Cops And Robbers on Graphs: A Survey

Hesed Guwn, Sam Munoz, Claire Yu, Ming Zhang

September 18, 2024

Abstract

This paper explores the optimization of the game Cops and Robbers by reviewing and analyzing various algorithms and winning strategies for both the cop and the robber. Such algorithms and winning strategies include: determining default Cop-Win Graphs, Cop-Win Graphs with multiple cops, Cop-Win graphs with a faster robber, and Cop-Number of graphs with a known treewidth.

1 Introduction

In graph theory, Cops and Robbers is a two player pursuit-evasion game inside a finite un-directed graph, where the goal of the game is for the cop to capture the robber by occupying the same vertex as the robber. The robber's objective is to try and evade capture indefinitely. To play the game, the cop player begins at a vertex of its choosing, and then the robber chooses its vertex. From there, the cop moves to an adjacent vertex or can stay put, and then the robber can either move or stay in the same vertex. The players repeat their turns until the cop captures the robber, or the robber avoids capture forever. For example, a cop would win in a cop-win graph where there is only one cop and one robber placed in a graph shaped like a triangle - three vertices and three edges (Figure 1). As the cop can get the robber no matter where it moves. An example of a robber-win graph is simply a rectangle of four vertices and four edges, as the robber can choose its initial position at the opposite corner of the cop and move away from the cop indefinitely.

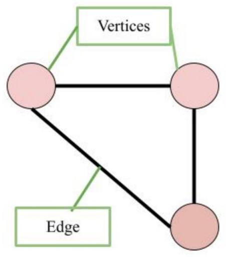


Figure 1: A graph with vertices and edges. This graph, if there is one cop and one robber, is cop-win since the cop can always capture the robber.

Terms that are consistently used throughout the paper include: **cycles, neighborhood**, **dismantlable graph**, and **dominant**.

Definition 1.1 (Cycle)**.** A looping path in a graph that contains four or more edges with the edges only connecting to the vertex ahead.

Definition 1.2 (Neighborhood)**.** A collection of adjacent vertices connected to the vertex that is currently being looked at, including itself.

Definition 1.3 (Dominant)**.** A vertex is dominant over another if it contains the neighborhood of the other vertex.

Definition 1.4 (Dismantlable Graph)**.** A graph is dismantlable if there is a sequence of vertices which are all dominated by a vertex remaining in the graph after all previous vertices are removed until one vertex remains.

Definition 1.5 (Level of Domination)**.** For vertices v_i and v_j, if v_j dominates v_i, then $v_j > v_i$ and it has a higher level of domination than v_i.

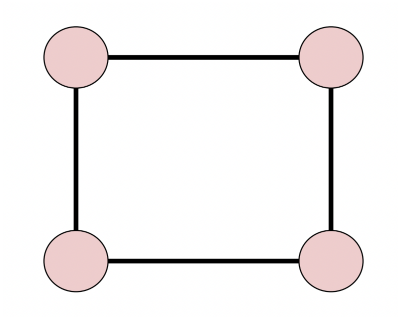


Figure 2: A rectangle is an example of a cycle of four vertices. This graph, if there is one cop and one robber, is robber-win since the robber can indefinitely evade the cop.

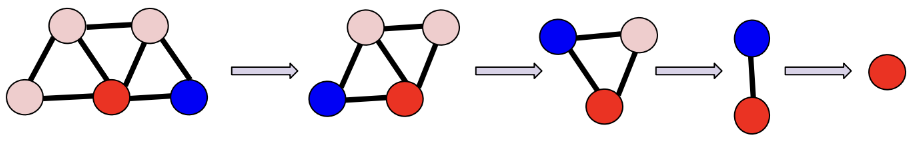


Figure 3: An example of a dismantable graph, where there exists a sequence of vertices that are all dominated by the red vertex such that every time the graph dismantles, a blue vertex gets removed until the red vertex is left.

In the following paper, we survey previous results for what constitutes as Cop-Win Graphs, Cop-Win Graphs with multiple cops, Cop-Win graphs with a faster robber, and Cop-Number of graphs with a known treewidth.

2 Determining Cop-Win Graphs

To determine whether a graph is cop-win or robber-win for one cop and one robber, we need to determine whether the graph is dismantlable or not. Through the definition of domination and the process of dismantling, we can "remove" vertices that are dominated by other remaining vertices, until we are left with only one or more vertices. In the case of only one vertex, the cop can use chase algorithms to eventually capture the robber, as there are no methods for the robber to indefinitely extend its escape. However, if there are multiple vertices left after dismantling, the robber can indefinitely play on such vertices and will always escape from the cop. So, one vertex remaining after dismantling would mean the graph is cop-win, and multiple remaining would be robber-win.

Theorem 2.1. *A graph is robber-win if more than one vertex remains after dismantling.*

Proof. Suppose that a graph after dismantling contains more than one vertex, and that the cop is placed on a vertex and robber on a vertex that is not adjacent to the cop. This is possible because none of the vertices dominates each other, hence why they are not removed during dismantling. So these vertices are not adjacent to all of the remaining vertices.

While the game plays, the robber is always able to move to a vertex that is not adjacent to the cop, due to the remaining vertices not dominating over each other. By playing on these remaining vertices, the robber would evade capture indefinitely. □

Example 2.2. let there be four vertices left in the graph below during dismantling and let there be edges between vertices shown below. Because all vertices do not dominates each other, none of these vertices are removed during dismantling.

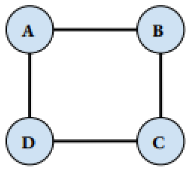


Figure 4: An example of a graph where no vertices are dominant over each other, in which dismantling does not occur.

Lemma 2.3. *After the robber's turn where the cop's position is c and the robber's position is r, either:*

- *$r < c$ with respect to level of domination*
- *or r is adjacent or equal to c*

Lemma 2.4. *Let $v_i < v_j$ in the level of domination and $i < j$. If there is an edge from v_i to some v_k where $j <$, then all vertices $v_i, \ldots, v_x, \ldots, v_j, \ldots, v_{k-1}$ must be adjacent to v_k*

Proof. Since $k > j > i$, v_k would still be in the graph when v_i and v_j have been deleted. Note that v_i is only less than v_j, meaning that they are not adjacent to each other, and that at least one other vertex in between that connects v_i and v_j, denoted v_x, where $i < x < j$. Since v_k is the largest it is the most dominant and thus if we follow the sequence of deletion and because v_k has an edge with v_i, all other edges must have an edge to v_k (v_i, v_j, v_x). □

Proof. (Lemma 2.3) Suppose a game is played on a dismantlable graph, where $r < c$. Consider the movement of the robber during some arbitrary round i. Let r be the robber current position and let r' be the position after the robber's movement. Three cases are possible: $r > r'$, $r = r'$, and $r < r'$. If $r > r'$, then $c > r > r'$, where $c > r'$. So the robber's new position would be less than the cop's position. If $r = r'$, then $c > r = r'$, where $c > r'$. So the robber's new position would still be less than the cop's position. If $r < r'$, then the case $r' > c > r$ is possible. For this case, there is an edge from r' to r. Since $c > r$, and there is and edge from r to r' and $r < r'$, then by lemma 2.4 there has to be an edge from c to r'. Therefore, c would be adjacent to r'. One other case that results from $r < r'$ would be $c = r'$, and that would be the capture of the robber.

Following a sequence, such properties are applied to every move of the robber and the cop. For a finite sequence such as a dismantlable graph, the robber would eventually be caught at most n moves, where n is the number of vertices in the graph. □

Using the above lemmas, one can use such sequence to produce an cop-win algorithm based on the dismantling techniques. First, determine whether a graph is dismantlable by dismantling dominated vertices and removing them. While in the process, push the dismantled vertices into a stack. Keep dismantling until one vertex remains in the graph. That vertex can be used as the cop's starting position for the graph. From that stage, the stack now contains dismantled vertices with levels of domination, with the top of the stack being more dominant and the bottom less.

Using this folded stack of vertices, we begin to unfold the stack by popping out vertices as the cop and Robber makes their moves. As long as the cop is placed at the most dominant vertex of the dismantlable graph, then it will not matter where the Robber is placed. For every vertex that is unfolded, the cop will make their move accordingly to how the Robber moves. Through this moving algorithm, the cop will get closer and closer to the Robber, and will capture the Robber in at most n moves, where n is the number of vertices in the graph.

Below is the pseudo code for the cop algorithm on a dismantlable graph. A stack that will contain vertices and their edges have already been declared. Global variables c represents the cop's position, and r represents the robber's position.

```
while (graph does not contain 1 vertex)
        push vertices and its edges into stack
        delete vertices from graph after pushed into stack
end while
pop vertex from stack
set $c$ to the popped vertex
if ($c$ == $r$)
     return cop-win, where the cop successfully catches the robber on this turn.
else
   continue
```

3 Determining Cop-Win Graphs with Multiple Cops

If we were to follow the default rules of one cop and one robber on a cycle of 4 vertices that would mean the robber wins, however if we changed this rule to be $k > 1$ number of cops, this robber-win graph is now a cop-win graph. With this in mind, the question becomes, "what constitutes a cop-win graph when there are k cops?"

To answer this question, [1] proposed an algorithm that had a runtime of $O(n^{2k+2})$.

Regarding the implementation, we first construct all the states and then dynamically determine if the states are cop-win or not. A state is a tuple of vertices that contain the positions of the cops and robber. To do this, we map the states to an integer that determines how many turns it takes for the cops to win the game, which we denote as t. We then loop through the states and check if each state can win in a certain amount of turns. For instance, working with the current state that is being looked at, we get the neighbors of the state, which are states that contain adjacent vertices of each position in the current state, and check if they are all cop-win and that every single one has a result that leads to at least $t - 1$ turns to win. This process repeats until $t = 0$ signifying that the cop has won and concluding that this graph is cop-win.

This algorithm determines if a graph is cop-win by compiling all possible cop-win states and checking the neighbors of the states to find the neighbor that takes at least $t - 1$ turns. This checks for all possibilities that the robber could move and makes sure that the robber will be captured at every position it moves to. Since no matter where the robber moves, the cop will always capture, leading to the proof that the graph is cop-win. Otherwise, the graph is robber-win.

This algorithm was recently improved upon when in 2021 [2] developed a new algorithm with the current fastest runtime of $O(kn^{k+2})$. This new algorithm works similarly to the above by first checking if there are the same number of cops or more than the number of vertices, then compiling all the possible states and filtering them by if they are cop-win or not. However, this algorithm implements a breadth-first search and keeps track of the player turns to go through every feasible cop-win state or eventual cop-win state and from there ultimately decides if the graph is cop-win, thus making it less brute-force like and faster in time complexity. In this context, a state is a tuple that holds the vertices that each player is currently located in and a record of which player's turn it is. The algorithm and its pseudo code are described below.

First we determine if the number of cops is greater than or equal to the number of vertices. If so, then the graph is cop-win as all the cops are able to occupy the available locations and the robber will have nowhere to go but be on top of a cop.

After this check, the algorithm initializes 3 structures:

- COPSWIN: A hashmap that maps its state to a boolean to determine if the current state is cop-win or not. A state is considered cop-winning, if the cops can definitely catch the robber in their next turn.
- COUNTER: An array of non-negative integers that keeps track of the number of neighbors of a state which the robber can still move from without being captured.
- QUEUE: A queue that holds known cop-win states and starts off empty.

In terms of initializing each structure, COPSWIN takes O(kn^{k+2}) operation as it takes into account all the possible states. COUNTER takes O(kn^{k+1}), since this variable looks at the edges of each vertex, and loops through each state. Lastly, QUEUE when initialized, takes O(1) time since it is just making an empty linear data structure.

```
for $s = (p_0, p_1, ..., p_k, t) \in S$ do
     for $i \in [k]$ do
          if $p_0 = p_i$ then
               enqueue $s$ in QUEUE
               COPSWIN($s$) := 1
               break
          end if
     end for
end for
```

Moving along, the for-loop traverses through all the states in the state-space S. If the state of the robber p_0 is equal to the state of the current index p_i, then we add the state to the queue. Adding the state to the queue means that the state is cop-win, and thus we set the COPSWIN value of the state in the hashmap to equal 1.

Since first for loop looks at all the states, this takes O(kn^{k+1}). Then the for loop inside goes through and check all the cops, which takes O(k). The if statement and the operations below all take constant time O(1). So with the nested elements being multiplied together, this section takes O(k^2n^{k+1}) time.

```
while QUEUE not empty do
     dequeue $s = (p_0, p_1, ..., p_k, t)$ from QUEUE
     if $t \neq 1$ then
          for $q \in I(s)$ do
               if COPSWIN($q$) = 0 then
                    enqueue $q$ in QUEUE
                    COPSWIN($q$) := 1
               end if
     end for
     else
          for $q \in I(s)$ do
               COUNTER($q$) := COUNTER($q$) - 1
               if COUNTER($q$) = 0 then
                    if COPSWIN($q$) = 0 then
                         enqueue $q$ in QUEUE
                    COPSWIN($q$) := 1
                end if
            end if
        end for
    end if
end while
```

While the queue of winning states still has elements inside it, we perform a breadth-first search through the states. We first do this by polling a state S from the queue then checking the turn of the state to see if robber or cop moves next. We then check if turn t is not equal to 1, which means that the previous turn is the cops', we get the neighbors of the current popped state, $I(s)$, and loop through those states. After performing the loop, we examine if the neighbor has a `COPSWIN` value of 0 and if so, offer it into the queue and set its `COPSWIN` value to 1. However, if the robber's turn is next (when t is equal to one), the following code goes to the else-statement. In this statement, we loop through all the neighbors of the current polled state and decrement its `COUNTER` value by 1. If `COUNTER` is equal to zero, then the neighbor is offered into the queue and its `COPSWIN` value is set to 1. This ends the while loop and the breadth-first search.

From this part, the if-statement and else-statement take O(n) time as a result of the for-loops, while the while loop takes O(kn^{k+2}) time as the queue contains all the states.

```
if there exists $(p_1, ..., p_k) \in [n]$ s.t. for all $p_0 \in [n]$ :
COPSWIN(($p_0, p_1, ..., p_k, 0$)) = 1 then
    return true
else
    return false
end if
```

Finally, we check through the if-statements that determine if the graph in its entirety is a cop-win or not. We check if all the states in the state-space contain the `COPSWIN` value equal to 1. If all of them have 1, we return true to signify that the graph is cop-win. Otherwise we return false and the graph is robber-win.

Going through all the states takes O(kn^{k+2}) time as we have to go through every state and check their `COPSWIN` values. Each check of the if statements and the returning of the booleans are constant time operations so the overall operation takes O(kn^{k+2}).

The overall time complexity of this algorithm is O(kn^{k+2})) as the while loop is the most dominant of the operations.

This algorithm is the most effective because it makes sure that all cop-win states are mapped to a `COPSWIN` value of 1. and is mathematically a cop-win state. In other words, if a state has a `COPSWIN` value of 1, it has to be true that this state is a cop-win, and with this information we see that every state in the hashmap is a cop-win state which proves that the graph in its entirety is a cop-win graph. If it doesn't follow this exact criteria then it is a robber graph, which makes this complex case simpler. Below is the lemma describing this paragraph.

Lemma 3.1. *Let s be a state in S. COPSWIN(s) = 1 if and only if s is cop-winning.*

Proof. If `COPSWIN(s)` = 1, then s is must be cop-win state. If s is a cop-win state, then `COPSWIN(s)` = 1. Assume q is not cop-win, but `COPSWIN(q)` = 1. There are three places where q had its `COPSWIN` value set to 1, in the nested for-loop, in the first if statement of the while loop, and the else statement in the while loop when looking back at the pseudocode.

In the nested for-loop, where `COPSWIN` values are set to 1. If `COPSWIN(q)` = 1 then q must also be `COPSWIN` because the other states in the queue are already set as `COPSWIN`, such that the cop must move into a cop-win state. Thus, as a result of a contradiction, `COPSWIN`(q) $\neq 1$ and so the algorithm determines this graph is not cop-win.

This leaves us with the else block where q enters the queue when COUNTER(Q) = 0. This cannot work because, all neighbors of q are robber turns, which are states that are cop-win as they are in the queue of cop-win states so, any robber moves will lead to a cop-win and thus q must be a cop-win state. Thus, this provides another contradiction that q cannot have a COPWIN value of 1 and not be a cop-win graph in the only place left to prove this. Thus, if a COPWIN has a value of 1, it is a cop-win state.

□

4 Determining Cop-Win Graphs with a Faster Robber

Now, what if instead of having multiple cops, we just have one cop and one robber again, but with the twist of a modified robber who is twice as fast as it was before. How could we determine a cop-win graph with a robber that can move twice?

For instance, Figure 5 below shows a graph that is normally a cop-win graph, but with the robber moving twice, it becomes a robber win. But why? To figure that out we must redefine vertex dominance.

Definition 4.1 (k-neighborhood)**.** K is the total number of levels for each neighborhood of a vertex. A one-neighborhood would follow our previous definition of neighborhood that captures the adjacents of the referenced vertex. So by extension a K-neighborhood would capture the vertices that are adjacent to the neighborhood of $K - 1$.

Definition 4.2 (K Dominance)**.** K is the total number of levels of neighborhoods that a vertex dominates. The original definition of dominance that we defined above is a one-dominant vertex where a vertex is dominant if the adjacent vertices is reachable by itself. Now changing the definition, we can say a vertex V is K dominant over vertex W, if every vertex in the K-neighborhood of W is also directly inside the K-neighborhood of V. If needed, refer to Figure 6 for a visual representation and a detailed description.

Theorem 4.3. *If a robber moves k turns faster than the cop, then a graph that can be dismantled under the new definition of dominance is cop-win.*

Proof. Suppose that a graph after dismantling leaves one vertex left, then the robber is placed on the same vertex as the cop. This is possible because all of the vertices dominate each other, hence they are removed during dismantling. So these vertices are adjacent to all of the remaining vertices. □

With this definition, we can now figure out why certain graphs like figure 5 can be robber-win. Referencing to figure 5, the cop can go to the top most vertex, top left corner vertex, or bottom left corner vertex and catch the robber in one turn, but since the robber can now move twice, it can evade capture indefinitely by moving to vertices that will always take more than one turn to capture. This is since the graph is not dismantlable from our new definition of dominance, where instead of being reduced to one dominant vertex, which would be the top most vertex in our previous definition, it is now reduced down to two vertices that the robber can constantly move toward to avoid capture. For this particular graph, these vertices are the top most and the left most vertex (the triangle). These two vertices are the adjacents of their adjacents, but since they do not directly connect to each other, there is no singular true dominant vertex to dismantle into, thus creating a 2 vertex loop, which is why this former cop-win graph is now a robber-win graph. So, to determine if a graph of 1 cop and 1 fast robber is a cop-win or not, we have to use this new definition of dominance to decide if this graph is dismantleable or not.

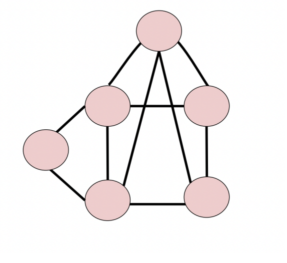


Figure 5: An example of a graph where the cop cannot capture the robber, even if the robber moves twice as fast as the cop. When the robber moves the triangle that is demonstrated on the left, it can evade capture.

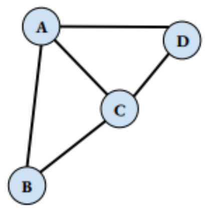


Figure 6: An example of a cop-win graph where if we look at vertex A, and then check the adjacents of the adjacents of A, vertex A still remains the most dominant out of the vertices, therefore, when the graph dismantles, A will be left. With a single vertex, the cop can capture the robber.

Such that we can conclude that if the cop can reach all the adjacent of the robbers adjacent, then the cop can capture the robber. The adjacent of a robbers adjacent refers to the neighborhood of the robber when it can move twice as fast.

If the graph contained k-cops and there exists a fast robber, the idea that as long as the cops can reach the adjacent of the adjacent of the robber, then the graph is considered cop-win.

5 Cop-Number of Graphs with Known Treewidth

In this section, we are going to introduction the concept of treewidth for a graph. Informally, it describe how similiar a graph G is to some tree. Specifically, the treewidth of a graph G maps G to some positive integer. Treewidth as been used in the past to produce results on the upper bound of the cop number of a graph if the graph has bounded treewidth. In the past, treewidth has been used to provide polynomial-time algorithm for NP-complete problems for graph with bounded treewidth. [4] One such example is the graph coloring problem. It is well known that this problem is NP-complete. However, if we want to find a coloring for graphs with bounded treewidth, there exists a polynomial-time algorithm to determine a coloring. This algorithm uses the tree decomposition and its properties to determine a valid coloring of the graph if it is exists. [4]

Treewidth as also been used in theorem to find an upper bound to the cop-number of a graph. One such example of this is Seymour's work on the Helicopter variant of the Cops and Robbers game. In this version of the game, we have a set of k cops to capture the robber. The cops traverse the graph by using a set of helicopters to move them from any one vertex on the graph to another. However, if the cops move from one vertex to another, the cop player must give this information to the robber player. Before the cops land on their destination, the robber is allowed to move on any vertex on any path that includes the current position of the robber and a cop does not occupy vertex in this path. If the robber can evade the cops, then the cops land and next round begins. Seymour showed for that for this variant of the Cops and Robbers game, the graph G has treewidth k if and only if the graph is $(k+1)$-cop-win. [3]

From our previous results, we know that all trees are 1-cop-win graphs. Given this result, we hope show that the treewidth of G provide some upper bound on cop-number of the graph G.

If possible, add a couple examples here about how tree width has been used in the past.See here

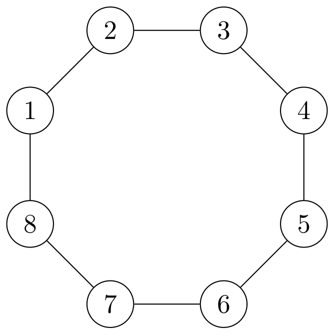
Definition 5.1 (Tree Decomposition)**.** For a given graph G, we say that a tree T is a tree decomposition of G if it satisfies the following properties:

1. There exists a injective function $\beta_T : V(T) \to 2^{V(G)}$ such that it maps every vertex of T to some subset of vertices of G ($2^{V(G)}$ denote the set of all subsets of $V(G)$). A **bag** is some element of Range(β_T). When we say a bag B is part of the tree decomposition T, this means that there exists some vertex $v \in V(T)$ such that $\beta_T(v) = B$. When we use language similar to this phrase, it has the same meaning as the previous phrase.

2. For every edge in the graph G, there must exists some bag B such that it contains the endpoints of that edge.

3. For all bags in the tree decomposition T that contain some vertex $v \in V(G)$, the vertices in T corresponding to these bags must be connected.

Example 5.2 (Examples of Tree Decompositions)**.** For the graph G below:



We show two tree decompositions for the graph above:

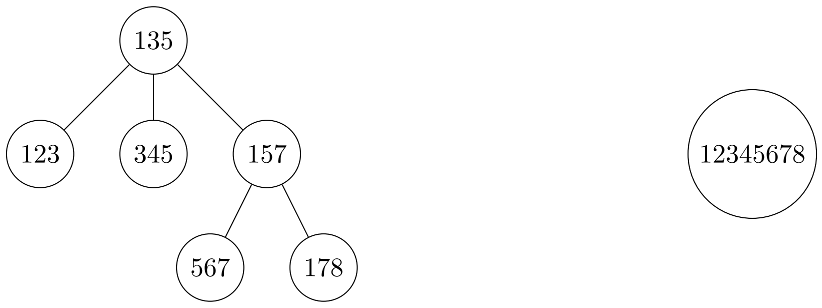


Figure 7: Two tree decompositions for graph G. The left tree is an example of a non-trivial tree decomposition. The right tree is an example of a trivial tree decomposition.

Notice that the tree decomposition of a graph is not unique. The reader can check that trees above satisfy all the properties of a tree decomposition.

Definition 5.3 (Width)**.** For some tree decomposition T, the width of T is defined as the following:

$$\{\max(|B_i|) - 1 : B_i \in \text{Range}(\beta_T)\}.$$

In words, the width of the tree is largest bag size of the tree minus one.

Definition 5.4 (Treewidth)**.** For a given graph G, the treewidth of G is minimum width of all possible tree decompositions of G. Let $\text{tw}(G)$ denote the treewidth of G.

The rest of the section will be devoted to proving the Theorem 5.5. We will state the theorem, show an outline of the proof for Theorem 5.5, prove some necessary lemmas, and show the proof Theorem 5.5. The proof for Theorem 5.5 is an adapation of a proof for the same theorem for helicopter variant of the Cops and Robbers game. [3]

Theorem 5.5. *If a connected graph G has a treewidth of at most k, then G must be $(k+1)$-cop-win graph.*

Our goal is to show that there exists a strategy for the cops to capture the robber by using a tree decomposition T of width $\text{tw}(G)$. In order to show our strategy, we need a way to express the positions of cops and robber on T. Since our strategy uses $k+1$ cops, we can always place one cop on every vertex of any bag in T. Recall that for T, the maximum bag size is $k+1$. Therefore, the position of the cops can be expressed by some vertex of T. Specifically, it is the vertex $v \in V(T)$ such that there is a cop placed on every vertex in $\beta_T(v)$. The robber's position is represented by a subgraph of T that corresponds to all vertices of T that are associated with bags containing the vertex the robber occupies. We will refer this subgraph as the "subgraph corresponding to the robber's position" or T_R.

Our strategy for capturing the robber is as follows. At the beginning of the game, when the cop player places the starting position of the cops on the graph, they choose to place the cops on every vertex in some bag in our tree decomposition. If it is the cop player's turn and the set of vertices the cops occupy is the same set as some bag of T, we want to move the cops onto some vertex in T such that reduces the distance between the position of the cops on T and T_R. An example of this is as follows:

Example 5.6. Let the graph G of left in the figure below be the graph we play the game on. The tree T on the right is the tree decomposition for G. We play the game with 3 cops.

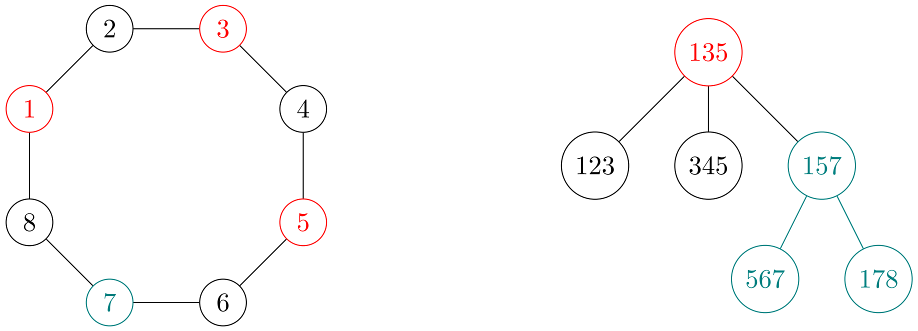


Figure 8: Suppose the cops occupy vertices $\{1, 3, 5\}$ on G. Let the robber occupy vertex 7 of G. If it is the cop player's turn, we want to reduce the distance between vertex 135 (the cop's position in T) and the subgraph of T containing vertices $\{157, 567, 178\}$ (the subgraph of T corresponding to the robber's position). The cops moving from vertex 135 to 157 achieves this goal. To make this move on G, we spend several rounds moving the cops on some path such that they occupy the vertices $\{1, 5, 7\}$. In the example, we find a path to move the cop on vertex 3 to vertex 7.

We need to show that as the cop's position changes from one vertex in T to another, we need to show that the cops can eventually occupy a vertex in T_R. If this happens, then there is one cop that occupies the same vertex as the robber. In order to that, we need to prove the following lemmas.

Lemma 5.7. *For a connected tree T and a connected subgraph H of T, if $v \notin V(H)$ for some vertex $v \in V(T)$, then if we remove vertex v and all edges containing v (let us denote this as $T - \{v\}$), then H is contained in exactly one connected component of $T - \{v\}$.*

Proof. (Lemma 5.7) Since T is a tree and is connected, by removing a vertex from T, the resulting graph is a forest with connected components. Since H is connected, $V(H)$ cannot contain vertices from two different components of $T - \{v\}$. There does not exists an edge in $T - \{v\}$ whose endpoints are two different vertices from two different components of $T - \{v\}$. As a result, H is contained in exactly one connected component of $T - \{v\}$. □

For our strategy, there exists some round such that the cops will be on some vertex in $V(T)$. By the third property of a tree decomposition, T_R must be connected because for all vertices in this subgraph, the corresponding bags of these vertices contains the vertex the robber occupies. Notice that T_R cannot contain the vertex the cops occupy. If it did, then there exists a cop who occupies the same vertex as the robber. As a result, Lemma 5.7 provides the following corollary:

Corollary 5.8. *For a connected graph G and a tree decomposition T of G, if the cops occupy every vertex in some bag $\beta_T(v)$ for $v \in V(T)$, then T_R is contained in exactly one component of $T - \{v\}$.*

Lemma 5.9. *For a connected graph G, let T_R be the subgraph corresponding the robber's position at round i. Let T'_R be the subgraph corresponding the robber's position round $i+1$. Then, $V(T_R) \cap V(T'_R) \neq \emptyset$.*

Proof. (Lemma 5.9) By the second property of a tree decomposition, we know that T_R and T'_R are not empty. The robber occupies some vertex v in G. Since G is connected, the neighborhood of v is not trivial. Then, there exists some edge in G such that there exists some bag who contains v and some element not v in $N[v]$. If the robber occupies vertex R in G at round i and vertex R' in round $i+1$, then there exists an edge whose endpoints are R and R'. Therefore, both vertices are contained in some bag for our tree decomposition. Let that vertex be u. Therefore, $\{R, R'\} \subseteq \beta_T(u)$ and $u \in V(T_R) \cap V(T'_R)$. Hence, $V(T_R) \cap V(T'_R) \neq \emptyset$. □

Lemma 5.10. *For a connected tree T and some vertex $v \in V(T)$, for any component C_i of $T - \{v\}$, there is only one vertex in C_i such that it is in the neighborhood of v.*

Proof. (Lemma 5.10) Suppose there are two vertices u and w in C_i such that they are in the neighborhood of v. Since C_i is connected, then there exist path from u to w. Therefore, in the original tree T, there exists path from v to u to w to v. This path is cycle. Trees do not contain cycles. Therefore, therefore, this is not possible. Hence, only one vertex in C_i is in the neighborhood of v. □

All the lemmas needed to prove Theorem 5.5 have been presented and proven. Now we can present the proof for the main theorem of this section.

Proof. (Theorem 5.5) We will show a strategy the cops use to capture the robber. Let graph G be the connected graph to play the game of Cops and Robbers, and let tree T be the tree decomposition of G whose width equals $\text{tw}(G)$. At the beginning of the game, we place the cops on every vertex some bag B for our tree decomposition. Let u be the vertex such that $\beta_T(u) = B$. Let $(T_R)_i$ be the subgraph corresponding the robber's position at round i. By Corollary 5.8, $(T_R)_0$ will be contained in exactly one connected component of $T - \{u\}$.

When it is the cop player's turn, either one of two cases will occur. For the set of vertices C the cops occupy in G, either there exists some bag B such that $C = B$ or no such bag B exists. If the former case occurs, then we want to move the cops in T such that the distance between the cops position in T and the subgraph $(T_R)_i$ is reduced. Example 5.6 is an example of this move. We claim that as the cops transition from vertex v to w in T, if the T_R is contained C_i in the graph $T - \{v\}$, then for every round it takes for the cops to transition from vertex v to w, T_R is contained in C_i, or at some turn in this transition the robber will occupy the same vertex as a cop.

When we initially transition from vertex v to w, we only move the cops on vertices that are not in the set $\beta_T(v) \cap \beta_T(w)$. Let us consider some arbitrary round j in which the cop player is transition from v to w. Let us assume that $(T_R)_j$ is entirely contained in one component of C_i for the forest $T - \{v\}$. Such a round must exist since this holds when the cops are on v. Suppose it was possible for $(T_R)_{j+1}$ to contain vertices that in component C_k for the graph $T - \{v\}$. Let C_k be a component of $T - \{v\}$ that is not C_i. By

Lemma 5.9, $(T_R)_{j+1}$ must contain a vertex in component C_i. Additionally, $(T_R)_{j+1}$ is connected because all subgraphs corresponding to the robber's position are connected. As a result, $(T_R)_{j+1}$ contains a path from a vertex in C_i to some vertex in C_k. Since T is a connected tree, this path must include v. Moreover, Lemma 5.10 tells us that the subpath from the vertex in C_i to u must contain the vertex w. If both v and w are in $(T_R)_{j+1}$, then $\beta_T(v) \cap \beta_T(w)$ contains the position of the robber. If so, then the robber moved into a vertex occupied by a cop since our strategy ensures that cops are placed on vertex in $\beta_T(v) \cap \beta_T(w)$. As a result this, when the robber moves, $(T_R)_{j+1}$ must be entirely contained in C_i.

We repeat this process until the robber is captured. G is finite graph; therefore, T is finite. Therefore, after finitely many moves, the cops move onto a leaf in T. Our strategy ensures the order of the subgraph is decreasing. If the cops move onto a leaf, then the cops must have captured the robber. Hence, the algorithm terminates and captures the robber. □

Observation 5.11. The converse statement of Theorem 5.5 is not true.

It is a known fact that for any complete graph K_n, $\text{tw}(K_n) = n - 1$. However, it is easy to see that all complete graphs are 1-cop-win. Hence, for any integer m, there always exists a graph G such that G is 1-cop-win. One such choice is K_m.

6 Conclusion

In this paper, we have surveyed some results in the game of Cops and Robbers and analyzed a few of them. Our paper focuses on computing upper bounds for the cop-number of a graph. In the first few sections of the paper, we showed two algorithms to determine if a graph G can capture a robber. The first algorithm determine if one cop can capture the robber. The second algorithm determine if k cops could capture the robber. We also showed results for a variant of the game where there is one cop and one robber on the graph, but the robber is allowed to move from its current vertex R to any vertex on a path whose length is at most s from R. For this variant of the game, we showed if a graph is 1-cop-win. Lastly, we introduced the concept of treewidth. We introduced an algorithm that shows that if a graph has treewidth k, then the cop number of the graph is at most $k + 1$.

7 Future Work

Thus far, this paper has shown results for one variant of the game where cops and a robber take turns moving from one vertex to an adjacent one. The cop and robber player have perfect information about the position of every cop and the robber on the board. This version of this game has been very well studied in the past. Less is known about different variants of this game. We hope to continue our research by producing new theorems for different variants of the game. Either, we will create new variants of the game or expand on the work of others for more known variants of the game. For new variants of the game, we hope to be able to prove theorems related the cop-number of graph and other important concepts fundamental to this game. Some variants of the game of interest is exploring the game on a time-variant graphs. These are graphs whose edge set changes for every round that passes. We are interested in the determining the capture time of a graph or the minimum number of rounds it takes k cops to capture a robber. Little is known about the capture time for k-cop-win graphs.

References

[1] N. E. Clarke and G. MacGillivray. Characterizations of k-copwin graphs. *CoRR*, abs/2112.07449, 2021.

[2] J. Petr, J. Portier, and L. Versteegen. A faster algorithm for cops and robbers. *CoRR*, abs/2112.07449, 2021.

[3] P. Seymour and R. Thomas. *Graph Searching, and a Minimax Theorem for Tree-width.* DIMACS technical report. DIMACS, Center for Discrete Mathematics and Theoretical Computer Science, 1989.

[4] Wikipedia. Treewidth — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Treewidth&oldid=1125634298`, 2023. [Online; accessed 31-January-2023].