Polynomial Time Approximation Algorithm for Maximum Independent Set for graphs with average degree = $O(|V|^{0.5})$

Zeling Li

For CSCE 669

Introduction

In this project I introduce and discuss a polynomial time approximation algorithm that can achieve a bounded approximation ratio for the maximum independent set problem for graphs with O($|V|^{0.5}$). There are already algorithms that solve maximum independent set where graphs have a limited degree or are close to planar [Magen,2009] (which covers a large range of real world applications). However, this algorithm tolerates a broader set of possible graph inputs, and thus it potentially covers a broader range of real world applications. Included in the report is code and data that demonstrates the performance of the algorithm on graphs with up to 841 vertices.

The algorithm runs in O(NV^2) time, which is O($N^{2.33}$) = O($V^{3.5}$) time for graphs fulfilling the condition of average degree <= $|V|^{(1/2)}$. The data collected in this algorithm shows that the ratio of the size of the algorithm's output over the expected size of the maximum independent set increases monotonically with the size of the input graph.

This report is a preliminary discussion of my algorithm, and my plans for continued development will be outlined in the "Future Work" section.

Approach

A naïve algorithm for approximate max-independent set would be to start with all the vertices in a graph and removing the highest degree vertex until we get an independent set. However, if we start with a subgraph where we know beforehand that large chunks of vertices are independent of each other, we likely get much better results than the naïve algorithm. Trying to achieve this latter result in polynomial time is part of the intuition for my algorithm.

The basic idea of the algorithm is that we first divide the input graph into planar subgraphs, combine the independent sets of those planar subgraphs, and then remove vertices from the result until we are left with only vertices that form an independent set on the input graph. This approach relies on the following facts:

1) For planar graphs, there exists a fully polynomial time approximation algorithm for independent set [Baker, 1983].

2) The planarity of a graph can be checked in linear time [Mehlorn,1996].

3) The number of planar subgraphs of the input graph is sublinear to the number of vertices in the input graph.

The initial intuition behind my algorithm was that by first getting the independent sets of all our planar subgraphs, we start with a set of stronger candidates that could be part of the maximum independent set. Henceforth, let us call the set we get by combining the independent sets of all our planar subgraphs the "candidate solution set". In addition, removing vertices from the set of candidate solution vertices, every vertex removed effectively eliminates many edges that prevent our set of candidate solution vertices from being an independent set. In the analysis, I will discuss why we can actually expect a large proportion of the vertices in our initial candidate solution set to be included in our final solution.

The first step in our algorithm is to divide the input graph into planar subgraphs. This is done by the following procedure:

```
subgraphs = {}
While graph is not planar:
      graph_copy = deep_copy(graph)
      while graph_copy is not planar:
            delete a random vertex from graph_copy
      subgraphs.add(graph_copy.vertices)
      remove vertices of graph_copy from graph
```

This is the most expensive procedure in the algorithm. The outer while loop takes O(V) iterations as at least a planar subgraph with at least 1 vertex is removed every run. The inner while loop also takes O(V) iterations as only one vertex is removed from the graph copy for every iteration. As each iteration of the inner while loop checks for the graph's planarity in O(N) time, this procedure takes O(NV^2) time and dominates the runtime of the algorithm.

We use Baker's method [Baker,2009] to construct our protocol for finding the maximum independent sets of each of our planar subgraphs. Baker's method, when applied to the maximum independent set problem, is essentially to construct a tree on the planar subgraph and create subproblems (the smallest of which we approximately solve by giving a maximal independent set) by removing the vertices at every ith level of the tree, where we try every value for i from 0 to an arbitrary constant k. For more details on why Baker's method works, see the original paper cited in the bibliography.

The pseudocode for our implementation of Baker's method is as follows:

```
k = 1/ε
Tree = BFS_tree(planar_subgraph)
possible_solutions = {}
For 0 < i < k:
      Tree_Copy = Tree - vertices whose level%k = 0
      this_solution = {}
      For disjoint_chunk in disjoint_chunks(Tree_Copy):
            this_solution.add(maximal_ind_set(disjoint_chunk))
      possible_solutions.add(this_solution)
return largest solution in possible_solutions
```

Once we have applied our implementation of Baker's method to every planar subgraph and combined the results, we reduce the result to an independent set on the original input graph, We do this by removing the vertex in the result with the highest degree on the subgraph of the original input graph containing only vertices still in the result repeatedly, until the vertices left in the result form an independent set on the original input graph. The pseudocode for this last step is best illustrated in context along with the pseudocode for the rest of the algorithm, and it is as follows:

```
planar_subgraphs = get_planar_subgraphs(G)
solution_candidates = {}
for planar_subgraph in planar_subgraphs:
    solution_candidates += baker(planar_subgraph)
priority_map = subgraph(G, solution_candidates).adjacency_matrix
sort priority map descending by length
target = first entry in priority_map
while solution_candidates is not an independent set:
    remove priority_map[target]:
    for entry in priority_map:
        for adjacency in entry:
            if adjacency == target:
                remove adjacency
    target = vertex with longest entry in priority map
return solution_candidates
```

The while loop occurs O(V) times, the outer for loop occurs O(V) times, the inner for loop occurs O(average degree = $V^{0.5}$) times, and map operations occur in constant time if implemented with a hash map. Thus, after the planar subgraphs and their independent sets have been computed, the rest of the algorithm takes O($V^{2.5}$) time.

As a quick proof of correctness, I state that the algorithm does not terminate while the set of candidate solution vertices is not an independent set on the input graph.

Analysis

For every trial in the data shown in this section, a new graph was generated.

Now, we can proceed with a discussion of the algorithm’s performance, as well as our experimental results.

In our experiments, we generated graphs where the average degree of the graph was expected to be $|V|^{(1/2)}$. We did this simply by selecting for each vertex a random integer between 0 and $2|V|^{(1/2)}$ and adding that number of edges at random. From our experimental results, we know with high confidence that the number of planar subgraphs is O($V^{0.573}$) = O($V^{0.6}$) (R^2 = 0.9991 when testing 11 vertex counts from 36 to 1024 vertices with 10 trials per vertex).

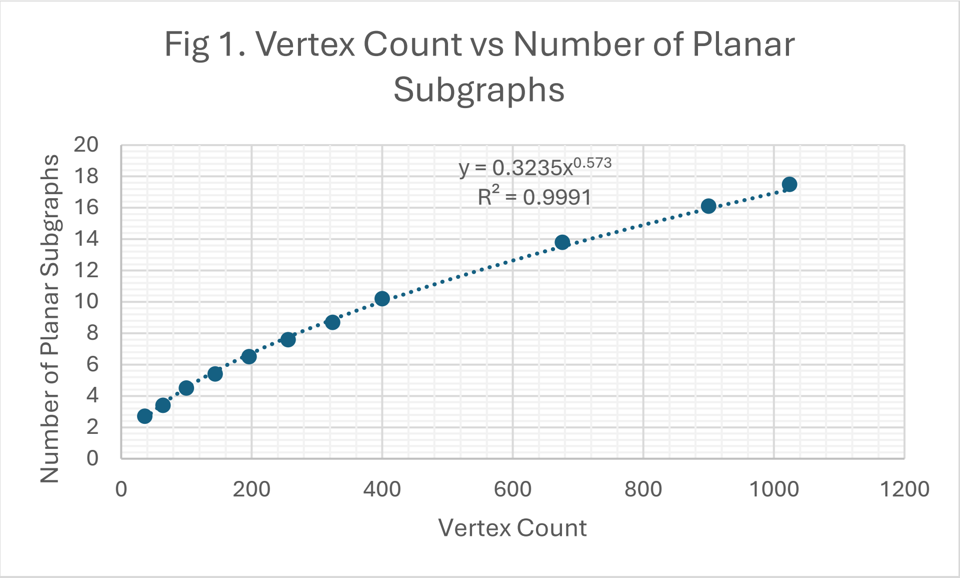


Fig 1. Vertex Count vs Number of Planar Subgraphs

Thus, our planar subgraphs have expected size $V^{0.437}$, as the set of all vertices in all of our planar subgraphs is all the vertices on the input graph. If I am able to come up with a rigorous proof of this result, then the following could be constructed into a proof of the performance of my algorithm. As it stands, the following is a speculative explanation for the good performance of my algorithm that I will discuss later in this section.

Now let us consider the subgraph we get by removing from the input graph any vertex not in our initial candidate solution set (henceforth to be known as the "initial candidate subgraph") . We know that that subgraph cannot contain any edges between vertices that were initially grouped into the same planar subgraph, since our initial candidate solution set consists of the independent sets of our planar subgraphs.

We also know that each vertex on the initial candidate subgraph has expected degree $O(V^{0.5})$, as that was the expected degree on the input graph. As the number of planar subgraphs is expected to be on the order of $V^{0.573}$, this means that vertices on the initial candidate subgraph are likely to be connected to fewer than 1 vertices in every planar subgraph.

Given that the independent sets of each planar subgraph still have much fewer vertices than the whole subgraph, this also means that the in the initial candidate subgraph, each vertex is unlikely to be connected to any vertex in a given planar subgraph.

Suppose that on our planar subgraphs, we expect a proportion of x of the vertices to be in the planar subgraph independent set. Then, for every vertex in our initial candidate subgraph, we expect it to have degree $< V^{0.573}x$ (number of other planar subgraphs * odds of connecting to a vertex in a given planar subgraph). It is important to state that x is expected to be lower bounded by a constant with respect to the size of the planar subgraph [Borodin,1995].

There will be an expected $1/x$ vertices out of a given planar subgraph that make it into our initial candidate subgraph. Recall that in our initial candidate solution set, no vertex is connected to another vertex from the same planar subgraph. Thus, if we sum all the edges in our initial candidate solution set attached to a vertex that belongs to one of our expected $V^{0.573}$ planar subgraphs, we expect to find $1/x\ V^{0.477}\ V^{0.573}x\ = V$ edges.

Therefore, we can expect that our initial candidate subgraph will have around $(V)(V^{0.573}) = V^{1.573}$ edges. We expect O(V) vertices in our initial candidate solution set, since it is the combination of our planar subgraph independent sets, *x* is a constant, and the combination of all our planar subgraphs contain all of the vertices of the input graph.

Let us call the subgraph of the input graph consisting of only the surviving candidate vertices at a given point during our algorithm the "candidate subgraph".

Since we expect a degree of $O(V^{0.573})$ per vertex in the initial candidate solution set, when we first remove one we remove $O(V^{0.573})$ vertices. If removing that vertex did not change the average degree of the candidate subgraph, then we would have to remove $O(V^{(1.573 – 0.573) = 1})$ vertices from our solution set. However, my experimental data shows that this is not true.

Fig 2. Vertex Count vs Solution Size (Average over 25 trials)

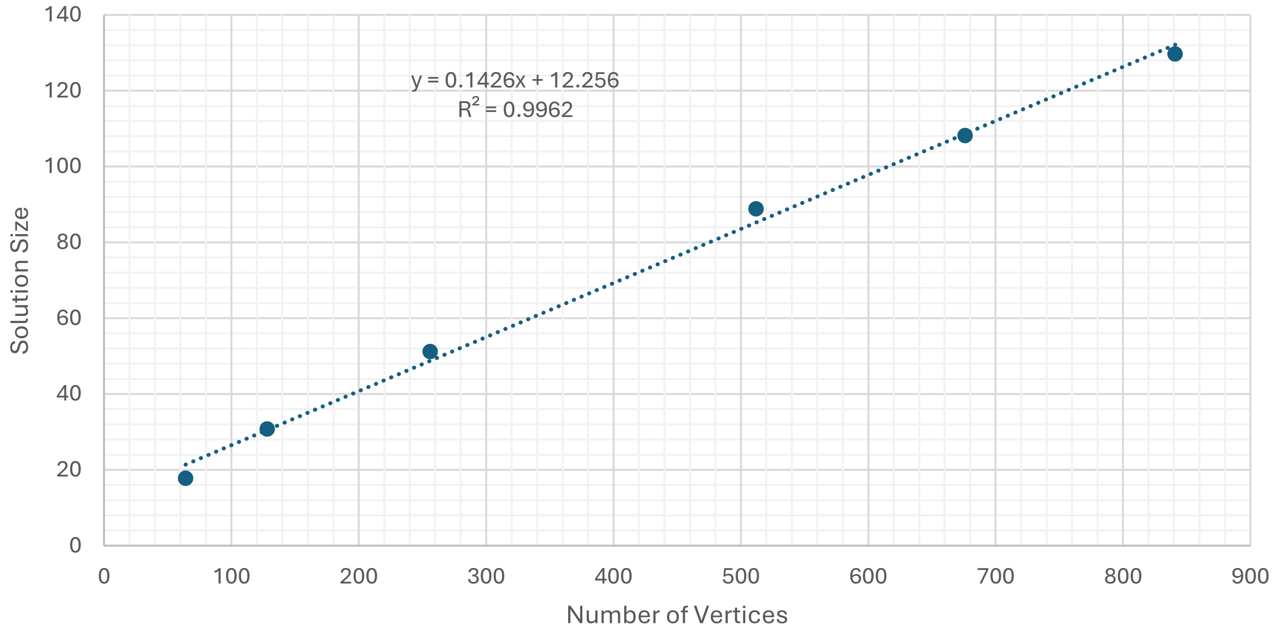
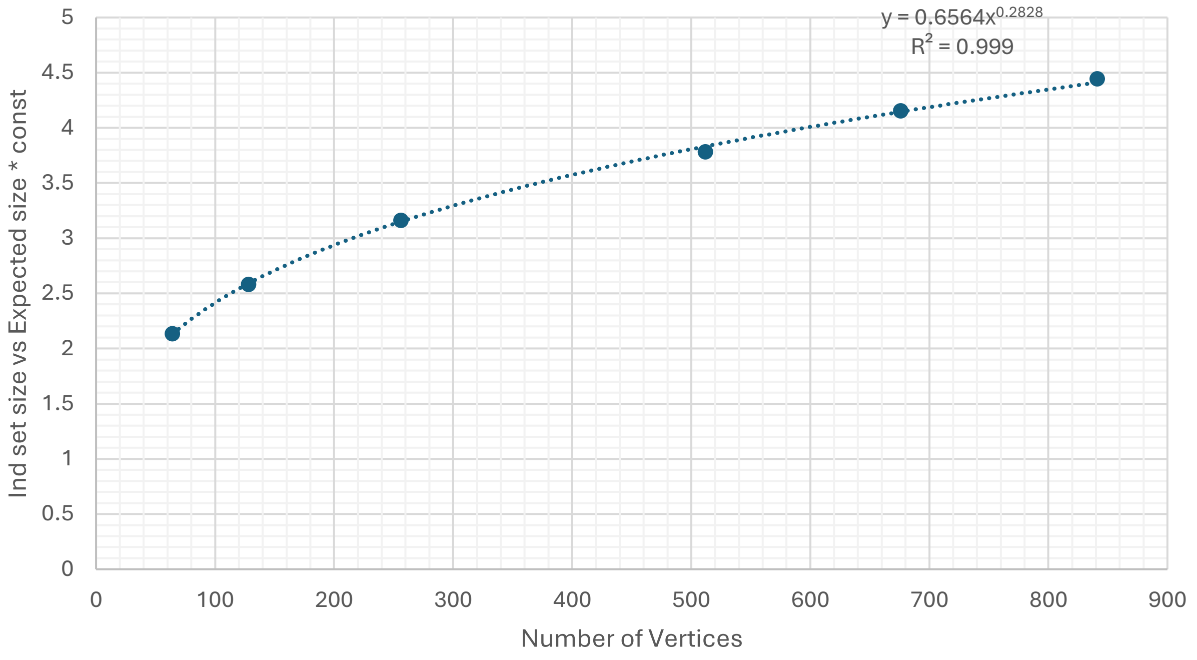


Fig 3. V vs Independent set size over const*expected size (|V|/(average degree)) over 25 trials



It has been established that the maximum independent set size for a graph is at least |V|/(average degree of graph) [John L., 2023]. Thus, we use this as a metric to judge the "score" of our algorithm at a given number of vertices. In figure 3, we can see that the ratio of the size of our output over the expected size of the maximum set increases monotonically with the number of vertices.

If we assume that the true maximum independent set size for a non-planar, somewhat dense graph is O(|V|/(average degree of graph)), this means there is a lower bound for the approximation ratio of our algorithm.

In figure 2, we see that the solution size increases linearly with the number of vertices, which would mean the solution size, like the expected independent maximum independent set size, is O(V) and thus achieves a bounded approximation ratio.

It should be noted that the results in figure 2 would seem to suggest that the assumption that the true maximum independent set size as O(|V|/(average degree of graph)) is pessimistic; it is likely that if we took the true maximum independent set size as the denominator in figure 3 the curve would flatten out considerably.

I have two main ideas as to why my algorithm seems to work.

Firstly, at every step we are removing the vertex in the candidate subgraph with the highest degree, we are probably increasing the degree of the candidate subgraph until a relatively small number of vertices are left in it.

Secondly, the fewer vertices are left in our candidate subgraph, the more likely that all edges connected to any vertex from a given planar subgraph have been removed, at which point all remaining vertexes from that planar subgraph are now safe and will be included in the final solution.

Since we expect more vertices in our solution (O(V)) than we have planar subgraphs ($O(V^{0.573})$), we expect that in our final output we will have $O(V^{0.437})$ vertices from a given planar subgraph. Thus, as we continue removing vertices from our candidate subgraph, groups of vertices from the same planar subgraph start becoming “safe”, and the result is that the output set is a good approximation of the true maximum set.

A naïve non poly-time algorithm for approximate max-independent set would be to start with all the vertices in a graph and removing the highest degree vertex until we get an independent set. However, if we start with a subgraph where we know before hand that large chunks of vertices are independent of each other, we likely get better results than the naïve algorithm.

Conclusion

The experimental results for my algorithm seem promising, and apparently suggest that the algorithm gives a polynomial time approximation on maximum independent set for graphs with average degree = $O(V^2)$. However, further work is needed to rigorously verify and/or theoretically prove these results.

Future Work

1. Success Boosting

Because the planar subgraph generation process is randomized, it may be possible to improve the accuracy of my algorithm by running it multiple times on the same input and taking the best result. This is admittedly relatively easy to try and is thus first on this list. It will probably also improve the results of success boosting if I can find ways to induce more randomness in my algorithm.

2. Further Testing

Because of computational constraints, I was only able to test my algorithm on graphs with a limited number of vertices in a reasonable amount of time. However, not only would using a high performance computing cluster allow me to test my algorithm on bigger inputs, it would also allow me to compare my results with the true maximum independent sets obtained by brute force.

3. Better Analysis

Because of time and knowledge constraints, I was unable to do rigorously analyze my algorithm’s performance from a theoretical perspective. In the future, I may try to understand it better.

Appendix A: Code Usage

The code to generate the data used in figure 2 and figure 3 is the function runExperiment and the driver code that calls it. The code that generates the data used in figure 1 is commented out right below that.

To call the algorithm, do the following:

```
sgs = getPlanarSubgraphs(G,verbose = False)
resultSet = getIndSet(G,sgs,k)
```

Where k = 1/ε. Note that using any k greater than the depth of the tree representing a planar subgraph will yield the same results.

Uncomment lines 325,326,327, and 328 to show the input and outputs for the first trial at each vertex count.

Appendix B: Data Files Description

sizes.xlsx: Data for figure 2.

scores.xlsx: Data for figure 3.

subgraph_count.xlsx: Data for figure 1.

Acknowledgements

Special thanks to Dr. Chen for his advice on my project.
I acknowledge the creators & contributors of the networkx python library for making a comprehensive, well documented, and easy to use library.

Bibliography

[1] Magen, Avner & Moharrami, Mohammad. (2009). Robust Algorithms for on Minor-Free Graphs Based on the Sherali-Adams Hierarchy. 258-271, doi:10.1007/978-3-642-03685-9_20.

[2] Baker, Brenda S. (1983), "Approximation algorithms for NP-complete problems on planar graphs (preliminary version)", 24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7–9 November 1983, IEEE Computer Society, pp. 265–273, doi:10.1109/SFCS.1983.7

[3] Mehlhorn, Kurt; Mutzel, Petra (1996), "On the Embedding Phase of the Hopcroft and Tarjan Planarity Testing Algorithm" (PDF), Algorithmica, **16** (2): 233–242, doi:10.1007/bf01940648

[4] Borodin, O.V. (1995), A new proof of the 6 color theorem. J. Graph Theory, 19: 507-521. doi:10.1002/jgt.3190190406

[5] John L. (https://cs.stackexchange.com/users/91753/john-l), Prove the expected size of the independence set got by a random algorithm is at least 1/d of the maximum size, URL (version: 2023-01-17): https://cs.stackexchange.com/q/156745

[6] Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008