The Game of Cops and Robbers and Computability Theory

Natalie DeVos

B.S. Pure Mathematics, Central Michigan University, Mt. Pleasant, Michigan, 2020

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

at the

University of Connecticut

2023

APPROVAL PAGE

Master of Science Thesis

The Game of Cops and Robbers and Computability Theory

Presented by

Natalie DeVos, B.S.

Major Advisor _____
Reed Solomon

Associate Advisor _____
Damir Dzhafarov

Associate Advisor _____
Jeremy Teitelbaum

University of Connecticut

2023

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Reed Solomon, for all of his help and guidance. Reed was always available when I had questions and always made me feel capable and supported. I couldn't have completed this project without his dedication, for which I am incredibly grateful.

I would also like to thank Damir Dzhafavrov and Jeremy Teitelbaum for serving on my committee and for taking the time to attend my defense.

Finally, I am thankful for my family and friends and their support and encouragment throughout my time at UConn. In particular, I would like to thank my mom for all of our conversations filled with love and support and for helping me realize and pursue the path I am on now.

Contents

Chapter 1

Introduction

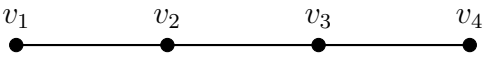
1.1 The Game of Cops and Robbers on Graphs

The idea behind the game of cops and robbers, in general, is that a cop is chasing a robber who is trying to evade the cop's capture. We can play this game on graphs by allowing the cop and robber to occupy vertices and move along edges. The game starts with the cop and robber first selecting their initial positions. The game is then played in rounds where, on each round, the cop makes the first move followed by the robber. The goal of the cop is to catch the robber, while the goal of the robber is to avoid being caught by the cop.

We say a cop *captures* a robber if they both occupy a single vertex at any point in the game. On each turn, the cop or robber must traverse an edge to a neighboring vertex or decide to stay in place. A *winning strategy for a cop* is a rule telling the cop how to move in every situation of the game that, if followed by the cop, guarantees the capture of the robber. A *winning strategy for a robber* is a rule telling the robber

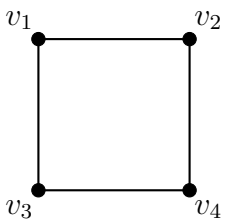
how to move in every situation of the game that, if followed by the robber, guarantees the avoidance of the cop's capture. We say that a graph with a winning strategy for the cop is *cop-win* while a graph with a winning strategy for the robber is *robber-win*. Every finite graph is either cop-win or robber-win; that is, there are no draws in the game.

Example 1. A path on four vertices is an example of a graph that is cop-win.



Without loss of generality, suppose the robber starts on a vertex to the right of the cop. Then the cop can move to the rightmost neighboring vertex on each turn. The robber can do the same in an attempt to avoid capture, but the cop will eventually trap and capture the robber at the end of the path.

Example 2. A cycle on four vertices is an example of a graph that is robber-win.



Suppose the cop starts on vertex v_1. Then the robber must start on vertex v_4, otherwise the cop will be able to capture the robber on its next move. If the cop moves to v_2, the robber can move to v_3. If the cop moves to v_3, the robber can move to v_2. In general, the robber will always be able to move to a vertex that is not adjacent to the vertex the cop is currently on and, as a result, avoid capture.

1.2 The 0-Visibility Variant

There are many variations of the game of cops and robbers on graphs. One such variation is called *0-visibility Cops and Robbers*. The key difference in this game is that the cops and robbers have different knowledge about the other's whereabouts. We assume the cop does not know which vertex the robber is on, unless it is occupying the same vertex as the robber, in which case the game is over and the cop has won. The robber, on the other hand, knows which vertex the cop is on as it plays. This way the robber can try to evade the cop's moves without losing by chance.

We will now discuss some definitions related to the 0-visibility variant on a graph G. A *cop sequence of moves* is a function $c : \mathbb{N} \to G$ such that, for all n, there is an edge from $c(n)$ to $c(n+1)$. Similarly, a *robber sequence of moves* is a function $r : \mathbb{N} \to G$ such that there is an edge from $r(n)$ to $r(n+1)$, for all n. For a cop sequence c and a robber sequence r, we say that c *beats* r if and only if

$$(\exists n)(c(n) = r(n) \vee c(n+1) = r(n)),$$

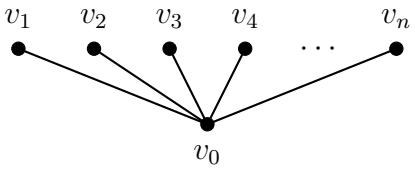
and we say that r *beats* c if and only if

$$(\forall n)(c(n) \neq r(n) \wedge c(n+1) \neq r(n)).$$

Then a graph is *0-cop win* if and only if there exists a cop sequence c such that for every robber sequence r, c beats r. A graph is *0-robber win* if and only if for every cop sequence c, there exists a robber strategy r such that r beats c. Note that when showing a graph is 0-robber win, the robber can play as though it knows all of the cop's moves. That is, when proving G is 0-robber win, we start with an arbitrary cop

sequence c and then build a robber sequence r in such a way that it beats c. Thus, the robber sequence r can use all the information about the arbitrary cop sequence c in order to beat it.

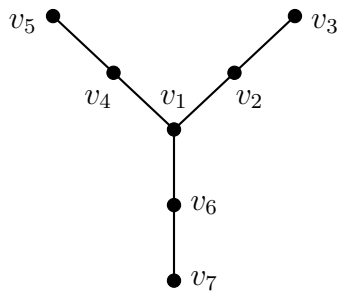
Example 3. The following graph is 0-cop win.



Consider the cop sequence $v_0, v_1, v_0, v_2, v_0, v_3, v_0, v_4, \ldots$. If the robber starts on v_0 or v_1, then the cop will capture the robber by the end of the first round. Suppose, then, that the robber starts on v_i for $2 \leq i \leq n$. The robber has only two options: move to v_0 or stay in place. If the robber moves to v_0, then it is either immediately captured by the cop who is currently on v_0 or it will be captured after the next round when the cop moves to v_0. Thus, the robber has no choice but to stay in place. However, the cop will eventually visit each v_i in its sequence, so the robber will be captured at the vertex it is forced to remain at. Therefore, this cop sequence beats all robber sequences.

Note that this graph would still be 0-cop win if it were infinite. In particular, suppose there were infinitely many vertices with an edge connecting itself and v_0. Then the cop sequence would be an infinite sequence, but the robber still would be stuck on some vertex v_i that is a part of the cop sequence.

Example 4. The following graph is 0-robber win.



A robber can use the center vertex v_1 to switch between the three branches of the graph and escape to a branch the cop is not on or about to move to. For example, consider the cop sequence $v_5, v_4, v_1, v_2, v_3, v_2, v_1, v_6, v_7$. The robber can start on v_6 and remain there for a while, but will need to move eventually to avoid being caught. Once the cop moves to v_2, the robber can move to v_1 and then v_4 to avoid the cop who is about to move to v_6. Thus, a winning robber sequence would be $v_6, v_6, v_6, v_1, v_4, v_4, v_4, v_4, v_4$.

In general, if the cop moves to an outside vertex (i.e. v_5, v_3, or v_7), then the robber has an opportunity to move to v_1 and switch to a new branch of the graph to avoid the cop. Similarly, if the cop stays in place on a middle vertex (i.e. v_2, v_4, or v_6), then this also gives the robber an opportunity to switch branches via v_1.

Consider another cop sequence $v_5, v_4, v_1, v_2, v_1, v_6, v_7$. Unlike the previous example, the robber does not have time to switch to a new branch via v_1 without being caught. However, the cop never reaches the end of one branch, namely the vertex v_3. Therefore, the robber could remain on that vertex the entire sequence and win.

1.3 Computability Theory

We will be exploring the 0-visibility game of cops and robbers using tools from computability theory. Hence, we will briefly discuss some ideas and tools that will be utilized in the later chapters.

Consider a program P which consists of a finite lists of instructions. Suppose we run the program P on an input m. Then our program can either halt and output an answer or it never halts. In particular, if the program P halts on input m, we say that P *converges* on input m and we write $P(m)\downarrow$. Alternatively, if the program P never halts, we say that P *diverges* on input m and write $P(m)\uparrow$.

A *partial function* $p(x) : \mathbb{N} \to \mathbb{N}$ is a function whose domain is contained (possibly strictly) in $\mathbb{N}$. A partial function is called *total* if its domain is all of $\mathbb{N}$. We want to consider partial functions that can be computed by a program. That is, consider a list of partial computable functions $\Phi_0, \Phi_1, \Phi_2, \ldots$. In general, we say that Φ_e is the partial function computed by the program coded by the index e. However, the specific programming language and the coding method doesn't matter here, so we can ignore the choices being made in the background and describe the algorithms informally. (One such way to informally describe an algorithm is using flowcharts, which we will see later in Chapter 3). Then, we can use Φ_e to define a computable function. In particular, we say that a total function f is *computable* if and only if $f = \Phi_e$ for some e. Moreover, to make a total function not be computable, we need that $f \neq \Phi_e$ for each e. This can happen in two ways: for each e, either (1) $\Phi_e(n)\downarrow \neq f(n)$ for some n or (2) $\Phi_e(n)\uparrow$ for some n.

In addition to the programming language, we can add extra instructions using oracles. Given oracle B, we write Φ_e^B for the partial function with fixed oracle B.

Then we can use our program to ask what the value of the oracle is on a given input. For example, if our oracle B is a set and we ask the oracle about the input n, then we get a branch of two possible answers: either $n \in B$ or $n \notin B$. Oracles can also be a three-valued function $c : \mathbb{N} \to \{0, 1, 2\}$ which we will see in Chapter 2. For instance, given a three-valued function c as an oracle, we can ask c for the value of $c(n)$. In this case, we would get a branch of three possible cases: $c(n) = 0$, $c(n) = 1$, or $c(n) = 2$.

A key idea with oracles is that if $\Phi_e^B(n)\downarrow$, then we know that the computation only asks finitely many questions. This means we can find some initial segment $\beta \subseteq B$ such that $\Phi_e^\beta(n)\downarrow$. Moreover, if $\Phi_e^\beta(n)\downarrow$, then for any $X \supseteq \beta$, $\Phi_e^X(n)\downarrow = \Phi_e^\beta(n)\downarrow$. We will see this idea used later with our cop and robber sequences.

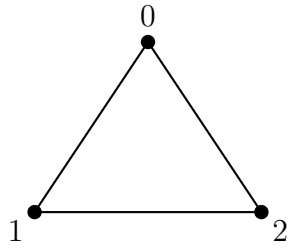
In general, even if a function is not computable, we can still measure its computable power by comparing it to another function (which may or may not itself be computable). First, we say that A is *computable* if and only if the characteristic function on A, denoted χ_A, is equal to Φ_e for some e. Then A *is computable from* B, denoted $A \leq_T B$, if there is some e such that $A = \Phi_e^B$. If both $A \leq_T B$ and $B \leq_T A$, then we denote this as $A \equiv_T B$. We will use this notion of relative computability to explore the relationship between computation and the game of 0-visibility cops and robbers on a particular graph.

Chapter 2

The 0-Visibility Cops and Robbers Game on K_3

2.1 Cop and Robber Sequences on K_3

We now consider the 0-visibility cops and robbers game on the complete graph of three vertices, K_3. We will label and refer to the vertices of K_3 as 0, 1, and 2.



The graph K_3

A *cop sequence of moves* on K_3 is a function $c : \mathbb{N} \to \{0, 1, 2\}$ and a *robber sequence of moves* on K_3 is a function $r : \mathbb{N} \to \{0, 1, 2\}$. Note that this satisfies the original definition of a cop and robber sequence on a graph G. In particular, since

K_3 is a complete graph, there is an edge between any two vertices. Hence, $c(n)$ and $c(n+1)$ share an edge and $r(n)$ and $r(n+1)$ share an edge, for all n.

Example 5. A cop or robber sequence on K_3 can be represented by a sequence of 0s, 1s, and 2s, denoting the vertices the cop or robber will move to on each turn. An example of a cop sequence on K_3 is

$$c : 0\ 1\ 2\ 2\ 2\ 1\ 1.$$

Here the cop starts on vertex 0, moves to vertex 1 on its next turn, then moves to vertex 2 and stays there for three rounds before moving back to vertex 1. An example of a robber sequence on K_3 is

$$r : 2\ 2\ 0\ 0\ 0\ 0\ 0$$

where robber stays on vertex 2 for the first two rounds, then moves to and remains on vertex 0 for the remainder of the game.

2.2 K_3 as a 0-Robber Win Graph

In the normal game of cops and robbers, the graph K_3 is cop-win since every vertex has an edge to the two remaining vertices. The cop can capture the robber after one round by moving to the vertex the robber chose to start on. However, in the 0-visibility variation, the graph is 0-robber win. Recall that when building a robber sequence r that beats a cop sequence c, we can use the information of c to build an r that beats c. That is, the robber can now anticipate and avoid the cop's current and next moves in order to escape the cop's capture.

Theorem 2.2.1. *For every cop sequence c, there exists a robber sequence r such that r beats c.*

Proof. Suppose we're given an arbitrary cop sequence, c. Then we want to build a robber sequence, r, so that the robber is never captured by the cop. Recall that in each round, the cop moves first. Thus, in order for the robber to avoid the cop's capture, the robber must avoid the vertices the cop is on and will move to next. In particular, for each stage n of the game, the robber needs to avoid the vertex the cop is on at the end of stage n as well as the vertex the cop will move to in stage $n+1$. This will either be one or two vertices depending on whether the cop stays in place or not. In either case, there will always be at least one vertex that the robber can move to that will guarantee its non-capture. That is, for every cop sequence c, there is a robber sequence r such that $c(n) \neq r(n)$ and $c(n+1) \neq r(n)$ for all n. □

Later we will need the corresponding fact that every robber sequence beats some cop sequence.

Theorem 2.2.2. *For every robber sequence r, there exists a cop sequence c such that c loses to r.*

Proof. Suppose we're given an arbitrary robber sequence, r. Then we want to build a cop sequence, c, so that the cop never captures the robber. Since the cop moves first each round, it will need to avoid the vertices the robber had previously moved to and is currently on. In particular, for each stage n of the game, the cop must avoid the vertex the robber was on at the end of stage $n-1$ as well as the vertex the robber moved to in stage n. This will either be one or two vertices depending on whether the robber has stayed in place or not. In either case, there is at least one

vertex that the cop can move to that will guarantee it doesn't capture the robber. Thus, for every robber sequence r, there is a cop sequence c such that $r(n-1) \neq c(n)$ and $r(n) \neq c(n)$ for all n. That is, for all r, there exists a c such that c loses to r. $\square$

2.3 Extensions of Cop and Robber Sequences

So far we have seen that given a cop sequence c, we can build the entire robber sequence r (or given a robber sequence r, we can build the entire cop sequence c) so that r beats c. We can also consider the case where we are given both a cop and robber sequence with a number of turns already established so that the robber is beating the cop so far. In this case, we can determine whether the cop and robber sequences can be extended so that the robber continues to beat the cop (or the cop continues to lose to the robber).

Definition 1. Let σ and τ be finite strings from $\{0, 1, 2\}$. We say σ is an *initial segment* of τ (or τ is an *extension* of σ) if $|\sigma| \leq |\tau|$ and $\sigma(i) = \tau(i)$ for all $i < |\sigma|$. We denote this as $\sigma \subseteq \tau$.

Lemma 1. *Let r be a finite robber sequence and let c be a finite cop sequence. Suppose $|c| = |r|+1$ and r beats c. Then for any extension of c, say c', there exists an extension of r, say r', such that $|c'| = |r'| + 1$ and r' beats c'.*

Lemma 2. *Let r be a finite robber sequence and let c be a finite cop sequence. Suppose $|c| = |r|+1$ and r beats c. Then for any extension of r, say r', there exists an extension of c, say c', such that $|c'| = |r'| + 1$ and c' loses to r'.*

The idea of these proofs is very similar to those of Theorem 2.2.1 and Theorem 2.2.2. Since we are setting $|c| = |r| + 1$, we can still make sure that $c(n) \neq r(n)$

and $c(n+1) \neq r(n)$ for all n (in the case of Lemma 1) or that $r(n-1) \neq c(n)$ and $r(n) \neq c(n)$ for all n (in the case of Lemma 2). The only difference here is that we have some starting values where r is beating c. We will see these Lemmas again in Chapter 4 when we discuss cop sequences with infinitely many repeated values.

Chapter 3

Finite Repeats in a Cop Sequence

We now consider a case of the 0-visibility game of cops and robbers on K_3 where the cop stays in place finitely often. That is, assume a cop sequence $c : \mathbb{N} \to \{0, 1, 2\}$ has a finite amount of places where a value is repeated so that the set $\{i : c(i) = c(i+1)\}$ is finite. From here we wish to explore two main ideas:

Q1. Suppose we fix a cop sequence c. Then how many robber sequences r exist such that r beats c? And can the cop sequence compute such a robber sequence?

Q2. Suppose we fix a cop sequence c and a robber sequence r such that r beats c. Then is it possible for the robber sequence to compute the cop sequence? In other words, can we reconstruct c from r?

We will investigate these questions in two cases: first, if the cop sequence has zero repeats and second, if the cop sequence has n many repeats.

3.1 Zero Repeats

In this section, we will assume that all cop sequences c have zero repeated values. That is, every cop sequence considered will move to a new vertex at every stage. Thus, we are looking at the case where the set $\{i : c(i) = c(i+1)\}$ is empty.

Example 6. Consider the following cop sequence c with zero repeats.

$$c : 0\ 1\ 2\ 1\ 2\ 0\ 1$$

In order to find a robber sequence r that beats c, we need the robber to avoid the cop at every move. We know that the cop starts on 0, so the robber cannot also start on 0 otherwise the game would be over. Let's say the robber starts on 1 instead. But on the next turn the cop also moves to vertex 1, thus ending the game. Therefore, the robber must start on vertex 2. In general, because the cop moves first each turn, the robber needs to avoid the vertex the cop is currently on and the vertex the cop is about to move to. Since the cop always moves to a new vertex, these two values are distinct, leaving only one option for the robber. In this case, the robber sequence that beats the cop sequence is:

$$r : 2\ 0\ 0\ 0\ 1\ 2.$$

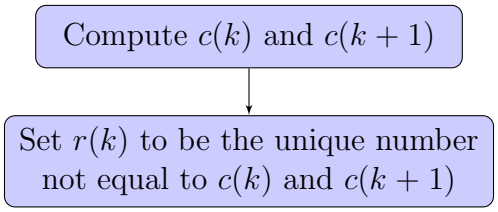
Lemma 3. *If $|\{i : c(i) = c(i+1)\}| = 0$ for a cop sequence c, then there exists a unique robber sequence r such that r beats c.*

Proof. We know that $r(k)$ cannot be equal to $c(k)$ or $c(k+1)$ for all k in order to avoid capture. But $c(k)$ and $c(k+1)$ must be distinct values since the cop sequence has zero repeats. Therefore, $r(k)$ is uniquely determined for each k. □

Lemma 4. *Fix a cop sequence c with zero repeats. Then c computes a robber sequence r that beats c.*

Proof. Suppose we're given a cop sequence c with zero repeats. Then we can determine a robber sequence r that beats c using the algorithm found in Figure 3.1.1. □

FIGURE 3.1.1: Algorithm for $r(k)$ with zero repeats



Notice that we have now answered the questions in Q1 in the case where the cop sequence has zero repeats. In particular, suppose we fix a cop sequence c. Then there is only one robber sequence r such that r beats c. And it is true that c can compute such an r. We will now explore the ideas in Q2. In particular, suppose we now start with a robber sequence r and want to find a cop sequence c such that c loses to r.

Example 7. Let's first look at a cop sequence c that consists of only two values with zero repeats. For example, consider

$$c : 1\ 2\ 1\ 2\ 1\ 2\ 1\ 2.$$

Then using the algorithm in Figure 3.1.1, we can determine that the unique robber sequence that beats the cop sequence is

$$r : 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0.$$

Suppose now we start with this r and want to determine c. We know that the cop can never move to 0 (otherwise it captures the robber) and that the cop moves to a new vertex each turn (since there are zero repeats). Thus, we can deduce that the cop must be moving back and forth between values 1 and 2. But which value did the

cop start on? Notice that this r also beats the cop sequence $c : 2\ 1\ 2\ 1\ 2\ 1\ 2\ 1$. Thus, in order for r to compute c, it needs to also know the value of $c(0)$.

Lemma 5. *Fix a cop sequence c that consists of only two values and has zero repeats. Fix a robber sequence r that beats c. Suppose we're given the value of $c(0)$. Then r computes c.*

Proof. We know that the fixed robber sequence r consists entirely of one value. We also know, for the fixed cop sequence c, that $c(k) \neq c(k+1)$ for all k (since c has zero repeats) and $c(k) \neq r(k)$ for all k (since r beats c). Thus, c must alternate between the two values not equal to the value in r. Since we're given $c(0)$, we can determine the exact cop sequence c that loses to r. □

There is a similar trivial case when the cop sequence has at least one repeated value. The cases where the cop only takes one or two values will be left to the reader in the future. We will assume from now on that all cop sequences will consist of three values.

Example 8. We now consider an example where the cop sequence consists of three values. Consider the following robber sequence.

$$r : 0\ 0\ 0\ 1$$

The only information we have about $c(0)$ is that it can't be equal to $r(0) = 0$. However, the fact that the robber eventually moves to 1 means that we will actually be able to determine $c(0)$. We know that the cop always moves to a new vertex on each turn, hence the cop must be traveling back and forth between vertices 1 and 2 while the robber stays put on vertex 0. However, we need to make sure that the cop isn't on vertex 1 when the robber moves there, i.e. we can set $c(3) = 2$. From here we

can work backwards to determine that $c(2) = 1$, $c(1) = 2$, and $c(0) = 1$. In general, we can determine $c(k)$ where k is the first position where the robber moves to a new vertex, and then work backwards to determine the first k many positions of the cop sequence.

Definition 2. Define $k_{\min} := \min\{k : r(k) \neq r(0)\}$. In other words, let $k_{\min}$ be the first place in a robber sequence in which the robber moves to a new vertex.

In the previous example, we saw why this least k value is important. In particular, for the robber sequence $r : 0\ 0\ 0\ 1$, we have that $k_{\min} = 3$.

Example 9. Suppose we start with a robber sequence $r : 0\ 0\ 0\ 1\ 2\ 2$ and assume that we've found the first four values of the cop sequence c as explained in Example 8. That is, we so far have

$$c : 1\ 2\ 1\ 2\ ?\ ?.$$

In order to determine $c(4)$ notice that, because the cop moves first each turn, the value of $c(4)$ cannot be equal to $r(4)$, nor can it be equal to $r(3)$. The robber will still be on $r(3)$ by the time the cop moves to $c(4)$, thus there are two robber values the cop must avoid. Thus, $c(4) \neq r(4) = 2$ and $c(4) \neq r(3) = 1$. And so $c(4) = 0$. If we next want to find $c(5)$, notice that $r(5) = r(4) = 2$. Thus we know that $c(5)$ can either be 0 or 1. However, we also know that the cop moves to a new vertex on each move, so we must have that $c(5) \neq c(4)$ which we determined to be 0. Thus, $c(5)$ must be 1. The cop sequence c that beats the robber sequence r, then, is:

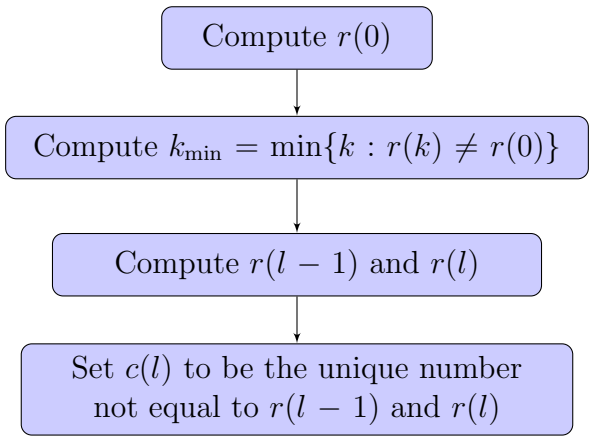
$$c : 1\ 2\ 1\ 2\ 0\ 1.$$

Lemma 6. *Fix a cop sequence c that consists of three values and has zero repeats. Fix a robber sequence r that beats c. Then r computes c.*

Proof. Suppose we're given a cop sequence c with zero repeats and a robber sequence r that beats c. Then we can determine c in three cases. First, compute $c(l)$ when $l = k_{\min}$ using the algorithm in Figure 3.1.2. Second, use $c(l)$ to compute $c(l-1)$ for $l \leq k_{\min}$ using the algorithm in Figure 3.1.3. And third, use $c(l)$ to compute $c(l+1)$ for $l \geq k_{\min}$ using the algorithm in Figure 3.1.4. □

For the algorithm in Figure 3.1.2, notice that $r(k_{\min} - 1)$ is equal to $r(0)$ since $k_{\min}$ is the first position where r changes value. Thus, $r(k_{\min}) \neq r(k_{\min} - 1)$, leaving one unique value for the cop at position $l = k_{\min}$ to avoid capturing the robber.

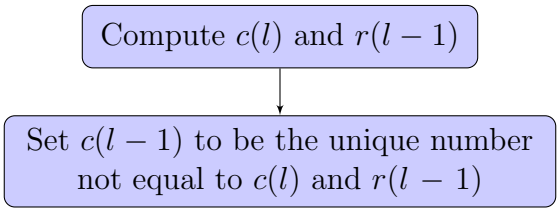
FIGURE 3.1.2: Algorithm for $c(l)$, $l = k_{\min}$ with zero repeats



After determining $c(k_{\min})$, we can use the algorithm in Figure 3.1.3 to work backwards to determine the values from $c(k_{\min} - 1)$ to $c(0)$. Up to this point, then, we will have determined the first $k_{\min}$ values of the cop sequence.

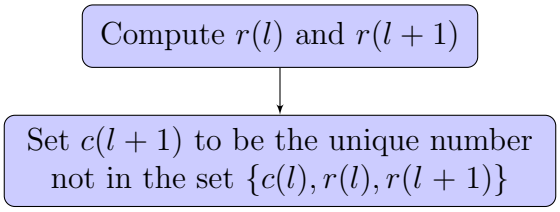
For the algorithm in Figure 3.1.4, note that that set $\{c(l), r(l), r(l+1)\}$ will always consist of two values. To see why first suppose that $r(l) = r(l+1)$. Then $c(l)$ must be a value different from $r(l)$ in order to avoid the robber. Hence $r(l)$, $r(l+1)$ and $c(l)$ have two distinct values between them. Second, suppose $r(l) \neq r(l+1)$. In particular,

FIGURE 3.1.3: Algorithm for $c(l-1)$, $l \leq k_{\min}$ with zero repeats



let $r(l) = x_1$ and $r(l+1) = x_2$. Then $c(l+1)$ must be the unique value not equal to either $r(l)$ or $r(l+1)$, say x_3. Then $c(l)$ can either be x_2 or x_3. However, since the cop always moves to a new vertex, $c(l) \neq c(l+1)$. Thus, $c(l) = x_2$. Overall, we have that $r(l+1) = c(l) = x_2$ and $r(l) = x_1$, two distinct values.

FIGURE 3.1.4: Algorithm for $c(l+1)$, $l \geq k_{\min}$ with zero repeats



Now we have answered the questions in Q2 in the case where the cop sequence has zero repeats. In particular, suppose we fix a cop sequence c with zero repeats and a robber sequence r such that r beats c. Then we can reconstruct c from r, meaning the robber sequence can compute the cop sequence.

Theorem 3.1.1. *If* $|\{i : c(i) = c(i+1)\}| = 0$ *for a cop sequence* c*, then for all* r *that beat* c*,* $r \equiv_T c$*.*

Proof. From Lemma 4, we can conclude that $r \leq_T c$ and from Lemmas 5 and 6 we can conclude that $c \leq_T r$. □

3.2 n Many Repeats

In this section, we will assume that all cop sequences c have n many repeated values. That is, every cop sequence considered will remain on the same vertex $n > 0$ many times. Thus, we are looking at the case where $\{i : c(i) = c(i+1)\} = \{i_1, i_2, i_3, \ldots, i_n\}$.

Definition 3. A *repeat block* is a maximal sequence of two or more consecutive values $i, i+1, \ldots, i+k$ such that $c(i) = c(i+1) = \cdots = c(i+k)$.

We'll identify repeat blocks within a cop or robber sequence by underlining them. For example, the following cop sequence has three repeat blocks.

$$c : 0\ 1\ \underline{2\ 2\ 2}\ \underline{1\ 1}\ 0\ 1\ 2\ \underline{0\ 0\ 0\ 0}\ 1\ 2$$

Note that a repeat block is dependent on the repeated values in a cop sequence, hence values in c will be underlined if and only if they are repeated values. However, in a robber sequence, the corresponding repeat values may not themselves repeat.

Example 10. Consider the cop sequence

$$c : 1\ \underline{2\ 2}\ \underline{1\ 1}$$

To determine a robber sequence r that beats c, we must have that $r(k) \neq c(k)$ and $r(k) \neq c(k+1)$ for each k. That is, in order for the robber to not be captured by the cop, it must avoid the vertex the cop is currently on and the vertex the cop will move to next (since the cop moves first each turn). First, we have that $r(0) \neq c(0) = 1$ and $r(0) \neq c(1) = 2$. Thus, $r(0) = 0$. We next have that $r(1) \neq c(1) = 2$ and $r(1) \neq c(2) = 2$. Because the cop stayed on the same vertex, the robber now has some flexibility in its next move. In particular, we could have that $r(1) = 0$ or $r(1) = 1$. Either choice would result in the robber avoiding the cop. Continuing on we have that $r(2) \neq c(2) = 2$ and $r(2) \neq c(3) = 1$, hence $r(2) = 0$. For $r(3)$, we

again face some flexibility since $r(3) \neq c(3) = 1$ and $r(3) \neq c(4) = 1$. Hence, $r(3)$ can either be 0 or 2, with either choice resulting in the robber avoiding capture. Thus, with these two repeat blocks, we have a total of four robber sequences that beat the cop sequence c. In particular, they are:

$$r : 0\ 0\ 0\ 0 \qquad r : 0\ 1\ 0\ 0 \qquad r : 0\ 0\ 0\ 2 \qquad r : 0\ 1\ 0\ 2$$

Lemma 7. *If $|\{i : c(i) = c(i+1)\}| = n$ for a cop sequence c, then there are 2^n many robber sequences r such that r beats c*

Proof. When the cop moves to a new vertex, the robber position is determined. That is, for each k if $c(k) \neq c(k+1)$, then $r(k)$ must be the unique value not equal to $c(k)$ or $c(k+1)$. When the cop stays in place, however, the robber now has two options of where to move. That is, the robber can stay in place or move to the unoccupied vertex. Thus, for each k if $c(k) = c(k+1)$, then $r(k)$ can be one of two values not equal to $c(k) = c(k+1)$. Furthermore, for every position where the cop stays in place, there are two robber positions, and therefore sequences, that can beat the cop. □

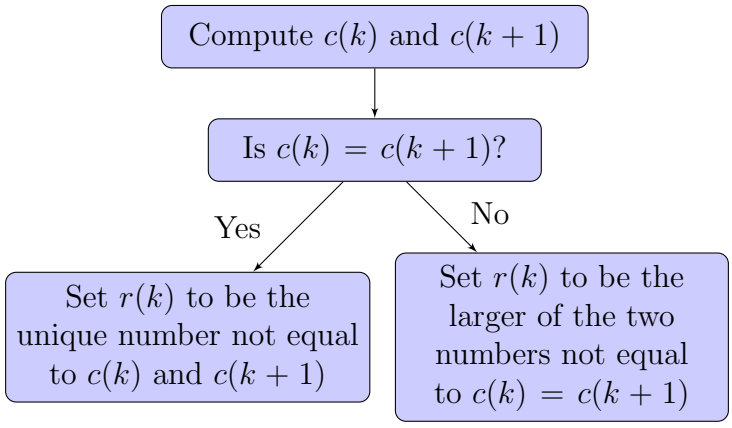
Lemma 8. *Given a cop sequence c with n many repeats, we can compute a robber sequence r that beats c.*

Proof. Suppose we're given a robber sequence c with n many repeats. Then we can determine a robber sequence r that beats c using the algorithm found in Figure 3.2.1. □

Notice that the algorithm in Figure 3.2.1 takes care of the flexibility that arises when the cop sequence repeats a value. That is, when $c(k) \neq c(k+1)$, there are two options for $r(k)$ and the algorithm simply choose the larger of the two. We could've also had the algorithm choose the smaller of the two, or set $r(k) = r(k-1)$ for $k \geq 1$.

But for consistency, we will have the algorithm always select the larger of the two possible robber sequence values.

FIGURE 3.2.1: Algorithm for $r(k)$ with n many repeats



We have thus far answered the questions in Q1 in the case where the cop sequence has n many repeats. That is, suppose we fix a cop sequence c with n many values that are repeated. Then there are 2^n many robber sequences r such that r beats c. And the cop sequence can compute such a robber sequence using the algorithm in Figure 3.2.1.

Suppose now we're given a robber sequence r that beats c and we want to reconstruct the cop sequence c that lost to r. In order to do so, we need to know where the cop sequence has repeated values. That is, our robber sequence must know the position of the repeat blocks in the cop sequence we are trying to reconstruct.

We mark the position of the repeat blocks in the cop sequence by underlining the corresponding positions in the robber sequence. For example, $r : 0\ 0\ \underline{1\ 2}\ 0$ indicates that the original cop sequence c beaten by r had a repeated value in positions 2 and 3. That is, $c(2) = c(3)$.

Example 11. Consider the following robber sequence with a repeat block.

$$r : 1 \ \underline{1 \ 1} \ 0$$

The only information we can gather about $c(0)$ is that it cannot be equal to $r(0) = 1$. For $c(1)$, we know that it cannot be equal to $r(0)$ or $r(1)$, but these are both equal to 1. Thus, is it beneficial to first look at the position where the robber moves to a new vertex, in this case $k = 3$. Since $r(2) = 1$ and $r(3) = 0$, we know that $c(3)$ must be 2. We can now work backwards to determine the first 3 values of the cop sequence. Notice that the repeat block will be very important here. In particular, we know two values in a cop sequence are equal to each other if and only if they are in the same repeat block. Thus, because $k = 2$ and $k = 3$ are not in the same repeat block, we know that $c(2) \neq c(3) = 2$. Additionally, we know that $c(2) \neq r(2) = 1$ and so $c(2)$ must be 0. Then, since $k = 1$ and $k = 2$ are in the same repeat block, we can immediately conclude that $c(1)$ must be 0 as well. Finally, since $k = 0$ and $k = 1$ are not in the same repeat block, $c(0) \neq c(1) = 0$. And, since $c(0) \neq r(0) = 1$, we have that $c(0) = 2$. Therefore, the cop sequence that loses to the given robber sequence is

$$c : 2 \ \underline{0 \ 0} \ 2.$$

Recall that we defined $k_{\min} = \min\{k : r(k) \neq r(0)\}$ to be the first place in a robber sequence where the robber moves to a new vertex. In the example above, we had that $k_{\min} = 3$ and were able to first determine $c(k_{\min})$ then work backwards to find the values for $c(k_{\min} - 1)$ down to $c(0)$. We can next look at an example where we have to determine $c(k_{\min} + 1)$ and beyond.

Example 12. Consider the following robber sequence.

$$r : 1 \ \underline{1 \ 1} \ 0 \ 2 \ 2$$

Assume we have determined the first four values of the cop sequence, as explained in the previous example. That is, suppose we so far have the cop sequence

$$c : 2\ \underline{0\ 0}\ 2\ ?\ ?$$

and want to determine $c(4)$ and $c(5)$. First, to find $c(4)$, we know that $c(4) \neq r(4) = 2$ and $c(4) \neq r(3) = 0$. Thus, $c(4)$ must be equal to 1. To find $c(5)$, we must have that $c(5)$ is not equal to $r(5)$ or $r(4)$, both of which are equal to 2. Thus, $c(5)$ could either be 0 or 1. However, because we know the exact positions where the cop sequence has repeated values, we can also look at the repeat blocks and the value of $c(4)$. In particular, since $k = 4$ and $k = 5$ are not in a repeat block, we know that $c(4) \neq c(5)$ where $c(4)$ was just determined to be 1. Therefore, $c(5) = 0$ and the cop sequence that loses to the given robber sequence is

$$c : 2\ \underline{0\ 0}\ 2\ 1\ 0.$$

Lemma 9. *Fix a cop sequence c that has n many repeats. Fix a robber sequence r that beats c. Then r computes c.*

Proof. Fix a cop sequence c with n many repeats. Suppose we start with a robber sequence r knowing where the corresponding repeat values are in the cop sequence. Then we can compute the cop sequence c from r such that c loses to r in three cases. First, compute $c(l)$ when $l = k_{\min}$ using the algorithm in Figure 3.2.2 (page 25). Second, use $c(l)$ to compute $c(l-1)$ for $l \leq k_{\min}$ using the algorithm in Figure 3.2.3 (page 25). And third, use $c(l)$ to compute $c(l+1)$ for $l \geq k_{\min}$ using the algorithm in Figure 3.2.4 (page 26). □

For the algorithm in Figure 3.2.2, notice that $r(k_{\min}-1)$ is equal to $r(0)$ since $k_{\min}$ is the first position where r changes value. Moreover, $r(k_{\min}) \neq r(k_{\min}-1)$, leaving

one unique value for the cop at position $l = k_{\min}$ to avoid capturing the robber.

FIGURE 3.2.2: Algorithm for $c(l)$, $l = k_{\min}$ with n many repeats

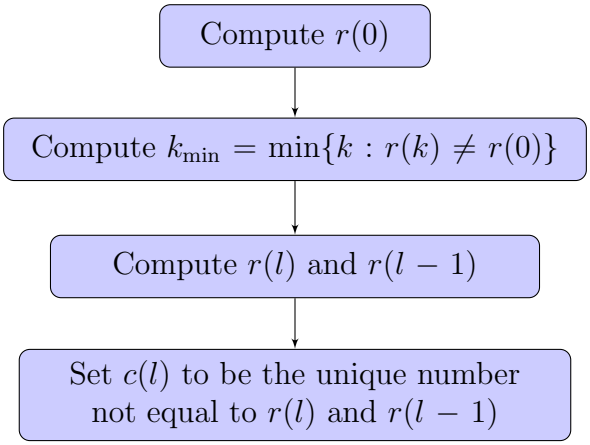
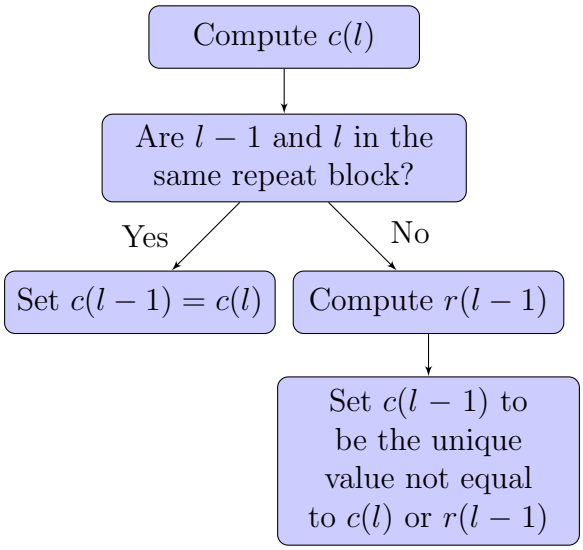


FIGURE 3.2.3: Algorithm for $c(l-1)$, $l \leq k_{\min}$ with n many repeats

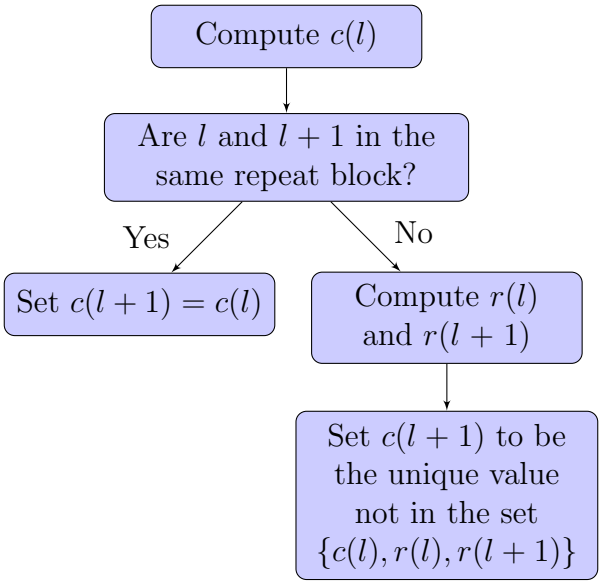


After determining $c(l) = c(k_{\min})$, we can use the algorithm in Figure 3.2.3 to work backwards to determine the values from $c(l-1)$ to $c(0)$. Note that we are using the

information about where repeat blocks are in the cop sequence in this algorithm. Up to this point, we will have determined the first $k_{\min}$ values of the cop sequence.

Using the algorithm in Figure 3.2.4, we can determine the remaining values for the cop sequence, beyond the first $k_{\min}$ many. Note that we again need to use the information about where the repeat blocks are in the cop sequence. Also, note that the set $\{c(l), r(l), r(l+1)\}$ will always consist of two values. An argument for why this is the case was outlined on page 19 when discussing the algorithm in Figure 3.1.4.

FIGURE 3.2.4: Algorithm for $c(l+1)$, $l \geq k_{\min}$ with n many repeats



Theorem 3.2.1. *If* $|\{i : c(i) = c(i+1)\}| = n$ *for a cop sequence* c*, then for all* r *that beat* c*,* $r \equiv_T c$*.*

Proof. From Lemma 8, we can conclude that $r \leq_T c$ and from Lemma 9 we can conclude that $c \leq_T r$. □

Chapter 4

Infinite Repeats in a Cop Sequence

We now consider a case of the 0-visibility game of cops and robbers on K_3 where the cop stays in place infinitely often. That is, for a cop sequence c, we consider when the set $\{i : c(i) = c(i+1)\}$ is infinite.

Theorem 4.0.1. *There exists a cop sequence c and a robber sequence r such that r beats c and $r \not\leq_T c$ and $c \not\leq_T r$.*

In Section 3, we proved that when the set $\{i : c(i) = c(i+1)\}$ is finite, we can always compute c from r and r from c for every cop sequence c and robber sequence r that beats c. However, this theorem tells us that this is not always the case when $\{i : c(i) = c(i+1)\}$ is infinite. Moreover, because we determined $r \equiv_T c$ when $\{i : c(i) = c(i+1)\}$ finite, the fact that both $r \not\leq_T c$ and $c \not\leq_T r$ in our theorem guarantees that we are in the infinite case.

Proof. We want to build a robber sequence $r : \mathbb{N} \to \{0, 1, 2\}$ and a cop sequence $c : \mathbb{N} \to \{0, 1, 2\}$ such that r beats c and $r \not\leq_T c$ and $c \not\leq_T r$.

Note that $c \not\leq_T r$ if $\Phi_e^r \neq c$ for every index e and $r \not\leq_T c$ if $\Phi_e^c \neq r$ for every index e. Thus, we want to construct r and c so that two requirements are satisfied, namely:

$$R_e : \Phi_e^r \neq c \quad \text{and} \quad S_e : \Phi_e^c \neq r$$

Note that $\Phi_e^r \neq c$ in two ways: either $\Phi_e^r(n)\downarrow \neq c(n)$ for some n or $\Phi_e^r(n)\uparrow$ for some n. Similarly, $\Phi_e^c \neq r$ can mean either $\Phi_e^c(n)\downarrow \neq r(n)$ for some n or $\Phi_e^c(n)\uparrow$ for some n.

We can now begin our construction of r and c which will take place in stages. Let r_s be the part of a robber sequence r and let c_s be the part of a robber sequence c determined at stage s. Then r_s and c_s are finite sequences consisting of 0s, 1s, and 2s that are built at the end of stage s. In particular we have that

$$r_0 \subseteq r_1 \subseteq r_2 \subseteq \cdots \quad \text{and} \quad c_0 \subseteq c_1 \subseteq c_2 \subseteq \cdots$$

That is, r_s and c_s are getting longer at every stage in that r_i and c_i are initial segments of r_{i+1} and c_{i+1}, respectively. Then we can set

$$r = \bigcup_{s\in\omega} r_s \quad \text{and} \quad c = \bigcup_{s\in\omega} c_s.$$

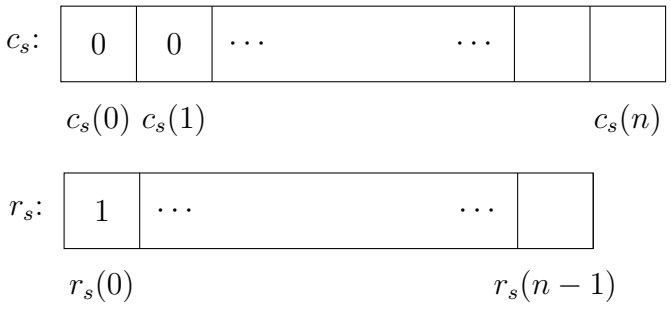
We want to require that r_s beats c_s and also that $|c_s| = |r_s|+1$. The latter is necessary to avoid r_s being determined by c_s since $r_s(i)$ cannot be equal to $c_s(i)$ or $c_s(i+1)$ for all i if r_s is to beat c_s.

Let's first look at stage 0. Here we can specifically set $c_0(0) = c_0(1) = 0$ and $r_0(0) = 1$. This way we will guarantee that r_0 beats c_0 and $|c_0| = |r_0| + 1 = 2$. Then at stage s, we will have the first two values of c_s determined and the first value of r_s determined. Figure 4.0.1 on page 29 gives a visualization of stage s where $|r_s| = n$ with these starting values.

For the remaining stages, we want to make sure that R_e and S_e are satisfied. We can do so by making sure S_e is satisfied when $s = 2e + 1$ and making sure R_e is satisfied when $s = 2e + 2$. We will now look at the construction at stage $s + 1$ in two

cases: (1) when $s = 2e + 1$ and (2) when $s = 2e + 2$. In particular, since we have already constructed r_s and c_s so that r_s beats c_s and $|c_s| = |r_s| + 1$, we now want to define r_{s+1} and c_{s+1} so that r_{s+1} beats c_{s+1}, $|c_{s+1}| = |r_{s+1}| + 1$, and either R_e or S_e is satisfied.

FIGURE 4.0.1



Construction at stage $s + 1$ when $s = 2e + 1$:

Let $n = |r_s|$. Our goal is to extend c_s to c_{s+1} while still satisfying S_e and having r_{s+1} beat c_{s+1} with $|c_{s+1}| = |r_{s+1}| + 1$. By making sure these are satisfied, certain restrictions will be placed on the extension of c_s.

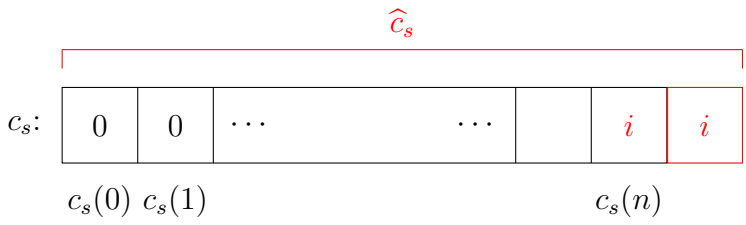
Let's first focus on extending c_s so that it continues to lose to r_s. In particular, let's look at how we can extend c_s so that the value $r_s(n)$ has the most choices. We want that $r_s(n) \neq c_s(n)$ and $r_s(n) \neq c_s(n + 1)$ for r_s to beat c_s. Thus, if we set $c_s(n) = c_s(n + 1)$, we create only one restriction on the value for $r_s(n)$.

Let $\widehat{c_s} = c_s \frown c_s(n)$. (See Figure 4.0.2 on page 30). Now we can look at extending $\widehat{c_s}$ to c_{s+1}. We will need to determine the value for $\Phi_e^c(n)$ since we still need to make sure S_e is satisfied. In particular, we want $\Phi_e^c(n) \neq r(n)$ for this value n. Hence, this will give us another restriction on the value of $r_s(n)$.

In determining the value of $\Phi_e^c(n)$, we have two cases to consider. Either we can

extend $\widehat{c}_s$ to some sequence α so that $\Phi_e^\alpha(n)\downarrow$. That is, at some point our program will halt and output a value, and we can extend our sequence to that point. Or, there is no sequence α that extends $\widehat{c}_s$ for which $\Phi_e^\alpha(n)\downarrow$. That is, our program will never halt and output a value. For example, it may have entered an infinite loop in the first few values of $\widehat{c}_s$, in which case no addition of a sequence will alter the fact that $\Phi_e^\alpha(n)\uparrow$ for some n. We will look at these two cases individually and define c_{s+1} and r_{s+1} for each case.

FIGURE 4.0.2



Case 1: There is a sequence α such that $\widehat{c}_s \subseteq \alpha$ and $\Phi_e^\alpha(n)\downarrow$.

Fix $\alpha \supseteq \widehat{c}_s$ such that $\Phi_e^\alpha(n)\downarrow$. Then we can set

$$c_{s+1} = \alpha.$$

Notice that $\Phi_e^\alpha(n)\downarrow = \Phi_e^c(n)\downarrow$ since $c = \bigcup_{s\in\omega} c_s$ extends $c_{s+1} = \alpha$. Thus, $\Phi_e^c(n)$ is defined in this case. Hence, we will want to make sure that $\Phi_e^c(n)\downarrow \neq r(n)$.

First, note that because $c_{s+1}(n) = c_{s+1}(n{+}1)$, the set $\{c_{s+1}(n), c_{s+1}(n{+}1), \Phi_e^\alpha(n)\downarrow\}$ consists of at most two values. Let $j \in \{0, 1, 2\}$ be a number not in this set. Then we can set $\widehat{r}_s = r_s{}^\frown j$.

Up to this point we have defined $\widehat{c}_s = c_s{}^\frown c_s(n)$ and $\widehat{r}_s = r_s{}^\frown j$ so that $\widehat{r}_s$ beats $\widehat{c}_s$ and $|\widehat{c}_s| = |\widehat{r}_s| + 1$. Also note that c_{s+1} is an extension of c_s. Therefore, by Lemma 1 (page 11), there exists a sequence $\beta \supseteq \widehat{r}_s$ such that $|c_{s+1}| = |\beta| + 1$ and β beats

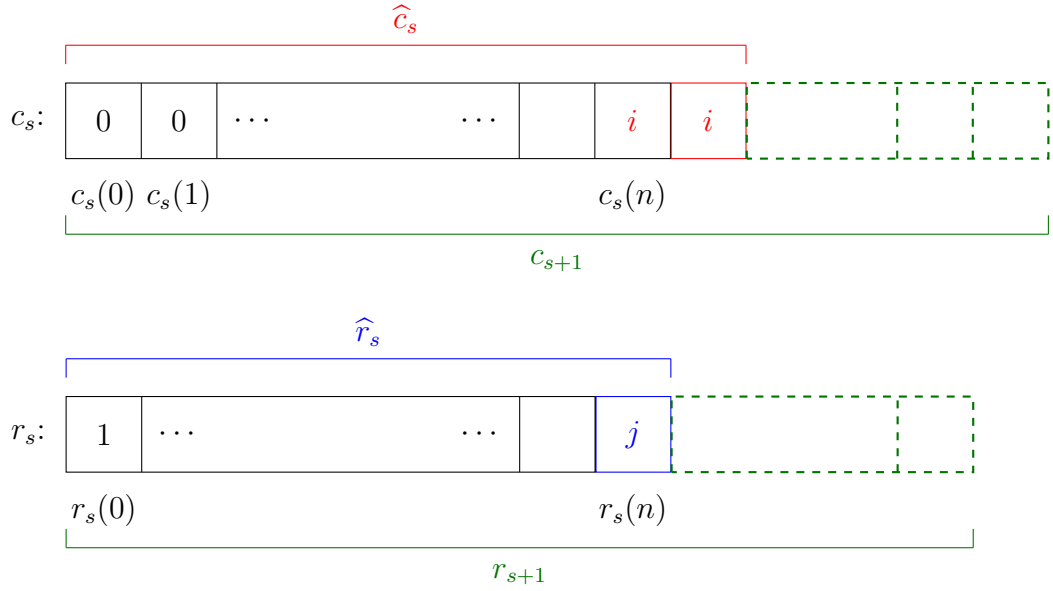
c_{s+1}. Thus, by setting

$$r_{s+1} = \beta$$

we get that $|c_{s+1}| = |r_{s+1}|+1$ and r_{s+1} beats c_{s+1}. See Figure 4.0.3 for a visualization.

Also notice that S_e is satisfied here. In particular, because $c_{s+1} \subseteq c$ and $r_{s+1} \subseteq r$ we get that $\Phi_e^c(n)\downarrow = \Phi_e^{c_{s+1}}(n)\downarrow \neq r_{s+1}(n) = r(n)$ and, consequently, $\Phi_e^c(n)\downarrow \neq r(n)$ for some n. Therefore, we have defined c_{s+1} and r_{s+1} so that S_e is satisfied in this case.

FIGURE 4.0.3



Case 2: There is no sequence α such that $\widehat{c}_s \subseteq \alpha$ and $\Phi_e^\alpha(n)\downarrow$.

In this case, we have that $\Phi_e^c(n)\uparrow$ for any infinite c extending $\widehat{c}_s$. In order to define $r(n)$, we needed to define $\Phi_e^c(n)$ first, but it is not defined in this case. However, notice that S_e is still satisfied. Namely, $\Phi_e^c(n)\uparrow$ means $\Phi_e^c \neq r$. Thus, we satisfy S_e

with no extra restriction imposed on the value for $r(n)$. So we can simply set

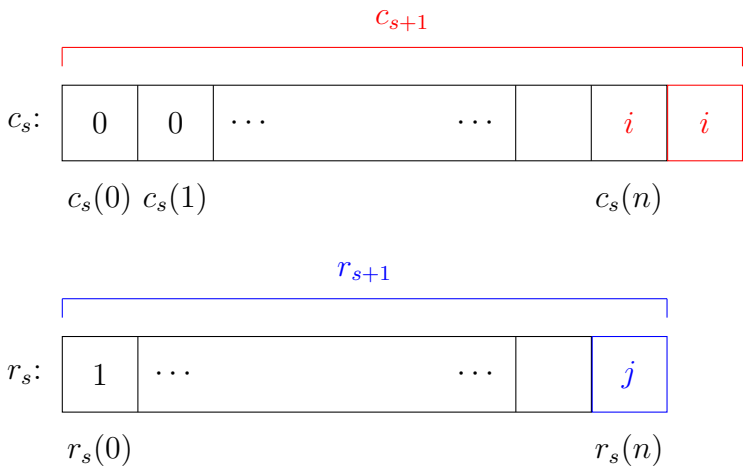
$$c_{s+1} = \widehat{c_s} = c_s \frown c_s(n)$$

and, for some $j \neq c_s(n)$,

$$r_{s+1} = r_s \frown j.$$

Then we get that $|c_{s+1}| = |r_{s+1}| + 1$ and r_{s+1} beats c_{s+1}. See Figure 4.0.4 for a visualization. Thus, we have found r_{s+1} and c_{s+1} so that S_e is satisfied in this case.

FIGURE 4.0.4



In both Case 1 and Case 2, we constructed c_{s+1} and r_{s+1} so that r_{s+1} beats c_{s+1}, $|c_{s+1}| = |r_{s+1}| + 1$, and $\Phi_e^c \neq r$. Overall, in the case where $s = 2e + 1$, we determined r_{s+1} and c_{s+1} so that S_e is satisfied.

Construction at stage $s + 1$ when $s = 2e + 2$:

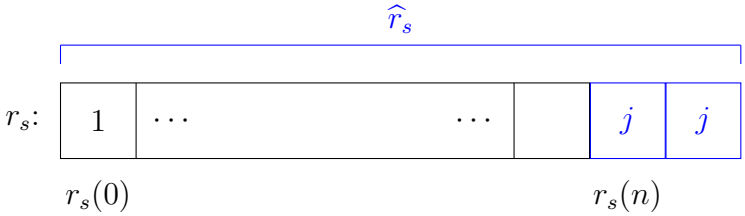
Let $n = |r_s|$. Our goal now is to extend r_s to r_{s+1} while satisfying R_e as well as having r_{s+1} beat c_{s+1} and $|c_{s+1}| = |r_{s+1}| + 1$. By making sure these are satisfied, certain restrictions will now be placed on the extension of r_s.

We can first focus on extending r_s to that is continues to beat c_s. In particular,

let's look at how we can extend r_s so that the value $c_s(n+1)$ has the most choices. We want that $c_s(n+1) \neq r_s(n)$ and $c_s(n+1) \neq r_s(n+1)$ so r_s beats c_s. Thus, we can set $r_s(n) = r_s(n+1)$. This way, we create only one restriction on the value for $c_s(n+1)$. Notice that neither $r_s(n)$ not $r_s(n+1)$ is defined yet. The only current restriction on these values is that they are not equal to $c_s(n)$.

Let $\widehat{r}_s = r_s \frown j \frown j$ where $j \neq c_s(n)$ (see Figure 4.0.5). Now we can look at extending $\widehat{r}_s$ to r_{s+1}. We will need to determine the value for $\Phi_e^r(n+1)$ since we need to satisfy R_e. In particular, we want $\Phi_e^r(n+1) \neq c(n+1)$ for this value n. Hence, this will give us another restriction on the value for $c_s(n+1)$.

FIGURE 4.0.5



When determining the value of $\Phi_e^r(n+1)$, we have two cases to consider. Either we can extend $\widehat{r}_s$ to some sequence β so that $\Phi_e^\beta(n+1)\downarrow$. Or, there is no sequence β that extends $\widehat{r}_s$ for which $\Phi_e^\beta(n+1)\downarrow$. We can look at these two cases individually and define c_{s+1} and r_{s+1} for each.

Case 1: There is a sequence β such that $\widehat{r}_s \subseteq \beta$ and $\Phi_e^\beta(n+1)\downarrow$.

Fix $\beta \supseteq \widehat{r}_s$ such that $\Phi_e^\beta(n+1)\downarrow$. Then we can set

$$r_{s+1} = \beta.$$

Notice that $\Phi_e^\beta(n+1)\downarrow = \Phi_e^r(n+1)\downarrow$ since $r = \bigcup_{s\in\omega} r_s$ extends $r_{s+1} = \beta$. Thus, $\Phi_e^r(n+1)$ is defined in this case. Hence, we will want to make sure that $\Phi_e^r(n+1)\downarrow \neq c(n+1)$.

First, note that the set $\{r_{s+1}(n), r_{s+1}(n+1), \Phi_e^{\beta}(n+1)\downarrow\}$ consists of at most two values since $r_{s+1}(n) = r_{s+1}(n+1)$. Let $i \in \{0,1,2\}$ be a number not in this set. Then we can set $\widehat{c}_s = c_s \frown i \frown i$.

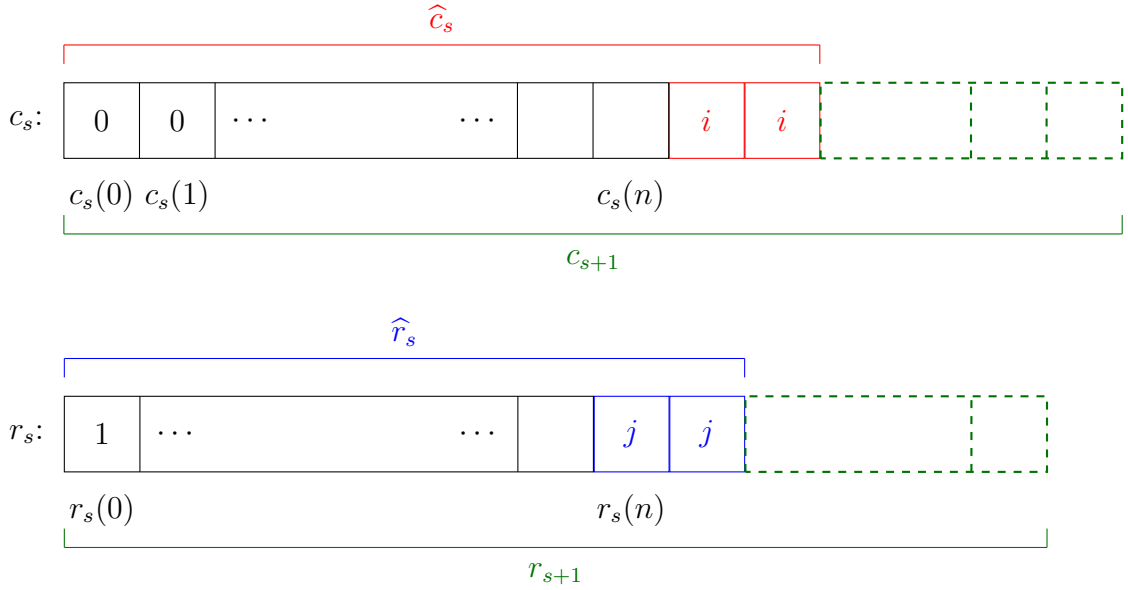
Up to this point, we have defined $\widehat{r}_s = r_s \frown j \frown j$ and $\widehat{c}_s = c_s \frown i \frown i$ so that $\widehat{r}_s$ beats $\widehat{c}_s$ and $|\widehat{c}_s| = |\widehat{r}_s| + 1$. Also note that r_{s+1} is an extension of r_s. Therefore, by Lemma 2 (page 11), there exists a sequence $\alpha \supseteq \widehat{c}_s$ such that $|\alpha| = |r_{s+1}|+1$ and r_{s+1} beats α. Thus, by setting

$$c_{s+1} = \alpha$$

we get that $|c_{s+1}| = |r_{s+1}|+1$ and r_{s+1} beats c_{s+1}. See Figure 4.0.6 for a visualization.

Also note that R_e is satisfied here. In particular, because $c_{s+1} \subseteq c$ and $r_{s+1} \subseteq r$, we get that $\Phi_e^r(n+1)\downarrow = \Phi_e^{r_{s+1}}(n+1)\downarrow \neq c_{s+1}(n+1) = c(n+1)$ and, consequently, $\Phi_e^r(n+1)\downarrow \neq c$. Therefore, we have defined c_{s+1} and r_{s+1} so that R_e is satisfied.

FIGURE 4.0.6



Case 2: There is no sequence β such that $\widehat{r}_s \subseteq \beta$ and $\Phi_e^{\beta}(n+1)\downarrow$.

In this case, we have that $\Phi_e^r(n+1)\uparrow$ for any infinite r extending $\widehat{r}_s$. In order to define $c_s(n+1)$, we needed to define $\Phi_e^r(n+1)$ first, but it is not defined in this case. However, R_e is still satisfied since $\Phi_e^r(n+1)\uparrow$ means that $\Phi_e^r \neq c$. Thus, we satisfy R_e with no extra restriction imposed on the value of $c(n+1)$. So we can set

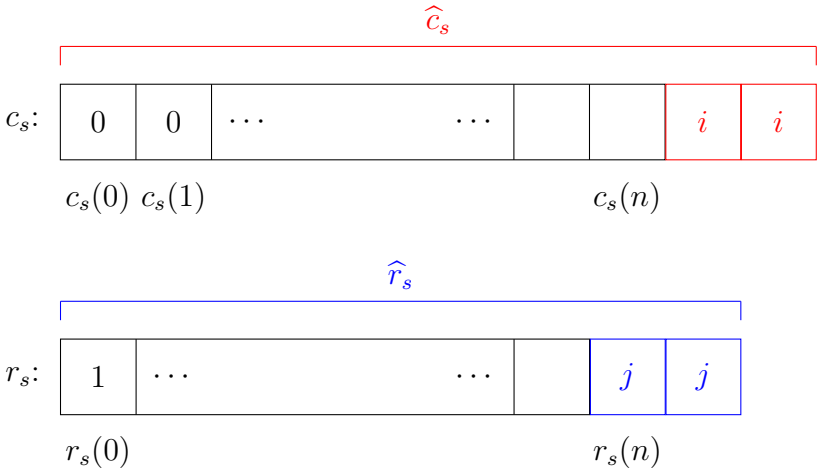
$$r_{s+1} = \widehat{r}_s = r_s \frown j \frown j$$

for some $j \neq c_s(n)$ and set

$$c_{s+1} = c_s \frown i \frown i$$

for some $i \neq r_s(n)$. Then we get that $|c_{s+1}| = |r_{s+1}| + 1$ and r_{s+1} beats c_{s+1}. See Figure 4.0.7 for the picture. Thus, we have found r_{s+1} and c_{s+1} so that R_e is satisfied.

FIGURE 4.0.7



In both Case 1 and 2, we constructed c_{s+1} and r_{s+1} so that r_{s+1} beats c_{s+1}, $|c_{s+1}| = |r_{s+1}| + 1$, and $\Phi_e^r \neq c$. Overall, when $s = 2e+2$, we determined r_{s+1} and c_{s+1} so that R_e is satisfied.

□

With the result from Theorem 4.0.1 in mind, we might hope to also get the result that for all cop sequences c such that the set $\{i : c(i) = c(i+1)\}$ is infinite, there exists a robber sequence r such that r beats c and $c \not\leq_T r$. However, as we will see with our next result, this is not always the case.

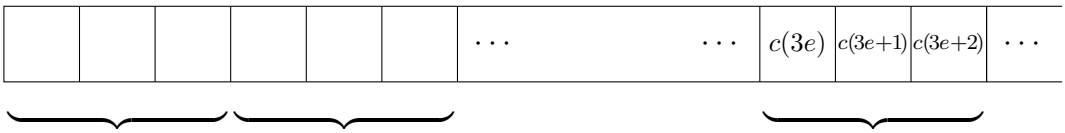
Theorem 4.0.2. *There is a noncomputable cop sequence c such that the set $\{i : c(i) = c(i+1)\}$ is infinite and $c \leq_T r$ for all robber sequences r that beat c.*

Proof. We want to construct a cop sequence c so that three conditions are satisfied.

1. c is noncomputable
2. $\{i : c(i) = c(i+1)\}$ is infinite
3. $c \leq_T r$ for all r that beat c

We can build such a cop sequence in blocks of three, defining $c(3e), c(3e+1)$, and $c(3e+2)$ for all indexes e. In particular, at $c(3e)$ we will try to satisfy condition 1, at $c(3e+1)$ we will try to satisfy condition 2, and at $c(3e+2)$ we will try to satisfy condition 3.

FIGURE 4.0.8: Constructing c in blocks of three



First, in order to guarantee condition 1, we want to satisfy

$$R_e : c \neq \Phi_e.$$

The idea here is to pick an n to use for R_e. Then if $\Phi_e(n)\downarrow$, we can define $c(n)$ so that $c(n) \neq \Phi_e(n)\downarrow$. If, on the other hand, $\Phi_e(n)\uparrow$, then R_e is immediately satisfied

with no restriction on $c(n)$. In general, for every index e, we can define

$$c(3e) = \begin{cases} 0 & \text{if } \Phi_e(3e)\downarrow > 0 \text{ or } \Phi_e(3e)\uparrow \\ 1 & \text{if } \Phi_e(3e)\downarrow = 0 \end{cases}.$$

With this we satisfy $R_e : c \neq \Phi_e$ since either $\Phi_e(3e)\uparrow$ or $\Phi_e(3e)\downarrow \neq c(3e)$. And so we have defined $c(3e)$ for all e so that c is noncomputable.

Next, we wish to satisfy condition 2 by making sure the set $\{i : c(i) = c(i+1)\}$ is infinite. By repeating a value in each of the blocks of three, this will guarantee a value will be repeated in c infinitely often. In particular, we can set $c(3e+1) = c(3e)$. Note that this means $c(3e+1) = c(3e) = 0$ or $c(3e+1) = c(3e) = 1$ based on how we defined $c(3e)$. Thus, we have defined $c(3e+1)$ so that c has infinitely many values that repeat.

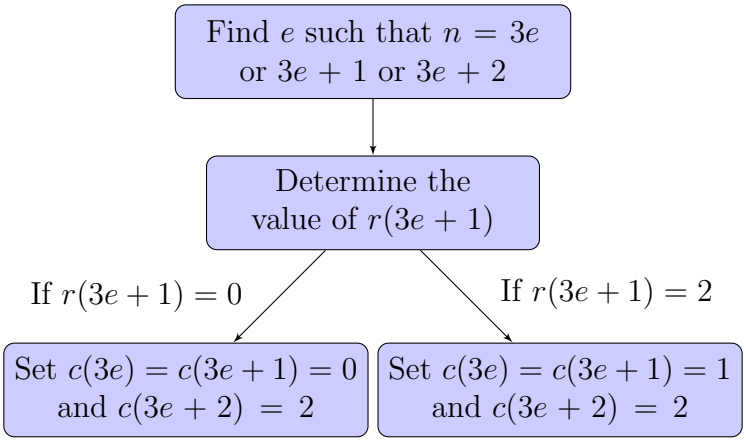
Lastly, we want to satisfy condition 3 by filling in $c(3e+2)$ so that $c \leq_T r$ for all r that beat c. That is, we need that an r that beats c can recover the values of $c(3e) = c(3e+1)$ and $c(3e+2)$. The value of $c(3e+2)$ can be assigned in an algorithmic way so that from $r(3e+1)$, we can recover the values of $c(3e+1)$ and $c(3e+2)$.

To see how to do this, assume that r beats c. Then $r(3e+1) \neq c(3e+1)$ and $r(3e+1) \neq c(3e+2)$. Note that $c(3e+1) = c(3e) = 0$ or $c(3e+1) = c(3e) = 1$. We will look at how to define $r(3e+1)$ in each of these cases. That is, we want to assign a value to $c(3e+2)$ so that $r(3e+1)$ is determined. First, if $c(3e+1) = c(3e) = 0$, then we can set $c(3e+2) = 1$. This will guarantee that $r(3e+1) = 2$. Second, if $c(3e+1) = c(3e) = 1$, then we can set $c(3e+2) = 2$. This will guarantee that $r(3e+1) = 0$. In general, we can set $c(3e+2) = c(3e+1) + 1$.

If we wanted to determine c from r, then we can look at these two cases. For

example, say we wanted to determine $c(10)$ given r. First note that $10 \equiv 1 \bmod 3$. That is, 10 is the $(3e+1)$-spot for $e = 3$. So given a value for $c(3e+1)$, we need to look at $r(3e+1)$, in this case $r(10)$. If $r(10) = 2$, then we know $c(10) = 0$ (and hence $c(9) = 0$ and $c(11) = 1$). If $r(10) = 0$, then we know $c(10) = 1$ (and hence $c(9) = 1$ and $c(11) = 2$). In general, to determine the value of $c(n)$ from r, we can use the algorithm in Figure 4.0.9. Hence, $c \leq_T r$ for all r that beat c.

FIGURE 4.0.9



Altogether, we have constructed c by defining $c(3e)$, $c(3e+1)$, and $c(3e+2)$ for all indexes e so that c is noncomputable, the set $\{i : c(i) = c(i+1)\}$ is infinite, and $c \leq_T r$ for all r that beat c.

□

Bibliography

[1] Anthony Bonato, *An invitation to pursuit-invasion games and graph theory*, Student Mathematical Library, vol. 97, American Mathematical Society, 2020.

[2] Anthony Bonato and Richard J. Nowakowski, *The game of cops and robbers on graphs*, Student Mathematical Library, vol. 61, American Mathematical Society, 2010.

[3] Robert I. Soare, *Turing computability: Theory and applications*, Springer, 2016.