On Maximum Independent Sets ©

Or S. Naim
Founder of CUCUMBER - An OrSN Company
Bar-Ilan University ISRAEL
Faculty of Exact Sciences
HelpDeskCucumber@Yahoo.Com

Abstract
We consider the problem of finding a Maximum Independent Set in a graph. A recent paper by Xiao and Nagamochi (2017) shows an algorithm that can find a maximum independent set in $O(1.1996^n)$ time and polynomial space. Even with restrictions on the degree of vertices, no polynomial-time algorithm discovered yet. In this paper we are showing a polynomial-time algorithm solving this problem.

1. Introduction

All graphs $G = (V, E)$ in this paper are simple, undirected, and unweighted.
A Maximum Independent Set of a graph is the largest subset of vertices that we can assemble without an edge $e \in E$ connecting any of its members.
A Maximal Independent Set (MIS) of a graph is a subset of vertices with no edge connecting any of its members. By adding any additional vertex from the vertices set V, we would violate this condition. It is considerably easy to find a Maximal Independent Set using a greedy algorithm.

Lemma 1: Any Maximum Independent Set is a Maximal Independent Set.
Proof: Let us assume in contradiction, there is a Maximum Independent Set that is not a Maximal Independent Set. According to the definition of Maximal Independent Set, the Maximum Independent Set has two vertices that are neighbors therefore, the Maximum Independent Set is violating the term of an independent set and is not in contradiction.

2. Preliminaries

The easiest way of finding a Maximum Independent Set is to check all $O(2^{|V|})$ combinations of vertices and returning the largest subset that is independent. The runtime of this approach would be exponential. In their paper: "On Cliques in graphs" (1965), Moon, and Moser, show that the number of maximal independent sets in a graph is at most $3^{\frac{n}{3}}$. A possible method to increase our chances of returning a correct output would be to repeat the process multiple times, start from a random vertex on each iteration, and output the largest subset among all iterations. To amplify the accuracy to be polynomial, we have to repeat the algorithm exponential number of times, hence losing the runtime benefit of a Monte-Carlo algorithm.

3. The Cucumber Method:

After gaining the intuition, a probabilistic approach isn't fruitful will try to solve the problem differently. For starters, we need to make some definitions:

Definition 2: A **degree** of a vertex v: $deg(v)$ is the number of neighbors vertex v has.

Definition 3: A **Social Degree** is a number reflecting the summary of all vertices' degrees in a set: $\Sigma_{i=1}^{n} deg(v_i)$.

Lemma 4: The Social Degree of a subset is lesser or equal to the Social Degree of V.
Proof: Since the degree of a vertex is a non-negative number, the more vertices you have in a set, the higher the social degree would become.

Lemma 5: The Social Degree of V equals $2 * |E|$.
Proof: Since each edge increases the degree of two vertices by one, the Social Degree of V increases by 2, so overall, the social degree of V equals to $|E| * 2$.

Lemma 6: The Social Degree of any independent set is lesser or equal to $|E|$.
Proof: Let us assume in a contradiction there is an independent set with a social degree of $|E| + 1$. As such, from the pigeonhole principle, at least two members of the Independent Set must be neighbors, in contradiction.

We are now ready to present the Cucumber method of finding a Maximum Independent Set in a graph. The general idea is to look for the vertices with the smallest degree and add them one at a time. The purpose of this is to shrink the input with each iteration while letting as many vertices as possible joining the set.

Now we will see a preparation polynomial algorithm to help get our input graph ready.

```
$CuPreperation(G(V,E))$:
    1. $Initialize\ an\ empty\ array\ CuArray = \emptyset$
    2. $Scan\ the\ graph\ and\ give\ each\ vertex\ a\ unique\ name\ using\ Breadth - First\ Search$
    3. $Scan\ the\ graph\ using\ BFS\ and\ add\ to\ CuArray\ each\ vertex\ with\ the\ following\ fields$:
        3.1. $Degree$
        3.2. $Blacklist\ (Containing\ all\ neighbors\ of\ the\ vertex)$
    4. $Sort\ CuArray\ by\ the\ degree\ of\ each\ vertex.$
    5. $Return\ CuArray$
```

Runtime: Step 1: $O(1)$, Step 2: $O(|V| + |E|)$, Step 3: $O(|V| + |E|)$ for Breadth-First Search and $O(|V|)$ time on each vertex to assemble the local blacklist field for each vertex. The total runtime for this step is $O(|V|^2 + |E|)$.
Step 4: Efficient sorting algorithm takes: $O(|V| * log(|V|)$.
Overall: $O(|V|^2 + |E|)$.

This algorithm however, is not complete.

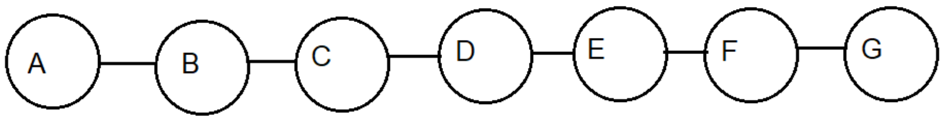

Figure 1, A graph that allows too much "freeness" to CuPreperation.

The problem with CuPreperation is, that it doesn't force our need to let as many vertices as possible joining the set in the next iteration. The graph in figure 1, consists of seven vertices, and each vertex has the degree of 2. Except A and G. Since they are no neighbors of each other they would definitely added to the independent set. As such, B and F are definitely disqualified. In regards C,D,E it is clear that we want C and E to join, but there is no guarantee, Vertex D will not "win" in the sorting, and come before them. In that scenario, we would add D, preventing both C and E from joining, and output an incorrect set.

We need to ensure somehow, Either C or E or both defeat D in the sorting, despite all having the same degree.

Definition 7: Shortest Path $\delta(v,u)$ between two vertices v and u is a number representing the minimum number of edges we need to traverse in order to reach from vertex v to vertex u. The shortest path from a vertex to itself: $\delta(v,v) = 0$.

Lemma 8: Vertex u would be called a neighbor of vertex v if and only if $\delta(v,u) = 1$.
Let us assume $\delta(v,u) = 1$, in other words there is an edge connecting v to u and therefore they are neighbors.
Let us assume vertices v and u are neighbors. As such, there is an edge connecting them (By definition) and therefore $\delta(v,u) = 1$.

Notice that when adding a vertex to an independent set, we prevent all vertices with whom it has a shortest path equals to 1 from joining, but ideally, if possible, we would want to add all vertices with whom it has a shortest path equals to 2. Of course, if their degree is the smallest.

Definition 9: Single Source Shortest Path (SSSP), We calculate the shortest path from one vertex to the other $|V| - 1$ vertices. An algorithm that solves this problem for the type of graphs we are working with is the Depth First Search (DFS) runs in $O(|V| + |E|)$.

To fulfill our ideal, and prevent such scenarios as described in figure 1, we would add a "Tie breaking" mechanism, in an event multiple vertices have the same degree. We are adding to each vertex a field of the shortest path to each of the vertices in the graph, and in an event of a tie for the next vertex to win in the sorting, the one with the shorter path will win.

```
$CuPreperation(G(V,E))$:
    1. $Initialize\ an\ empty\ array\ CuArray\ = \emptyset$
    2. $Scan\ the\ graph\ and\ give\ each\ vertex\ a\ unique\ name\ using\ Breadth - First\ Search$
    3. $Scan\ the\ graph\ using\ BFS\ and\ add\ to\ CuArray\ each\ vertex\ with\ the\ following\ fields$:
        3.1. $Degree$
        3.2. $Shortest\ path\ vector$
        3.3. $Blacklist\ (Containing\ all\ neighbors\ of\ the\ vertex)$
    4. $Sort\ CuArray\ by\ the\ degree\ of\ each\ vertex. In\ an\ event\ of\ a\ tie, refer\ to\ the\ shortest\ path$
       $vector\ of\ the\ last\ vertex\ added\ and\ among\ the\ vertecies\ with\ the\ smallest\ degree\ to\ be$
       $added, add\ the\ vertex\ with\ the\ shortest\ path \geq 2.$
    $In\ an\ event\ of\ another\ tie, add\ an\ arbitrary\ vertex\ from\ the\ involved\ in\ the\ second\ tie.$
    5. $Return\ CuArray$
```

Runtime: Step 1: $O(1)$, Step 2: $O(|V| + |E|)$, Step 3: $O(|V| + |E|)$ for Breadth-First Search and $O(|V|)$ time on each vertex to assemble the local blacklist field for each vertex and $O(|V| + |E|)$ for DFS. The total runtime for this step is $O(|V|^2 + |E|^2)$. Step 4: Efficient sorting algorithm takes: $O(|V| * log(|V|)$, but handling the ties, would require an additional $O(|V|)$ time for a total of $O\big(|V|^2 * Log(|V|)\big)$ time.
Overall: $O\left(\max\left((|V|^2 + |E|^2), \big(|V|^2 * Log(|V|)\big)\right)\right)$.

Coming up next, we are about to see the algorithm that fulfills our general idea.

Now we are ready to see the polynomial-time algorithm for finding a Maximum Independent Set.

```
$CuAlgorithm(CuArray)$:
    1. $Initialize\ an\ empty\ set\ CuSet\ = \emptyset$
    2. $Initialize\ a\ global\ Blacklist\ = \emptyset$
    3. $Initialize\ Social\ Degree\ = 0$
    4. $Loop\ through\ CuArray$
        4.1. $If\ vertex\ IS\ NOT\ in\ global\ Blacklist$
                4.1.1. $Add\ vertex\ to\ CuSet$
                4.1.2. $Merge\ the\ vertex\ local\ blacklist\ with\ the\ global\ blacklist\ and\ add\ the\ vertex\ to\ the\ blacklist.$
                4.1.3. $Increase\ Social\ Degree\ by\ the\ degree\ of\ the\ vertex.$
                4.1.4. $If\ |global\ Blacklist| \geq |V|$
                        4.1.4.1 $Return\ CuSet$
                4.1.5. $If\ Social\ Degree\ = |E|$
                        4.1.5.1 $Return\ CuSet$
    5. $Return\ CuSet$
```

Steps 4.1.4 and 4.1.5 are optional.
To prove the correctness of CuAlgorithm, we need to prove three lemmas.

Lemma 10: The independency property of CuSet. Steps 4.1 and 4.1.2 of CuAlgorithm, enforce that only vertices without neighbors in CuSet are permitted to join the set. As such, CuSet is an Independent Set.

Lemma 11: CuSet is a Maximal Independent Set: Let us recall the definition of a Maximal Independent Set. It is an independent set that an additional vertex will cease to keep this property for the set. By Lemma 7, we know that CuSet is independent. By the end of the run, the global blacklist contains the entire V. Therefore, all vertices are in CuSet or are neighbors of CuSet. As such, we cannot add a vertex to CuSet.

4. First Selection Matters
The algorithm in its current form still has an undesired degree of freedom. Let's take a look at the following graph:

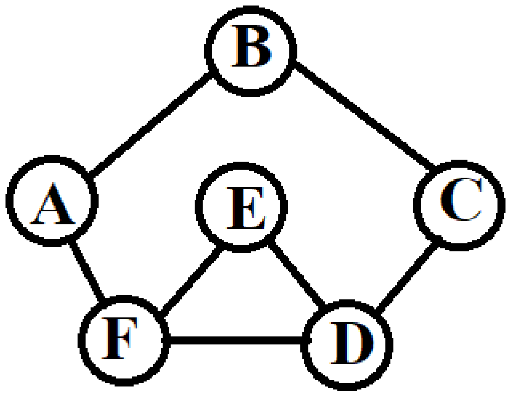

Figure 2, a graph that the degree of freedom can be abused on.

The graph in figure 2, has four vertices with the lowest degree (A, B, C, E), as such, CuAlgorithm will add an arbitrary vertex from these four, and returns an incorrect output if B was added first. In such a case, both A and C will be blacklisted, and among (D, E, F) we may add only one more vertex (E, since its degree is the lowest among the three) while the other two get blacklisted, overall, a set of two vertices will be returned. If we are starting with either (A, E or C) however, the output consists of three vertices (A, E AND C). To combat that, we will run CuAlgorithm from each vertex with the lowest degree (Four times for the graph in figure 2 and up to $O(|V|)$ times if all vertices in the graph have the same degree), and returns the largest set returned from the iterations.

Lemma 12: CuSet is larger or equal in size to any other independent set.
To prove this, we will remove an arbitrary vertex v from CuSet, and show we can add up to a single vertex in its place. Since the neighborhood is a mutual property, we can be sure that no neighbor of the removed vertex is in the set and safely remove it from the blacklist. In regards to its neighbors, we will only remove them from the blacklist if they are not neighbors of other vertices in CuSet. Because the graph is connected, One of three scenarios can occur:

1. The degree of the neighbor of the removed vertex is 1 (That neighbor is the only neighbor of the removed vertex). In such a case, we can add the neighbor to CuSet.
2. The degree of the neighbor of the removed vertex is more than 1. In such a case, we have multiple scenarios:
   2.1. Both the neighbor and the removed vertex have share neighbors. In such a case, all shared neighbors "compete" on one available spot in CuSet.
   2.2. The neighbor has neighbors which it doesn't share with the removed vertex, and they are already in CuSet, preventing the neighbor from joining.
   2.3. The neighbor has neighbors which it doesn't share with the removed vertex, and they are not in CuSet. In such a case, we can add the neighbor.
3. The graph consists of only one vertex, resulting in no substitution.

Scenario 1 can only occur if the graph consists of two vertices. That's because we sort the vertices by their degree and add the vertex with the lower degree first if possible. Since the neighbor has a degree of one, and it wasn't qualified to CuSet, the removed vertex also has a degree of 1. Scenario 2.3 can also occur only once for a similar reason.

Lemmas 10 to 12 prove the correctness of our algorithm, and that it returns a valid Maximum Independent Set.
Run time: Besides constant time operations, our algorithm requires $O(|V|)$ time for the loop in step 4 and $O(|V|)$ to merge the blacklists on each iteration. For a total of $O(|V|^2)$. Since the preparation took us longer, we are getting this algorithm for practically free. Overall, the runtime of our algorithm is:
$O\left(|V| * \left(\max\left((|V|^2 + |E|^2), (|V|^2 * Log(|V|))\right)\right)\right)$.

Space complexity: The length of CuArray, CuSet, and Blacklist are $O(|V|)$. CuArray also includes a field for the neighbors of each vertex as well as a Single Source Shortest Path vector for a total of $O(|V| * |V|) \in O(|V|^2)$. We also require $O(|V|^2)$ space to store the sets from all $O(|V|)$ runs.

5. P = NP:

To prove such a bold claim, we need to find a NP-Complete problem that can be solved in a polynomial time. NP-Complete problem is a problem which belongs to NP class as well as NP-Hard. We would say a problem belongs to NP, if given a solution to the problem, we could verify its correctness in a polynomial time. It was proved that Maximum Independent Set is NP-Hard problem, hence we will not prove this claim again.

Lemma 13: Maximum Independent Set belongs to class NP.
Proof: To prove lemma 13, we need to find an algorithm that verifies a solution to the problem in a polynomial time. We would use CuAlgorithm found in the previous paragraph. Given a subset of vertices S, we would check if any pair of vertices are neighbors. If there is no such a pair, S is independent set. We would then check if S is a Maximal Independent Set, by trying to add a vertex from V/S to S, without violating the independence property of S. If we failed, S is Maximal Independent Set. Finally, We will use CuAlgorithm, and check the size of the output set Y, as a reminder, CuAlgorithm returns Maximum Independent Set. As such if $|Y| = |S|$ then S is a Maximum Independent Set, and we verified the solution in a polynomial time. Overall
$Maximum\ Independent\ Set \in NP$.

Lemma 14: Maximum Independent Set is NP-Complete problem.

Lemma 15: $P = NP$
Proof: CuAlgorithm solves the NP-Complete problem, Maximum Independent Set in a polynomial time, as such
$Maximum\ Independent\ Set \in P \Rightarrow Maximum\ Independent\ Set \in P \cap NPC \Rightarrow P \cap NPC \neq \emptyset \Rightarrow \boldsymbol{P = NP}$.

6. Discussion

Our work is a huge step forward in solving a problem that was considered NP-Hard for decades. We now saw the Maximum Independent Set is in P. We also discovered $Maximum\ Independent\ Set \in NPC$ and we solved the Millennium Prized Problem $P\ Versus\ NP$.

References:

"Exact algorithms for maximum independent set" Xiao and Nagamochi (2017).

"On Cliques in graphs" Moon and Moser (1965).

Special Thanks:
Gilad Madmon.

Author: Or S. Naim ©, Bar Ilan University, ISRAEL.
Founder of CUCUMBER an OrSN Company.
Faculty of Exact Sciences.
January 24th , 2021.

...