Graph Theory Report

Hugh Frampton HUGHFRAMPTON2@GMAIL.COM
Computer Science
Montana State University
Bozeman, MT 59715, USA

Nathan Ritchlin NSRGY22@GMAIL.COM
Computer Science
Montana State University
Bozeman, MT 59715, USA

Joshua Aney JOSH.ANEY@CLOUD.COM
Computer Science
Montana State University
Bozeman, MT 59715, USA

Editor: Nathan, Hugh, and Joshua

1 Introduction

In the field of computational complexity, there presents a number of challenges, especially when dealing with NP-Complete problems such as finding Vertex Covers and Independent Sets in graphs. These problems are very important when pertaining to the field of computing as they are relevant in topics like computer security or resource optimization. For our project, we developed 4 algorithms: exact and inexact methods for both VC and IS. To evaluate these algorithms, we tested on graphs ranging from 10 to 1500 vertices, evaluating correctness, accuracy, and practical run times. We will report on the details of their implementation, performance, and trade-offs between exact and approximate approaches, giving insights into how they apply in solving NP-Complete problems efficiently under different constraints.

2 Algorithms

2.1 Exact Independent Set

To start we created an independent set verifier that we could use to decide if a subset of vertices was a valid independent set of a graph. We did this by checking that each vertex in the subset did not have an edge over any of the other vertices in the subset. If there were no edges to corresponding neighbors the subset would be declared a valid independent set. This algorithm ran in $O(n^2)$ time, as each node (O(n)) checks all of its edges (O(n)). Once we had a verifier we went about coding the exact independent set algorithm. Our exact Independent Set algorithm consisted of a brute-force search of the graph. We did this by finding the power set of the graph's vertex list. This process ran in $O(2^n)$ time. After

this was found, we checked if there was a valid Independent set in each set. Once we knew which sets were valid Independent sets, we found the set with the maximum size (O(1)). This will definitively be the maximum size-independent set for the specified graph because every possible set of vertices for the graph was tested. The overall running time for this algorithm was $O(2^n)$.

2.2 Exact Vertex Cover

Our exact vertex cover algorithm uses the exact independent set algorithm to find the largest/exact solution for the graph. Then, the complement of that set with respect to the graph is taken as the exact vertex cover, which takes at most O(n) time. We know this works via two principles. Firstly, as explained above, we know our exact independent set is valid. Secondly, as we learned in class, common graph theory states that the complement of the vertex cover of a graph is an independent set and vice versa. Thus, we know our exact vertex cover algorithm is valid and will always produce the minimum sized vertex cover. Because this algorithm uses the exact independent set algorithm, and taking the complement takes O(n), the running time is again $O(2^n)$.

2.3 Inexact Independent Set

Our inexact independent set algorithm was constructed by finding a base independent set, then randomly adding nodes to the base set to check if it was still an independent set. if it was still an independent set we would update our base set and run through a similar process until the base set stopped improving. Once improvement stopped we would return the set.

To find our base set, we started by defining an empty list of nodes. After this was done, we would loop through all the nodes in the graph and for each node we would check to see if there were any nodes in the list of nodes that were adjacent to the current node. If there were no adjacent nodes, we would then add the current node to the list. This part of the algorithm has a running time of $O(n^2)$ because of the nature of going through every edge of every node.

After we have our base set, we would then enter a while loop where we would check if there was a new node that we could add. We did this by looping through every node in the graph, adding it to our base set, and checking if it was a valid independent set. If it was a valid independent set you would update your base set to include the new node. Once we looped through all the nodes in the graph we would check if our base set had increased in size. If it increased in size we would iterate through the loop again, and if it did not increase in size we would break out of the loop and return the set. This part of the algorithm has a running time of $O(n^2)$ where n is the number of vertices, because the while loop can run a max of n times until all the vertices are added, and the inner loop goes through each vertex which ends up being n times. This part of the algorithm also calls the verifier which has a running time of $O(n^2)$.

This algorithm is guaranteed to find a valid independent set for the graph, but there is no guarantee it will find the optimal independent set. This is because it will often find a local optimum that it is not able to recover from. For this implementation, there is no way to avoid this local optimum problem. A way to try to avoid this would be running it

multiple times to see if the local optimum is avoided, but it is very common that it will end up with the same set of vertices. The running time of this algorithm is also polynomial which is far better than the running time of the exact algorithm which is exponential. This allows for much faster calculations of approximate independent sets.

2.4 Inexact Vertex Cover

In a similar methodology to our exact vertex cover, our inexact vertex cover looks at the inexact independent set created in the previous algorithm and takes the complement of that set with respect to the given graph, which takes O(n) time. We know this is valid as our inexact independent set algorithm is valid and by graph theory, the complement of that set will be a vertex cover. However, there is no guarantee that this will return the optimal minimum vertex cover, as the independent set calculated is not necessarily the maximum size. The running time of this algorithm is the same as the running time of the inexact independent set algorithm: O(n^2)

3 Evaluation Description

In our experiment, we tested a variety of metrics. We compared theoretical run time, actual run time, size of sets, and accuracy across all of our algorithms for the independent set problem and the vertex cover problem. We tested this by running each inexact algorithm on graphs of different sizes in a range of 10 to 1500 vertices, and a range of 5 to 25 vertices for our exact algorithms. We then tested each of these sizes 50 times on our inexact algorithms to see how the algorithm would perform with large graphs. For our exact algorithms, we only ran our algorithms on one graph because the algorithms tended to take a significant period of time to run. The way we generated our graphs is by taking in a set number of vertices and then we would allow for a 30% chance that each vertex would have an edge to every other vertex in the graph. This allowed for a fairly dense graph to be created. This would push the algorithm because of the pure amount of edges that would be created.

4 Experiment Results

In this section, we will show our results and figures for each algorithm's. We will also show the size of the set as well as the run time.

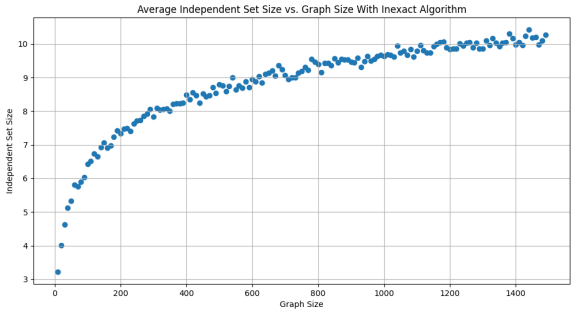

Figure 1: Average size of the inexact independent set compared to the size of randomly generated graphs.

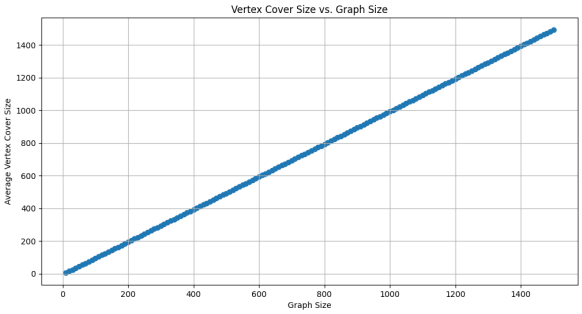

Figure 2: Average size of the inexact vertex cover compared to the size of randomly generated graphs.

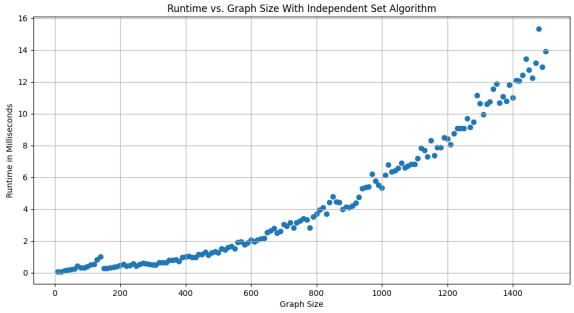

Figure 3: Average runtime of the inexact independent set algorithm on randomly generated graphs.

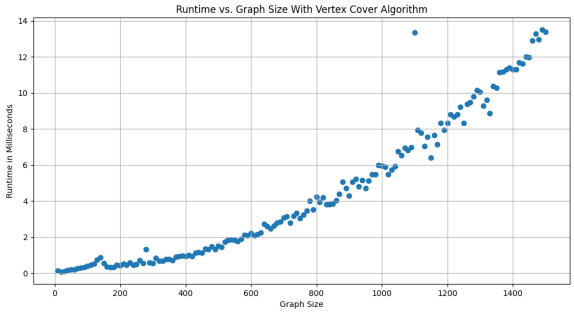

Figure 4: Average runtime of the inexact vertex cover algorithm on randomly generated graphs.

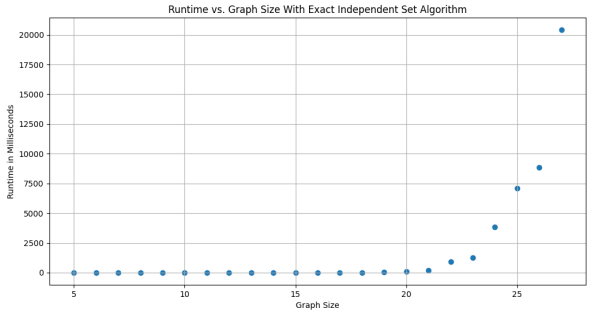

Figure 5: Average runtime of the exact independent set algorithm on randomly generated graphs.

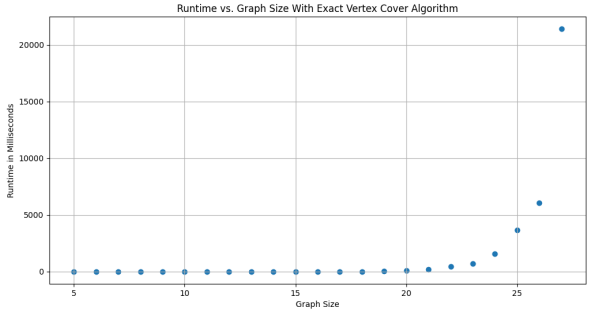

Figure 6: Average runtime of the exact vertex cover algorithm on randomly generated graphs.

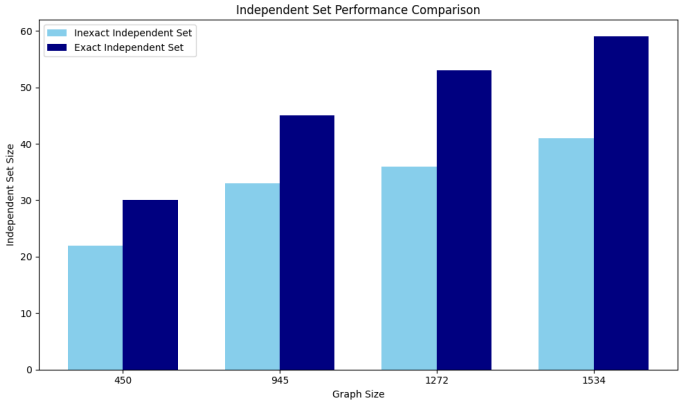

Figure 7: Comparison of independent set size between inexact and exact algorithms on the provided large graphs.

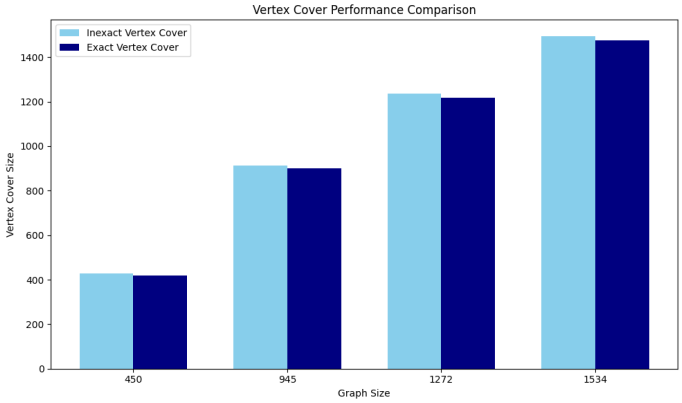

Figure 8: Comparison of vertex cover size between inexact and exact algorithms on the provided large graphs.

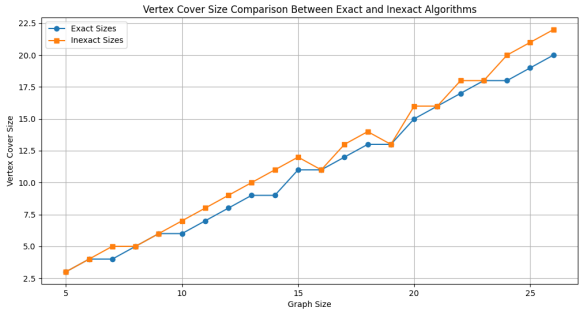

Figure 9: Comparison of vertex cover size between inexact and exact algorithms on generated graphs.

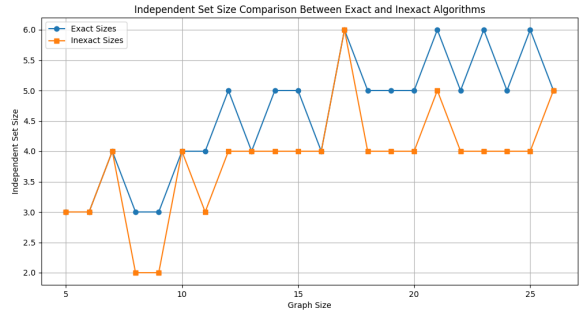

Figure 10: Comparison of independent set size between inexact and exact algorithms on generated graphs.

5 Summary

In this project, we explored how practical it would be to solve NP-Complete problems when implementing both exact and inexact algorithms for finding Vertex Covers(VC) and Independent Sets(IS). These implementations were tested over 50 iterations on graphs ranging from sizes of 10 to 1500 with the generated graphs being fairly dense, and the results gave us insight into the trade-offs between algorithmic accuracy and computational efficiency.

5.1 Exact Algorithms

The exact algorithms for VC and IS provided very good solutions, which was guaranteed by their exhaustive approach. However, due to the time complexity of this exhaustive approach $O(2^n)$, these algorithms were the best when used on smaller graphs, typically under 25 vertices. If the graph were to go beyond this size, the runtime would quickly become too much for the algorithm to be practical.

5.2 Inexact Algorithms

The inexact algorithms showed how to provide valid, however suboptimal, solutions in polynomial time. For our Independent Sets, the inexact approach used a strategy that iteratively improved over time, which balanced the computational efficiency with algorithm accuracy. For Vertex Covers, leveraging an inexact Independent Set and taking its complement. This also leads to a valid, yet suboptimal, solution. These algorithms proved practical applications of these algorithms on larger graphs where approximate solutions are acceptable.

5.3 Key Findings

To start, the accuracy and optimality of the exact algorithms consistently produced optimal solutions, whereas the inexact algorithms achieved near-optimal solutions for small to medium-sized graphs. However, as graph sizes grew, accuracy began to diverge due to the local optimal solutions. Next, the scalability of the algorithms. Our exact algorithms showed exponential growth in runtime, becoming impractical for graphs with more than about 30 vertices. In contrast, the inexact algorithms maintained a good runtime even when our graphs reached sizes up to 1500 vertices. Finally, the runtime of exact algorithms required exponential runtime, whereas inexact ran within milliseconds for graphs of practical size, making them significantly more efficient.

5.4 Conclusion

In conclusion, while our exact algorithms provided theoretical guarantees, they were limited in practicability due to computational constraints. Our inexact algorithms, despite their limitations in optimality, offered a good alternative for dealing with large-scale problems efficiently. This balance between precision and performance underscores how important it is to choose algorithms based on the requirements and constraints of the problem at hand.