to be the satisfying assignment for our variables, notice that the remaing $2 * m$ vertices must cover the non-true variables in the bottom row. If one covered a true variable, then we would not have a vertex cover because at least one end of an edge connected to a non-true variable would be uncovered. This results in one true variable per clause.
Then, we have a satisfying variable assignment.

3 Question 3

1) Transformation from 3-SAT to LPS
For each variable in ϕ, reorder variables such that each variable is AND with some other variables. AND with zeroes as necessary. Create n many rows, with n ANDs per row. Pad as necessary. Then we have an equation like
$((a_{11} \cap x_1) \cup ... \cup (a_{1n} \cap x_n) \cup (\neg b_1)) \cap ... \cap ((a_{n1} \cap x_1) \cup ... \cup (a_{nn} \cap x_n) \cup (\neg b_n)) = TRUE$

Transform matrix into a single boolean expression by joining each row with boolean AND.

$$\begin{bmatrix} (a_{11} \cap x_1) \cup ... \cup (a_{1n} \cap x_n) \cup (\neg b_1) \\ ... \\ (a_{n1} \cap x_1) \cup ... \cup (a_{nn} \cap x_n) \cup (\neg b_n) \end{bmatrix} = \begin{bmatrix} TRUE \\ ... \\ TRUE \end{bmatrix}$$

Transform multiplication into boolean AND. Transform addition into boolean OR. Transform subtraction into boolean OR with negation over the following variable. Interpret nonzero numbers as TRUE; interpret zeros are FALSE. Transform the zeroes on the right hand side to TRUE.

$$\begin{bmatrix} (a_{11} * x_1) + ... + (a_{1n} * x_n) - (b_1) \\ ... \\ (a_{n1} * x_1) + ... + (a_{nn} * x_n) - (b_n) \end{bmatrix} \geq \begin{bmatrix} 0 \\ ... \\ 0 \end{bmatrix}$$

For arbitrary matrix A in R^{nxn}, vector x in R^n, and vector b in R^n, form $Ax - b \geq 0$
2) Proving =>
Suppose we have phi in CNF form
This boolean expression may be converted without loss of information to
$((a_{11} \cap x_1) \cup ... \cup (a_{1n} \cap x_n) \cup (\neg b_1)) \cap ... \cap ((a_{n1} \cap x_1) \cup ... \cup (a_{nn} \cap x_n) \cup (\neg b_n)) = TRUE$

Then we split up the n rows into a matrix
Then, we transform AND to multiplication, OR to addition, and NEGATE to subtraction
Then we solve using LQS
3) Proving <=
Suppose we have $Ax \geq b$
Multiplying and subtracting, we receive $Ax - b \geq 0$
We may convert this to boolean expressions
Then this boolean expression may be put into CNF form and solved by SAT

4 Question 4

a) Proof
Let a bijective mapping from vertices of G to vertices of H be the certificate.
The verifier algorithm will be as follows:
Confirm that the mapping is a bijection between vertices of G and vertices of H. This can run in polynomial time because we are just checking one-to-one and onto.
Apply the mapping to vertices/edges of G and confirm that all of the transformed edges are in H. This can run in polynomial time because we are just confirming the existence of edges.
Since there exist a polynomial verifier for graph isomorphism, graph isomorphism is in NP.

B) Proof
Let a subset of vertices of I be the certificate.
The verifier algorithm will be as follows:
For each vertex x in graph I, confirm that either x is in the subset of vertices or that x is connected by an edge to another vertex which is in the subset of vertices. This can run in polynomial time because we are just checking the relationship of vertices of a subset to vertices of the graph.
Since there exist a polynomial verifier for graph isomorphism, graph isomorphism is in NP.