Heuristics of the graph covering problem

JOSEPH VICTOR WIJNAND VORAGE, University of Twente, The Netherlands

ABSTRACT

The graph covering problem is a lesser known problem in graph theory. It asks whether there is a neighborhood preserving mapping from the vertices of a graph G to a graph H. It has many similarities to the graph isomorphism problem and in some ways can be viewed as an extension to it. Research has been done into the complexity of the problem for several different classes of fixed based graph H. Additionally, the graph covering problem has been proven to be NP-complete for free base graph H. However, this research is spread amongst different papers and is expressed purely in mathematical form, without example code or pseudocode given to solve the problem. This paper shows a group of techniques which have been collected from existing research or created to solve the problem through a computer algorithm. Additionally, the results of this algorithm will be presented on a range of differing base graph structures to gain insight on its efficiency for these differing structures. The result is a working algorithm which solves the graph covering problem for any two input graphs G and H. The research also includes considerations for disconnected base and covering graphs which serves as an extension to what is found in current research. The benchmarking results show that for a great many cases of graphs, the graph covering problem can be solved in polynomial time even for slightly more symmetric structures, though graphs which are currently known to be difficult to solve in the graph isomorphism problem are generally difficult when extended to the graph covering problem. This research therefore shows that the mathematical research which has been done into the graph covering problem can be used into a working computer algorithm to solve it and serves as a practical confirmation of the theory.

Additional Key Words and Phrases: graph theory, graph covering, algorithm, complexity

ACM Reference Format:
Joseph Victor Wijnand Vorage. 2023. Heuristics of the graph covering problem. 1, 1 (July 2023), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The graph covering problem is a problem which has already been around in the field of graph theory for quite some time. N. Biggs covered the existence and some aspects of this problem [2]. As it has been researched, it takes a fixed graph H and admits a given graph G and asks whether there exists a neighborhood preserving mapping from the vertices of H onto the vertices of G. A covering graph of a connected graph is sometimes referred to as a "lift". The problem is known to be NP-complete for some fixed H [4]. For such base graphs graphs, the computation time scales exponentially with the size of the covering graph. Since the graph covering problem is NP-complete for these fixed H, the problem as it is researched here, admitting free H and free G is therefore NP-complete as well.

Author's address: Joseph Victor Wijnand Vorage, j.v.w.vorage@studente.utwente.nl, University of Twente, P.O. Box 217, Enschede, The Netherlands, 7500AE.

That being said, the complexity of the problem depends strongly on the structure of the base graph. Though the problem as a whole has been proven to be NP-complete, there exist structures of the base graph H such that the problem can be solved polynomially [4]. These proofs do not however come accompanied with a working algorithm for solving the problem.

This paper puts forth an overview of heuristics for solving the graph covering problem from existing research. Along with this, an algorithm for solving the graph covering problem with these heuristics included will be given together with the results of its performance on varying classes of graphs.

With this, this research attempts to create a practical implementation of theoretical research which has been done into the complexity of the graph covering problem. Since it is impossible to test such an algorithm on every possible graph, considerations are made as to which classes of graphs to include. Additionally, a benchmarking set used to test algorithms for the graph automorphism problem is used for this algorithm. Some well-known graph structures such as trees are included, as well as many graph structures which are known in the field of graph isomorphism to be difficult to solve. Additionally, an algorithm is included to produce random graphs in order to test the algorithm on various cases.

2 NOTATION

This paper makes use of shorthand notations for different concepts related to the graph covering problem. The base graph is generally referred to as H, and the covering graph is referred to as G. The set of vertices of any given graph G is written as $V(G)$. The number of vertices in a given graph, $|V(G)|$ will be referred to as the 'order' of the graph and denoted $O(G)$. Similarly, the set of edges of any given graph G is written as $E(G)$ and the number of edges contained in this set $|E(G)|$ is referred to as the 'size' of the graph, denoted $S(G)$. The 'neighborhood' of a given vertex v is denoted as $N(v)$ and denotes all vertices connected directly to this vertex through an edge. When denoting the neighborhood of a vertex in a specific graph G, it is denoted as $N_G(v)$. The size of the neighborhood $N(v)$ will be referred to as the 'degree' of that vertex and is denoted as d_v.

Definition 1. *The graph covering problem asks whether there exists a neighborhood preserving bijection f of the vertices $V(G)$ to the vertices $V(H)$, such that for every $v \in V(G)$, $f(N_G(v)) = N_H(f(v))$.*

Problem 1. *Given two graphs G and H, does there exist a neighborhood preserving mapping f of the vertices $V(G)$ to the vertices $V(H)$, such that for every $v \in V(G)$, $f(N_G(v)) = N_H(f(v))$. Produce a mapping if it exists.*

3 PARTITION REFINEMENT

3.1 Refinement

Degree partition refinement can be used to help solve the graph covering problem more efficiently, and is used similarly for solving the graph isomorphism problem[6]. The blocks of a partition will

be denoted as $B_1, ...B_n$, while the block a specific vertex v belongs to will be denoted as $B(v)$

Definition 2. *The degree partition of a graph is the partition of the graph's vertices into the minimum number of blocks $B_1, ..., B_n$ for which there are constants r_{ij} such that for each i, $j(1 \leq i, j \leq n)$ each vertex $v \in B_i$ is adjacent to exactly r_{ij} vertices in B_j[4].*

Theorem 1. *If G covers H, and G is connected, then for each $v_1, v_2 inV(G)$, one has $B_G(v_1) = B_G(v_2)$ if and only if $B_H(f(v_1)) = B_H(f(v_2))$.*

In a covering projection between two connected graphs, each vertex $v \in V(G)$ must map to n vertices in $V(H)$, where $n = O(G)/O(H)$. This means that each block in the partition set of G must be of size nm, where m is equal to the size of the corresponding partition block in H.

An important conclusion to draw from this is that if any block of G is not of size nm, then we can simply conclude that G is in fact not a covering of H. This means that if we can compute the stable partition of both graphs, we can compare the block sizes of each graph. This potentially rules out the possibility of a covering altogether. If it does not rule out the possibility, it often shrinks down the number of possibilities for a covering which needs to be checked as described in the next section.

One can reach the degree partition through a process of partition refinement. By initially colouring each vertex according to its degree. After this, the partition will be further refined based on each vertex and its neighbours' partition groups.

An algorithm for efficient partition refining exists created by John Hopcroft [1] which has a worst case complexity of $O(ns \log n)$, where n is the order of the graph and s is the size of the graph, for providing the degree partition of a graph. Hopcroft's algorithm is used to establish the degree partitions of the base and covering graphs in this research.

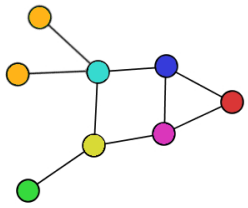

Fig. 1. Stable partition for an example graph

3.2 Finding a solution

The most naive way of solving the problem is to simply generate all possible covering projections and test each of them individually. Initially one could speculate that each vertex in G could possibly map to any vertex in H. A simple restriction one could then apply is limiting each $v \in V(G)$ to only possibly map to a $v' \in V(H)$ such that $d(v) = d(v')$.

What degree partitioning offers us is a stronger restriction. Any $v \in V(G)$ can only possibly map to any $v' \in V(H)$ if v and v' are in corresponding partition blocks. Showing that G and H have a matching stable partition is not enough to prove that a covering projection exists. It does, however, shrink down the problem significantly as each vertex $v \in B_{Gi}$ must map to one of the vertices $v \in B_{Hi}$ which means a significantly lower amount of vertices need to be considered as candidates.

Unfortunately, for some specific classes of graphs, such as connected graphs of k-regular graphs, the partition refinement algorithm does not shrink the problem down. In the specific case of regular graphs, the partition is stable after the first iteration of degree colouring.

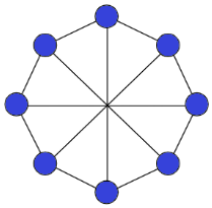

Fig. 2. The degree partition of a 3-regular graph of order 8

In the case of more symmetric graphs, there possibly exist other heuristics depending on the base graph's structure which may help solve the problem. But if no such options are available, the next step to resort to is simply trying out all valid options to see if one of them results in a valid covering. The strongest factor for the size of the labeling options depends primarily on the sizes of the partitions in the base graph and secondarily on the order factor n between the base graph and the covering graph. For each block $B'_1, ..., B'_n \in B(G)$ and corresponding $B_1, ..., B_n \in B(H)$, there are $\frac{(mn)!}{(n!)^m}$ options for mapping the vertices of the block in H to the vertices of the block in G with. In the cases where the size of the block in the base graph is equal to 1, there is only one possible mapping from the block of the covering graph to that of the base graph. This is why the partitioning is so powerful for base graphs which are not very symmetrical. However, for large block sizes, the problem insurmountable as the sizes of these partitions grow, especially when considering each other partition has labeling options which scale in a similar manner. However, in many cases, the problem is not as sizable as described here. After each step of choosing some m vertices of a block B'_i to map to some vertex of B_i. We can create new partition blocks B'_{n+1} containing the m vertices in G and a new partition block B_{n+1} containing the vertex mapped to in H. With these new partition blocks created, we can run Hopcroft's algorithm to refine the partition according to this choice. In many cases, this quickly causes a choice to be written off as the new partition refinement does not align for the two graphs, or it may at least further refine the partition so that fewer options remain for testing this particular path. On top of this, when choosing which vertices to assign to some new partition, we can choose the smallest partition set to choose from. This has the least possible options for this particular set while possibly allowing the options to become significantly smaller through running Hopcroft's algorithm after an assignment has been tried. These methods work to significantly limit the bruteforcing work required on many different graph structures.

This method however still faces issues when it comes to highly symmetrical graph structures such as for example regular graphs as seen in figure 2. For these structures the partition groups are generally very large and as such this method of bruteforcing grows extremely quickly.

4 HEURISTICS

4.1 Trees

If the base graph for the graph covering problem is a tree, the problem can be solved in polynomial time.

Theorem 2. *If H is a tree, G is a connected graph, and there exists a covering projection f from G, to H, then G must be isomorphic to H[4].*

We can use this fact, together with the fact that the graph isomorphism problem is solvable in polynomial time for trees[3] to show that the graph covering problem can be similarly solved for trees in polynomial time.

We can check if a given graph G is a tree in two steps. First, we determine whether the graph is connected by using for example breadth-first search from any given vertex as the starting point in the graph. Once we establish that the graph is connected, we can simply check whether $S(G) = O(G) - 1$. If both of these are true, we know that the given graph is a tree. Both of these steps can be performed in polynomial time.

Once we have established that both H and G are trees, we can simply test whether the trees are isomorphic by first finding the middle point of the tree by pruning the leaves iteratively, and then comparing the individual branches of the tree.

4.2 Unicyclic base graphs

Unicyclic graphs are a simple graph structure which expand upon the tree structure slightly. A unicyclic graph is any connected graph with $O(G) = S(G)$, meaning the number of edges and vertices are equal. A unicyclic graph can structurally be viewed as a single cycle structure with each vertex of the cycle having zero or more tree structures attached to it. One can construct a unicyclic graph by simply adding a single edge to a tree structured graph. With the size factor between the base graph and covering graph $n = O(G)/O(H)$, any covering graph of such a base graph contains a cycle with order equal to that of the base graph's cycle multiplied by n. Additionally, the covering graph must contain n copies of each of the trees attached to the base graph's cycle.

Theorem 3. *The graph covering problem can be solved in polynomial time if base graph H is unicyclic [4].*

Establishing whether a given graph G is unicyclic can simply be done by first checking for connectedness through breadth-first search and then verifying that $S(G) = O(G)$.

Once established, the problem can be solved as follows. First, seperate the trees and the cycle for both H and the covering G. This can be done by iteratively removing each vertex of a graph if its degree is equal to one. Repeatedly doing so prunes a unicyclic graph down to its cycle in polynomial time. Once the cycle has been obtained, the trees can be identified by taking the original graph, removing the cycle, and identifying the disjoint trees through breadth-first search.

Once the trees and cycle have been separated, verify that the size of the cycles in H and G match the ratio n. After this, establish the isomorphisms between the trees of H and the trees of G using the procedure described in the previous section. If any of the trees in G are not isomorphic to all the trees in H, then there exists no covering projection. If there exist isomorphisms for each tree in G, establish whether the trees are connected at corresponding locations in both the base graph in the covering graph. If this is the case, it is established that graph G covers graph H. Otherwise, there exists no covering projection.

4.3 Reduction to 2-satisfiability

Based on the result of the degree partition, some shortcuts can be possible. Specifically, if each block of the degree partition of H contains at most two vertices, the problem can be transformed into the 2-satisfiability problem in polynomial time. The resulting 2-satisfiability problem can then be solved in polynomial time to produce a solution.

Theorem 4. *If, for a given base graph H, $\nexists v_1, v_2, v_3 \in V(H)$ such that $B_H(v_1) = B_H(v_2) = B_H(v_3)$, then the graph covering problem is solvable in polynomial time.[4]*

This transformation is realised by transforming each connection between partition sets within the graph into a decision of assigning vertices a value of true or false.

For cover graph G, with degree partition blocks $B'_1, ...B'_n$ and base graph H, with degree partition blocks $B_1, ...B_n$, the 2-satisfiability problem is constructed as follows.

First, label the vertices of each block $B_1, ..., B_n$ with a true or false value. If the block contains only one vertex, label it true. Otherwise, if the block contains two vertices, label the first T and the second F. Then construct the 2-satisfiability for G. The 2-sat problem is constructed based on whether a vertex in G should map to the true or false node in the base graph.

Rules are constructed for each of the vertices in G based on their connections to other blocks as well as the way those corresponding blocks connect in the base graphs [4].

Once the problem has been reduced to the 2-satisfiability problem it can be solved in polynomial time. And since the reduction is in polynomial time as well, this means that for these graphs the problem as a whole can be solved polynomially.

4.4 Disconnected graphs

In the previous sections, this paper has covered cases for connected H and connected G. In order to solve the graph covering problem for disconnected H or disconnected G, additional considerations must be made. First considering disconnected base graph H with connected subgraphs $H_1, ..., H_n$ and connected covering graph G. The graph covering problem is polynomially solvable for disconnected H if and only if the problem is polynomially solvable for each disjoint subgraph of H [4].

Theorem 5. *For disconnected base graph H with the set of separated disjoint subgraphs $H_1, ..., H_n$ denoted $C(H)$ and connected covering graph G. $(\exists H_i \in C(H))(G \text{ covers } H_i) \implies G \text{ covers } H$.*

This is not found elsewhere in the literature. The mathematical proof for this theorem falls outside the scope of this research. There may exist covering projections from G to multiple $H_i \in C(H)$ but in order for G to cover H there need only be a single one. Notably, for such a disconnected graph, each $v_1 \in V(G)$ must map to some $v_2 \in V(H)$, but not all $v \in V(H)$ need to be mapped to.

Now considering connected base graph H and disconnected covering graph G.

Theorem 6. *For connected base graph H and disconnected covering graph G with the set of separated disjoint subgraphs $G_1, ..., G_n$ denoted $C(G)$. $(\forall G_i \in C(G))(G_i \text{ covers } H) \implies G \text{ covers } H$.*

Each disjoint subgraph of G here can be seen as a separate graph, all of which must individually cover H for G as a whole to cover H.

Combining these two theorems into a final theorem for disconnected G and disconnected H.

Theorem 7. *For disconnected base graph H with the set of separated disjoint subgraphs $H_1, ..., H_n$ denoted $C(H)$ and disconnected covering graph G with the set of separated disjoint subgraphs $G_1, ..., G_n$ denoted $C(G)$. $(\forall G_i \in C(G) \exists H_i \in C(H))(G_i \text{ covers } H_i) \implies G \text{ covers } H$.*

These theorems can be applied to the algorithm in a simple manner. Simply split the base graph and covering graph up into the set of their disjoint subgraphs and verify whether for each graph in the covering set, there exists a graph in the base set such that the algorithm produces a correct covering projection.

4.5 Comparison to isomorphism problem

As noted several times in this paper, the graph covering problem shares many of its aspects with the graph isomorphism problem. The graph covering problem can be seen in many ways as an extension to the graph isomorphism problem. Notably, the graph isomorphism problem is fully covered itself by the graph covering problem already and is a strict subset of it. To show this, simply take a base graph H of order n and a covering graph G of order n. If G covers H, then G is an isomorphism of graph H. The differences in solving the graph covering problem therefore mostly come as extensions to the algorithms which exist for the graph isomorphism problem instead of introducing new techniques. However, some classes of graphs such as complete base graphs, and complete graphs minus one edge are difficult to solve for the graph covering problem, while being significantly easier when it comes to isomorphisms. Additionally, there exist some heuristics for regular graphs for the graph isomorphism problem which are not easily translatable to the graph covering problem.

4.6 Resulting algorithm

The resulting algorithm applies the heuristics in the following order:

- Confirm $(|V(G)|/|V(H)|) \in \mathbb{Z}^+$
- Check if both graphs have a tree structure and apply tree heuristic if applicable
- Check if both graphs have a unicyclic structure and apply unicyclic heuristic if applicable
- Compute partition blocks and confirm size factors
- Check if the largest partition block of H has size ≤ 2. Reduce to 2-sat and solve if applicable
- Establish the mapping to one $v \in V(H)$ and repeat partitioning function
- Repeat previous step until a solution is found or determine there is no covering projection if all options have been tried

5 BENCHMARK

This section will review the results the algorithm has obtained on different classifications of graphs. Firstly, we will go over the performance of some graphs which follow specific structures for which heuristics have been implemented, such as trees and graphs of maximum partition size two. Then we will, the algorithm's performance will be shown for a benchmarking set used by the University of Twente for testing algorithms for graph isomorphisms and automorphisms. From these, double and quadruple covers were generated to test the performance as the size of the covering graph increases. Since these benchmarking graphs are used for testing automorphism algorithms, they generally contain some level of symmetry, which will likely be difficult graph structures for the algorithm to solve.

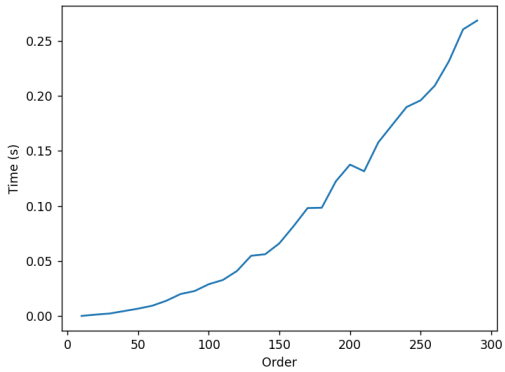

Fig. 3. Computation time for random graphs generated with $S(G) = O(G) \cdot 2$

5.1 Random graphs

The complexity of random graphs depends heavily on how many edges the base graph contains. A large amount of edges generally means that there is more symmetry present and as such, the block sizes of the partition refinement will be larger. This means that as cover graphs get larger the problem scales exponentially. On the other hand, a lower amount of edges generally means that the graph is easier to solve and often falls in the category of blocks with maximum size of either 1 or 2. The complexity for these two cases are displayed in the figures 3 and 4. The random graphs are generated by creating some input number n vertices, creating a Prüfer[5] sequence to create a random spanning tree, and adding additional edges up to some input size m.

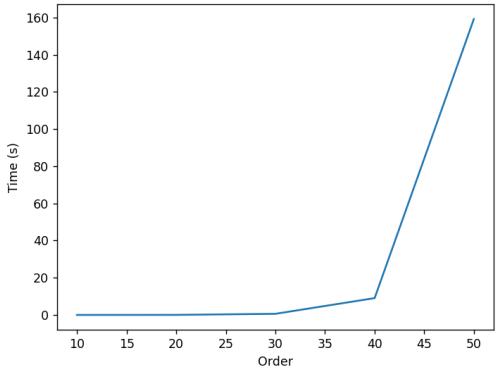

Fig. 4. Computation time for random graphs generated with $S(G) = \frac{O(G)\cdot(O(G)-1)}{2} - 2$

5.2 Trees

As can be seen in figure 5, the algorithm performs efficiently when solving tree-structured graphs as is expected. The complexity can be seen to scale polynomially as is congruent with the theory on tree isomorphisms.

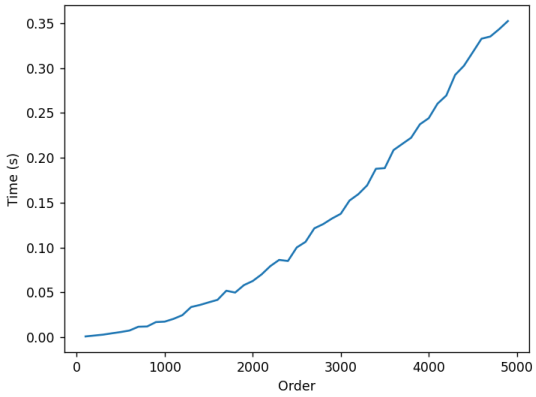

Fig. 5. The computation time for tree-structured graphs

5.3 Partition block size two

To test this benchmark, a small group of base graphs with order 6 with stable partition sets which contain blocks of both size 1 and size 2 were used. From these base graphs, 2^n covers were generated with $n \in 1, ..., 11$. The algorithm was similarly able to solve the problem for graphs of very high orders, scaling polynomially with the order of the covering graph. The results can be seen in figure 6.

5.4 Benchmark set

The data for the benchmarking set will instead be shown in a table where each file will have its computation time denoted for the discovery of a 1x cover, a 2x cover and a 4x cover. This table shows the

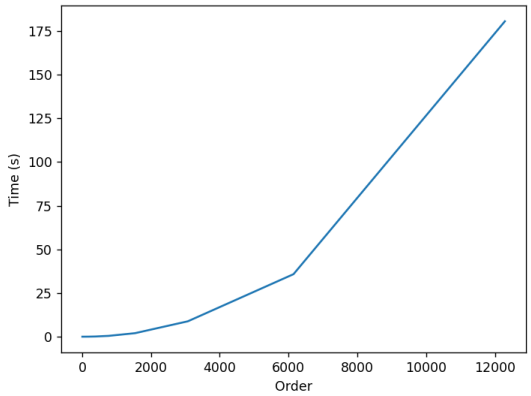

Fig. 6. The computation time for base graphs with partition size ≤ 2

Table 1. Benchmark performance

File name	1x cover	2x cover	4x cover
auttreeA17_1	0.002	0.02	0.06
badcref8_1	2.8	11.2	850.2
cycles350_1	26.8	65.8	218.43
double_auttreeB_88_1	0.35	0.8	3.2
hugecographsE_1	14.5	27.7	84.9
hugecographsF_1	0.26	2.4	25.1
Miyazaki4_1	0.02	0.03	1.8
pseudocographs3_1	0.06	0.09	0.23
tori60_4_1	0.7	1.47	4.2
wheelstar16_1	0.006	0.004	0.2

computation time for the graph when a single, double or quadruple cover is taken for that graph.

Based on the benchmarking data, the only graph structure that seems to fall in the area of scaling exponentially is badcref8_1. Notably for this graph, none of the graphs cover one another and as such every single possibly must be exhausted before the algorithm can confirm whether the graph is a covering or not. It is notable that many of the graphs do seem to scale polynomially, though more testing would be required to assert this. This could be due to that they are covered by Hopcroft's algorithm, have a tree structure, or have maximum block size of 2 for their partitions.

6 CONCLUSION

In this paper is shown the results of a working algorithm for the graph covering problem and a collection of the heuristics from research which can be used for said algorithm. The main heuristics used and presented in this paper are partition refinement through Hopcroft's algorithm, tree isomorphism testing through symmetry from the center, reduction to 2-sat for graphs with partition blocks of size ≤ 2 and an application of the tree symmetry and graph reduction to a cycle for unicyclic graphs. Finally, an extra piece presented here not found in current research is the consideration for disconnected graphs, solved by dividing the graphs up into disjoint parts. The proof for how to solve the problem for disconnected graphs is

not found in current research and the algorithm to solve such cases is therefore a notable addition from this research. The produced result is a flexible algorithm which allows for both connected and unconnected graphs to be tested for the existence of a covering projection. The algorithm works well on a large number of differing graph structures, though it may struggle on more symmetric structures such as regular graphs and complete graphs.

7 FURTHER RESEARCH

Further research could be done into adding additional heuristics to such an algorithm. This research focused specifically on the pieces of literature which were somewhat more straightforward or realistic to be applied to a computer algorithm. However, there exist more base graph structures which are proven to be solvable in polynomial time through more complicated proofs. If one was able to transform those proofs into a computer algorithm, it could be added to what is presented here in order to expand upon the number of structures which the algorithm covers. There exist proofs for graphs which only have two vertices of degree ≥ 2 as well as graphs which have all but two vertices having degree of 2 which could possibly be transformed into an algorithm [4]. Additionally, further research could be done in creating specific benchmarking graphs which challenge such an algorithm in interesting ways which are tailored specifically to the graph covering problem.

REFERENCES

[1] Alfred V Aho and John E Hopcroft. 1974. *The design and analysis of computer algorithms*. Pearson Education India.
[2] Norman Biggs. 1974. Algebraic graph theory. https://doi.org/10.1017/CBO9780511608704
[3] Scott Fortin. 1996. The graph isomorphism problem. (1996). https://doi.org/10.7939/R3SX64C5K
[4] Jan Kratochvíl, Andrzej Proskurowski, and Jan Arne Telle. 1995. Complexity of graph covering problems. *SpringerLink* (Jan 1995). https://link.springer.com/chapter/10.1007/3-540-59071-4_40
[5] Heinz Prüfer. 1918. Neuer Beweis eines Satzes über Permutationen. *Archiv der Mathematischen Physik* 27 (1918), 742–744.
[6] Stephan Schwartz. 2022. An overview of graph covering and partitioning. https://www.sciencedirect.com/science/article/pii/S0012365X22000905