CS 4820 : Algorithms

*HW1– **NP-Completeness***

Bryce Evans (`bae43`)
September 12, 2013

1 Near Cliques

1.1 Solution

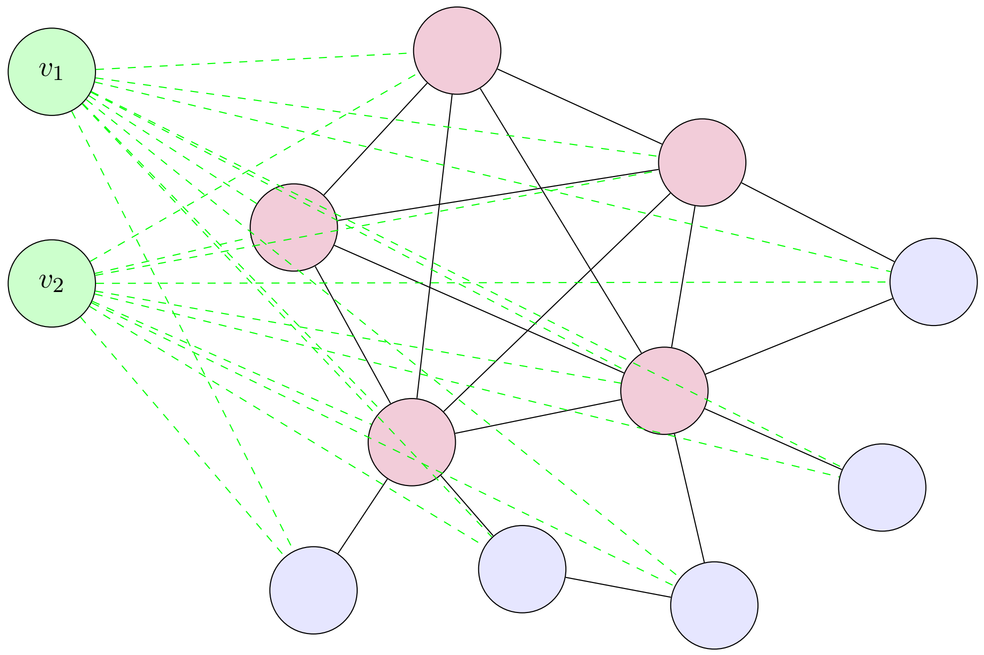
Prove: *Clique Problem* $\leq_p$ *Near Clique Problem*

The Clique of Size k Problem in NP-Complete reduces to the Near Clique of Size k Problem. Both the near clique and full clique problems take inputs $G = (V, E)$, and output `True` if a subgraph g (defined separately in each problem) has the property $g \subseteq G$, and `False` otherwise. Create the efficient reduction σ as follows:

σ **Reduction:** Given graph $G = (V, E)$, add two nodes v_1, v_2 that connect to every $v \in V$. Solving the near clique problem on this graph solves the clique problem as well.

If a clique of size k exists in G, then adding v_1, v_2 and connecting them to every other vertex will form a near clique of size $k + 2$ because no edge connects v_1 and v_2. If one or more edges are missing, then adding additional nodes that are not connected to each other will cause at least 2 edges to now be missing among the nodes, and therefore the near clique algorithm will return false.

Example:



Graph G with clique k_5 highlighted in red and v_1, v_2 connected to every vertex in G. A near clique of size $k+2$ forms between the red and green colored nodes.

Runtime of σ

Add nodes v_1, v_2:	$O(1)$
Connect v_1, v_2 to $v \, \forall v \in V$:	$O(V)$
Total Reduction Runtime:	$O(V)$

τ **Reduction:** Construct a function τ that takes the output of σ and converts it to a valid solution of the clique problem:

The construction of τ is trivial - the output of σ is equivalent to the clique problem. A near clique of size $k+2$ will exist after adding v_1, v_2 if and only if a clique of size k existed in G prior.

Runtime of τ

Output boolean is equivalent to solution to clique problem:	$O(1)$
Total Reduction Runtime:	$O(1)$

1.2 Efficient Verifier:

Given a solution set S of k elements to the near clique problem, remove v_1, v_2 and corresponding edges. Take this graph and verify that each node has an edge to every other node. If every node connects to all others, the algorithm should have returned `True`, and if not, then `False`.

Runtime of Verifier

Remove v_1, v_2	$O(1)$
Verify all k nodes connect to others	$O(k^2)$
Total Runtime:	$O(k^2)$

2 Submatrix Domination

2.1 Solution

Prove: *Clique Problem* $\leq_p$ *Near Clique Problem*

The Clique of Size k Problem in NP-Complete reduces to the Matrix Subdomination Problem. The clique problem takes input $G = (V, E)$, and output `True` if there exists a clique of size k that exists in G and `False` otherwise. Create the efficient reduction σ as follows:

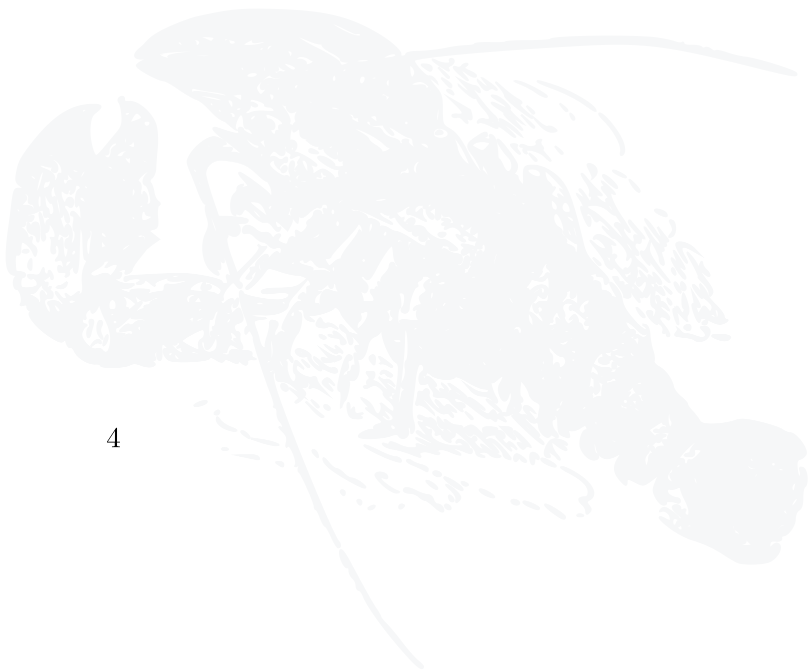
σ **Reduction:** Given Graph G, find if it contains a clique of size k. Create adjacency matrices M_k, M_G for both the clique and G, such that each row and column (r_i, c_i) represents some node $i \in G$. If i is connected to $j \in G$, $M_G[i, j], M_G[j, i] = 1$. The main diagonal has all values of 2.

Permute the columns so that the taken columns are consecutive. This is an isomorphism to the original graph as this essentially just a renaming of the nodes. Because the graph is undirected, the matrix is symmetric and the rows are then consecutive like the columns.

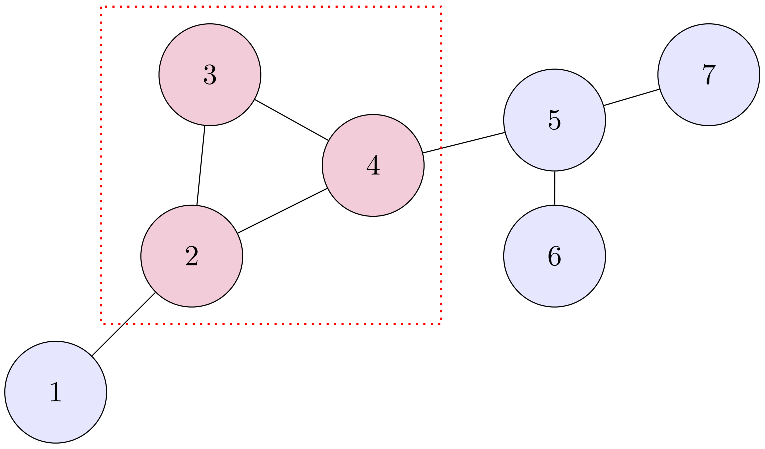
Because of the properties of this adjacency matrix, diagonals are the highest values in the matrices because no other value can be > 1, meaning diagonals cannot be dominated by anything but another value in the diagonal. This forces no node to be accidentally paired up with another node by domination. For all diagonal values of M_k to be dominated in

M_G, the diagonals must align to each other, as no value in M_G other than a diagonal is ≥ 2.

Because of the properties of adjacency matrices, if every value in the matrix is a 1 (excluding diagonal), then every node in that set is connected to all others in the set. Therefore, the mapping of $r(\cdot), c(\cdot)$ returns the set of nodes that form the clique. If M_G contains M_k, then G contains the clique and the algorithm returns `True`. It is guaranteed that if such a submatrix is found that it is a clique because the matrix must be centered on the diagonal, with each node the columns represent being connected to each other. If the submatrix is found within M_G then, the clique exists within G.



Example: Find if graph G contains k_3 (in red).



Graph G with clique k_3 highlighted.

In this simple example, the nodes are numbered in way such that the clique is formed by consecutively numbered nodes, causing the submatrix to be easily spotted in M_G. Note that this is not always the case, but the vertices may be renumbered to cause this.

$$M_k = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad M_G = \begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 \end{bmatrix}$$

Runtime of σ

Compute M_k:	$O(k^2)$
Compute M_G:	$O(V^2)$
Total Reduction Runtime:	$O(V^2)$

τ **Reduction:** Construct a function τ that takes the output of σ and converts it to a valid solution of the clique problem:

This is a decision problem with only a boolean output. `True` and `False` map to the same values and the reduction is trivial.

Runtime of τ

Output boolean is equivalent to solution of vertex cover:	$O(1)$
Total Reduction Runtime:	$O(1)$

2.2 Efficient Verifier:

Given a solution set S to the submatrix domination problem, test all values of A into $r(\cdot), c(\cdot)$. If every value of $r(\cdot), c(\cdot)$ matches, then the algorithm should have returned `True`, and if not, then `False`.

Runtime of Verifier

Iterate through m_1 rows and n_1 columns of A:	$O(n_1 m_1)$
Total Runtime:	$O(n_1 m_1)$

3 Party Invitations

3.1 Solution

Prove: *Vertex Cover* $\leq_p$ *Party Invitation Problem*

The Vertex Cover Problem in NP complete reduces to the party invitation problem. If we can solve the party invitation problem, we could solve the vertex cover problem as well. Create this efficient reduction σ as follows:

The party invitation takes inputs:

- a set of lists $L = \{l_1, l_2, l_3, \ldots, l_k\}$
- a set of corresponding values $M = \{m_1, m_2, m_3, \ldots, m_k\}$ of the *minimum* number of elements that can be chosen from $l_i \in L$.
- n - the max number of elements that can be chosen from all lists

The solution to the problem outputs a boolean `True` if there exists a set of elements smaller than n in cardinality that satisfies all the constraints, and `False` otherwise.

Recall that vertex cover has given $G = (V, E)$, and k, the max number of vertices that can be chosen in the cover, with the constraint $\forall e = (u, v) \in E$, e is connected to some $o \in O$. Vertex cover returns a boolean `True` if there exists some set of vertices S such that $|S| \leq k$ and `False` otherwise.

σ **Reduction:** Consider n to be the number of edges, $|E|$. For all $e \in E$, at least 1 vertex must be taken, so $M = \{1, 1, 1, \dots, 1\}$. For all $e = (u, v) \in E$, construct a list $L' = \{\{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_3\}, \dots, \{u_k, v_k\}\}$ Use L' as the input set of lists for the party invitation problem. This reduction maintains all of the constraints for Vertex Cover - each edge e must have one valid vertex be chosen to cover it where a valid vertex is one that has e as an endpoint.

Runtime of σ

Calculate number of edges:	$O(E)$
Create List L':	$O(E)$
Find M:	$O(1)$
Total Reduction Runtime:	$O(E)$

τ **Reduction:** Construct a function τ that takes the output of σ and converts it to a valid solution of the vertex cover problem:

The construction of τ is trivial - the output of σ is equivalent to if a vertex cover of size k exists. Vertex Cover (VC) will return true if and only if the party invitation problem (PI) returns true because each edge $e \in G$ requires at least one vertex it is connected to to be chosen. This is maintained in PI because each set has a minimum number of elements needed to be picked to be 1, with valid elements being only vertices touching that edge. Therefore, under the constraints of PI, each edge will have at least one vertex that it is connected to be chosen.

Runtime of τ

Output boolean is equivalent to solution of vertex cover:	$O(1)$
Total Reduction Runtime:	$O(1)$

3.2 Efficient Verifier:

Given a solution set S of n elements to PI, a polynomial verifier is as follows:

For each element e in list $l \in L$, check every element in S and see if it exists. If every e exists in S, the algorithm should have returned `True`, and if not, then `False`.

Runtime of Verifier

Iterate through every e in list $l \in L$,

with m being the total number of these elements: $m = \sum_{l \in L} |l|$ $\quad O(m)$

Iterate through every element of S $\quad O(n)$

Total Runtime: $\quad O(nm)$