Domain Design for the Cops and Robbers Problem

Connor Little, Christian Muise

{ connor.little, christian.muise } @queensu.ca
Queen's University
Kingston ON, Canada

Abstract

The cops and robbers problem is a well-researched problem in graph theory. The problem consists of a robber and one or more cops placed on a graph. Taking turns moving throughout the graph, the cops try to capture the robbers. We model a non-deterministic version of this problem using automated planning. We then extend it using numerous variations within the literature. These variations include the variable speed robber and the friendly robber. Given a class of graphs, we can exhaustively generate domain-problem pairs for all problems and use these pairs to test the properties of the class – a systemic and exhaustive compilation of all planning models for a class of graphs through an automatic translation. Our work demonstrates planning is capable of solving the cops and robbers problem efficiently.

Introduction

The cops and robbers problem is a popular problem in the field of graph theory (Bonato 2011). It simulates a chase between a robber and a specified number of cops across a graph. The robber tries to avoid capture forever while the cops want to catch the robber. Despite its simple rules, a myriad of research has revolved around both techniques to solve the game and variations on extending it.

To translate the problem to PDDL, we introduce a new perspective on the problem: the non-deterministic cop. This variation of the problem has each cop make non-deterministic moves each turn. As such, the planner must make policies for all reachable states as opposed to just a single decisive policy. By coupling this with a neutral initial state that allows the cops and robbers to choose their starting configurations, we can search for a policy that considers every configuration of a graph simultaneously. This allows the model to test and explore the environment exhaustively.

More precisely, we translate the problem into a fully observable non-deterministic (FOND) problem. Under these rules, all state information is known and there is no hidden information. There is nothing to be gained by limiting what information the cops and the robbers have access to, outside introducing new variations. The outcomes of actions, however, are not known. The cop's actions are non-deterministic in that when moving a random, but not probabilistic, node is chosen.

Unfortunately, FOND models are unable to be compared in terms of optimality. Instead, we will take a subjective approach to how these policies are utilized. The benefit of automated planning is that it allows us to efficiently create policies for the robber. If a policy exists such that the robber can evade capture, we can determine many attributes of that problem instance.

In addition to creating the problems and domains for the standard version, we explore some of the common variations of the cops and robbers' problem. We have implemented two versions: the friendly robber and the variable move speed robber (Frieze, Krivelevich, and Loh 2012). The variable move speed robber can take multiple moves per turn. This allows the robber to cover more ground and avoid capture where otherwise not possible. The friendly robber has the objective to be captured as soon as possible. This inverts the problem to be one of catching all possible cops instead of avoiding all possible cops. Both variations have been covered in the literature, the former extensively.

Finally, we take classes of graphs and generate exhaustive domains and problems for these classes. We use these generated sets to test the graph properties. Our work shows promising results for using mathematical planning to solve and explore properties of classes of problems.

Background

The Cops and Robbers Problem

The cops and robbers problem is a variation on the pursuit and evasion problem. It was first introduced into the literature by Nowakowski and Winkler (Nowakowski and Winkler 1983) in 1983, and by Quilliot (Quilliot 1986). It takes place on a graph and with perfect information played with two adversarial forces. The two main entities in which the game revolves are the cops and the robbers. There is one robber and $k \geq 1$ cops. Starting with the robber, they take turns making moves across a connected graph. Each player can only move to adjacent nodes and can only move unit length. The robber's goal is to avoid capture indefinitely, while the cop's goal is to capture the robber. Capture can be defined to be when a robber's only legal move is to move onto a node occupied by a cop, or if they occupy the same node.

A larger goal of the game is to determine if the graph is a cop-win or a robber-win. A robber-win is a graph such that

the robber has a strategy to avoid the cops indefinitely. The cop-win graphs are those in which the cop(s) can always capture the robber. Another common question is to determine the cop number of a graph. A cop number is the minimum number of cops required to capture the robber. One can imagine putting a cop on every single node of the graph, guaranteeing capture in one move. This would lead to the upper bound of the cop number being the number of nodes in a graph. The lower bound is often unknown.

Variations on the Cops and Problems

Frieze et al. (Frieze, Krivelevich, and Loh 2012) cover many of the popular variations on the problem. The first variation that is covered is that of the directed graph. In this variation, the graph has one-way edges and therefore has a much more restricted search space. The second variation is the fast robber. With this condition, the robber may move a variable number of spaces each turn. The maximum distance is predecided. This allows the robber much more maneuverability when avoiding the cops. This can be extended into the infinitely fast robber, which may jump to any unimpeded node on the graph.

The most similar variant that has been covered in the literature on the cops and robbers problem is the drunk robber variant. It was first introduced by Komarov and Winkler (Komarov and Winkler 2013). In this variation, the robber employs no strategy, instead wandering aimlessly throughout the graph. This has been expanded to the "tipsy cop and drunk robber" in which the robber still moves aimlessly and the cop may make knowledgable moves or may move randomly (Harris et al. 2020). There is also the case of the invisible robber by Kehagias et al. (Kehagias, Mitsche, and Prałat 2013). This variant has the robber's position unknown until capture. While this does make the cop's movement more unpredictable, there are still strategies they may employ. None of these variations fully capture the non-deterministic cop.

Automated Planning and Evasion Problems

In the domain of automated planning work on the cops and robbers problem, and similar games, is extremely limited. Even outside the domain of automated planning, utilizing AI to look into the cops and robbers problem has not been explored in depth. To our knowledge, there have been no such attempts. Instead, we may focus on work in adjacent domains, such as pursuit and evasion. Nussbaum and Yörükçü (Nussbaum and Yörükçü 2015) worked on the problem of Moving Target Search (MTS) in which an agent attempts to capture another non-stationary agent. The key inclusion in their algorithm was using environment abstraction to decrease the search space without compromising on the cost of the algorithm. As such, they were able to improve upon all compared algorithms in all tested environments.

Planning under visibility constraints is also a common occurrence in the pursuit-evasion planning literature. Guibas et al. (Guibas et al. 1999) and Stiffler and Kane (Stiffler and O'Kane 2020) both focus on the partially-observable pursuit and evasion problem. Guibas et al. (Guibas et al. 1999) are able to both introduce an algorithm for solving the problem and also introduce bounds for necessary conditions to capture the evader. Their paper uses a polygonal environment instead of a graph. Stiffler and Kane (Stiffler and O'Kane 2020) look at the domain of search and rescue. The evaders are victims who must be caught (rescued) and may move unpredictably. A key feature is that the algorithm must be able to guarantee each evader is found. They utilize a forward search style algorithm as well as an algorithm that can simulate evader trajectories to aid in path computation.

Macindoe et al. (Macindoe, Kaelbling, and Lozano-Pérez 2023) do work on a variation of the cops and robbers problem within an automated planning framework. They work on a partially observable Markov Decision Process to build sidekicks for the cops. This turns it into a multi-agent problem. They aim to have their sidekick assist the cop in capturing the robber by planning in belief space. Their sidekick is able to interpret what a human player would aim to do and assists in capturing the robber faster than simply committing to a single path would succeed.

A Planning Model for Cops and Robbers

Our model uses the PDDL language and the PR2 planner (Muise, McIlraith, and Beck 2023). Each problem is given its own domain file to prevent exponential blow-up of the search space. For every graph, the problem and domain files are generated programmatically.

The model we present is for the non-deterministic cops and robbers problem. No variations are included in this model unless explicitly mentioned. The standard version of cops and robbers starts with a graph $G = \{V, E\}$, where V is a set of vertices and E is a set of edges. A robber is placed on a vertex v. One or more cops are placed on vertices drawn from $V \backslash \{v\}$. On each iteration of the game the robber makes a move along one of the edges, and then the robbers make a move. This is done until the termination condition of the robber being unable to traverse an edge without being on an occupied node, or until the depth limit is reached.

To begin, we define the following predicates:

- (at ?x - entity ?y - location)
  - This fluent is true if the entity occupies said location.
- (edge ?x ?y - location)
  - This fluent defines the architecture of the graph.
- (turn ?x - cop)
  - This fluent determines which entities may make a move. It is the robber's turn if all turns are set to false.
- (caught)
  - This fluent is true the robber is ever caught.
- (survived)
  - This fluent determines if the robber is still alive.
- (done)
  - This fluent is a flag for ending the game.
- (nil)
  - This fluent does nothing. It is a placeholder for the identity action.

- (move0), (move1), etc
  - These fluent counts the number of moves a robber may make.

Next, we may define some of the important actions.

Algorithm 1: Robber Movement

```
1: (:action robber_move
2:       :parameters (?x ?y - location)
3:       :precondition (and
4:            (at rob1 ?x)
5:            (edge ?x ?y)
6:            (forall (?c - cop) (not (turn ?c)))
7:       :effect (and
8:            (when (exists (?c - cop) (at ?c ?y)) (caught))
9:            (not (at rob1 ?x))
10:           (at rob1 ?y)
11:           (when (move0) (and (not move0) (move1)
12:           (when (moveK) (and (not moveK) (forall (?c -
    cop) (turn ?c)))
13:           (oneof (survived) (nil))
14:      )
15: )
```

Algorithm 1 defines the movement of the robber. Since there is only 1 robber only the locations are needed as a parameter. Line 4 ensures the robber is at the first location. Line 5 ensures there is a connection between nodes. Line 6 ensures it is not the cop's turn. The effect is as follows. Line 8 checks for collision between the robber and the cop, to determine the end of the game. Lines 9 and 10 remove the robber from the first node and place them at the second. Line 11 increments the move counter for the robber. On the last move, on line 12, all cops have their turn set to be true. Lastly, line 13 non-deterministically ends the game. This is necessary as we don't want the game to end once the robber is caught. The objective of the algorithm is to generate a plan for the robber to avoid as long as possible. By choosing this method, the robber is incentivized to continuously avoid the cop without directly declaring a utility or reward function.

To add the variation of multiple moves, an additional set of statements are added. Each move is given a fluent: move0, move1, ..., moven. The first time a robber makes a move, move0 will be the only fluent set to true. For each move, a line is called that increments the move fluents. The first iteration will set move0 to false and move1 to true, the second will set move1 to false and move2 to true, and so on. Only on the last call will the cops' turns be set to true. In addition, each cop now must set move0 to be true. This adds some redundant computation, but as we demonstrate in the results is still an efficient implementation of this variation.

Algorithm 2 defines the movement of the cop. The number of nodes that a node can have adjacent is bounded. We need to numerically quantify how many edges a node has before problem creation. The number of edge preconditions will grow linearly with the number of adjacent nodes. The number of inequality preconditions will grow quadratically with respect to the triangle numbers. The number of

Algorithm 2: Cop Movement Unoptimized

```
1: (:action cop_move
2:       (?c - cop ?x ?y ?z - location)
3:       :precondition (and
4:            (turn ?c)
5:            (num_edges ?x 2)
6:            (at ?c ?x)
7:            (edge ?x ?y)
8:            (edge ?x ?z)
9:            (not (= ?y ?z))
10:      :effect (and
11:           (not (turn ?c))
12:           (move0)
13:           (not (at ?c ?x))
14:           (oneof
15:
16:           (and (at ?c ?y) (when (at rob1 ?y) (caught)) )
17:
18:           (and (at ?c ?z) (when (at rob1 ?z) (caught)) )
19:           )
20:      )
21: )
```

Algorithm 3: Cop Movement

```
1: (:action cop_move_node_2
2:       (?c - cop)
3:       :precondition (and
4:            (turn ?c)
5:            (at ?c node2)
6:       :effect (and
7:            (not (turn ?c))
8:            (move0)
9:            (not (at ?c node2))
10:           (oneof
11:
12:           (and (at ?c node1) (when (at rob1 node1)
    (caught)) )
13:
14:           (and (at ?c node3) (when (at rob1 node3)
    (caught)) )
15:           )
16:      )
17: )
```

branches in the oneof clause will also grow linearly. To avoid this growth, we use the programmatically generated version for each graph. This creates an action for every node that a cop may take by making each problem have a custom domain. Algorithm 3 shows an example of node2 with connections to node1 and node3. Both versions have some slight redundancy in that they always set move0 to be true. This is required to let the robber's move cycle restart.

In addition to these actions, some supplementary actions are provided. The terminate game action requires 'survived' and not 'caught'. It will return 'done'. The robber is also allowed to stay in place with a stay action. Each variant of the problem adds or alters one of the actions as well. The friendly variant creates a new terminal action with just requires 'caught', while a variable speed robber increases the number of move fluents required.

When running the algorithm a strong cyclic or strong acyclic solution corresponds to a robber win, while no strong solution found corresponds to a cop win. In practice, the result will always be a strong cyclic and not a strong acyclic as the game only is a robber win if they can avoid capture in all possible states. For a non-cyclic solution to be found the graph would need to be infinitely large.

Furthermore, to generate the domains for each instance, we also define the problem files for each of the problems. This comes with some customization to control what we solve for. The first addition is the inclusion of a node n0 that is outside of the graph. This node has a unidirectional connection with all possible nodes and cannot be returned to. Beginning with all of the cops, all entities make a move starting at n0 and placing themselves on the graph. With the non-deterministic properties of the cops, this allows all starting configurations of cops to be tested in implicit parallel. The problem files can be generated with known starting locations as well. We also allow the user to generate instances with variable numbers of cops and robbers. Lastly, digraphs and unconnected nodes are permitted.

Evaluation

All experiments were conducted on a Dell XPS 17 9720. A 12th Gen Intel(R) Core(TM) i7-12700H was used with 32 GB of available RAM.

An evaluation of the cops and robber's problem with planning will be broken into two sections. The first section will apply the technique to classes of problems to see how the planner fares empirically, as well as demonstrate the tool's usefulness for property testing and generation. The second part will look at individual problems. This section will be devoted to interpreting the policy output as well as determining the limits of the algorithm.

Property Testing

Many of the open questions regarding cops and robbers are whether certain classes of graphs obey certain properties. A specific example would be whether or not a specific group of graphs has a cop number k (Berarducci and Intrigila 1993). We can evaluate questions of this nature through exhaustive exploration of all configurations of the cops and robbers problem.

We evaluated our algorithm on 2 different classes of graphs: Seven vertex-connected graphs and 6 vertex Eulerian digraphs. Eulerian graphs are graphs that have the property that all nodes have an even number of connections. The exhaustive list of graphs comes from a database constructed by McKay (McKay 2023). These classes were chosen based on their size, as well as some properties they hold. Both classes are connected, which means that every node is reachable from every other node. Without this property, the cops and robbers problem would not work as the robber could hide on a disconnected graph. The Eulerian graph was also chosen to test digraphs within our methodology. Digraphs are graphs that have directed edges instead of undirected ones. Backtracking is not allowed on these graphs. Finally, both classes of graphs have no reflexive edges.

Each class of graphs is explored with various variations applied. In all variations, each entity starts outside the map and chooses a location with the robber choosing last. This allows every variation of the game to be considered. The standard variation involves the robber having one move and avoiding the cops. The friendly variation incentivizes the robber to be caught. Each additional move is an extra move a robber may do. Lastly, we increase the number of cops to determine the cop number. The results can be found in Tables 1 and 2.

Each run measures the following statistics. The first is the amount of time it takes to run all graphs in the class exhaustively. The next column keeps track of the number of robber win graphs. This is the number of graphs in which a robber can win by avoiding the cop indefinitely. The robber can always win in the friendly variation as there is a finite number of states that can be reached. Every state has a possible state that can be reached in which the robber wins, so all states can eventually lead to a win. This is a safe problem. The next column measures the total number of weak searches that are called on average. This is a measure of how many fast downward planner calls are needed. As the number of moves increases, so too does the number of weak searches on average. Solution size is the average number of states in the controller. This can represent how complex the policies are. FOND problems have no definition of optimality other than one policy dominating another, so this gives some insight into how our policy is but it cannot be used to definitively say one area is better than another. The next column is the number of rounds that are computed on average. This, much like the total time, is another measure of computation. A round is determined by restarting the search on a new incumbent solution with the knowledge learned from previous searches. The more rounds that need to be computed, the more computation is done. The last two columns are forbidden state-action pairs (FSAPs) and poison count. The former is the number of pairs of states and actions that we penalize the algorithm for choosing. The latter is the number of states that have their descendants poisoned. The poisoning process disincentives searching down bad paths.

Some interesting patterns emerge when examining the different variations. One of the largest outliers is the effect of adding an additional cop. One additional cop more than tripled the runtime for the standard version in the first trial.

Variation	Total Time	Robber Wins	Weak Searches	Solution Size	Rounds	FSAPS	Poison Count
Standard	37m39.610s	450/853	361.17	54.21	10.09	197.42	158.48
Friendly	33m51.066s	853/853	13	10	1	0	0
2 Moves	45m18.327s	503/853	618.94	66.59	14.09	85.23	174.83
3 Moves	48m7.595s	528/835	998.47	84.13	16.61	96.01	212.66
2 Cops	115m30.003s	0/835	454.44	228.56	5.70	4139.79	403.86

Table 1: Results over all 7 vertex connected graphs

Variation	Total Time	Robber Wins	Weak Searches	Solution Size	Rounds	FSAPS	Poison Count
Standard	89m12.562s	86/2162	71.82	32.59	4.09	397.92	45.95
Friendly	92m49.273s	2162/2162	23.68	10.95	1.35	3.69	6.49
2 Moves	108m47.285s	91/2162	303.78	48.52	10.64	85.79	160.13
3 Moves	144m34.607s	115/2162	849.92	64.02	22.73	138.47	329.89
2 Cops	1075m48.238s	0/2162	527.64	278.06	4.76	12952.74	460.23

Table 2: Results over all 6 vertex Eulerian Digraphs

It also greatly increased the search space while requiring far fewer searches and rounds. This is in part because of the significant number of FSAPS. When adding a cop to the 6 node digraphs the runtime increased by a factor of 10.

Comparatively, increasing the number of moves the robber can make had a far lesser impact on the computational burden. It added an additional standard variation worth of weak searches, which makes sense intuitively. Interestingly, the number of FSAPs decreased. Our guess as to why this happens is that you can enter and leave a forbidden zone without causing a loss as you get two moves.

Lastly, increasing the number of cops on each version gives us the ability to calculate the cop number for the graphs. 403 connected 7 vertex graphs have a cop number of 1 and the rest have a cop number of 2. The digraph results also reveal that they have a maximum cop number of 2.

Individual Problems

Next, we narrow the focus to individual problems to more thoroughly examine their output. By focusing on singular problems we can evaluate the policies directly in terms of human intuition and interpretability. Here we would like to highlight 4 different problems that each offer a unique insight into how the algorithm generates policies. This will allow better utilization of the policies in practice. The first problem to look at is a single line in which the cop must chase the robber to the end. This one is important for understanding the robber's survivability as well as how to compute friendly robbers. The second is a square graph. This one again offers insight into the friendly robber and whether or not the robber can take advantage of hiding on a square. The third is a tree. This problem demonstrates the issue with backtracking as well as tests the limits of survivability. The fourth is a tree with a cyclic ring that can only be reached through backtracking. This tests whether or not a planner can form policies that backtrack if they lead to eventual success. Each problem discussed will be very small so that the policy may be interpretable.

The first problem to examine is the line chase problem. This problem starts with 6 nodes. The cop begins on the first not and the robber begins on the second. The policy can be seen in Figure 1.

The policy can be read in the following way. The policy begins with the action in the square box. From there one of two results can happen. Either the game ends as the robber survives, leading to the terminate node, or the cop must now make a move. After the cop moves, the robber may make a choice depending on the move that was made. If a node containing -1 is reached that means that the robber was caught and no strong cyclic solution exists.

The robber begins by moving away from the cop and down the line. The cop will then chase the robber down. The robber once again moves away from the cop and then the non-deterministic action is taken and the cop either moves towards the robber or back to the beginning. If the cop moves back to the beginning then the robber will move back towards the cop. The planner always tries to return to known states to minimize the solution space.

One notable aspect of this problem is the use of the wait action. The robber correctly stays at the final node unless it is safe for 2 actions to move toward the cop. This ensures that it survives until the cop reaches the final node. This is the optimal behavior in terms of surviving as long as possible.

This problem has unique considerations when considering the friendly robber. Due to the nature of the planner to return to known locations, the robber does not take the shortest path towards the robber. Instead, they repeatedly return to known positions until the cop makes novel choices. To avoid this scenario and generate the shortest plans, the additional parameter "–localize-enabled 0" may be added to the PR2 planner (Muise, McIlraith, and Beck 2023). This disables the check for known states and instead, the policy is returned where the robber moves down the line each move.

The second notable plan is the square. This is the shortest possible cyclic plan in which the robber has a winning strategy. A triangle is infeasible as it is a fully connected graph.

This graph shows that the robber is capable of waiting on its initial state to act on the cop's action. It will always stay

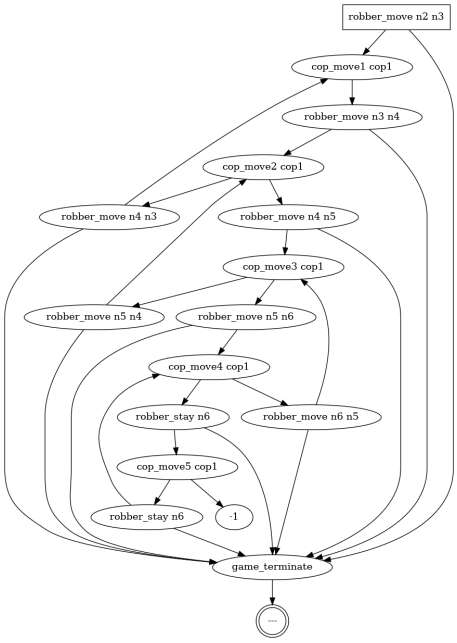

Figure 1: Policy to avoid capture on a straight line

in the opposite corner, which is intended behaviour. The full policy can be seen in Figure 2. This problem also gives interesting behavior when the robber is given multiple moves. If given 2 moves the robber will always wait on its second move, as a move would make it get captured the following turn. If given an odd number of moves, such as three, the robber will be instructed to make 2 moves which cancel each other out. It will move to a node only to move back. This pattern seems to be repeated in most multiple-move graphs. The planner's additional goal of returning to a goal state makes the addition of additional moves often negligible as the robber is incentivized to waste them, unless necessary.

The addition of a cycle also affects how the friendly variation works. We cannot use the additional parameter "–localize-enabled 0" as before. Because it's always possible for the cop to move away from the robber, we have to rely on returning to additional states. Unrolling the complete paths can lead to paths infinitely long, which is beyond the scope of this planner.

The third problem worth inspecting is a branching tree problem. This problem tests the robber's ability to make long-term decisions even if the outcome is the same. That is, can the robber choose the longest branch even if, in all cases, it will eventually be a cop win? The answer is yes, and the policy can be seen in Figure 3

The robber begins at the root of the tree, and the cop begins adjacent to the robber on the shortest branch. From this scenario, the robber is given the option of which branch to hide down. The robber picks the longest branch and successfully evades capture for the longest possible time.

The 4th problem to look at changes the prior problem in 2 different ways. Both involve making the branch the cop starts on the longest. The first version simply makes it the

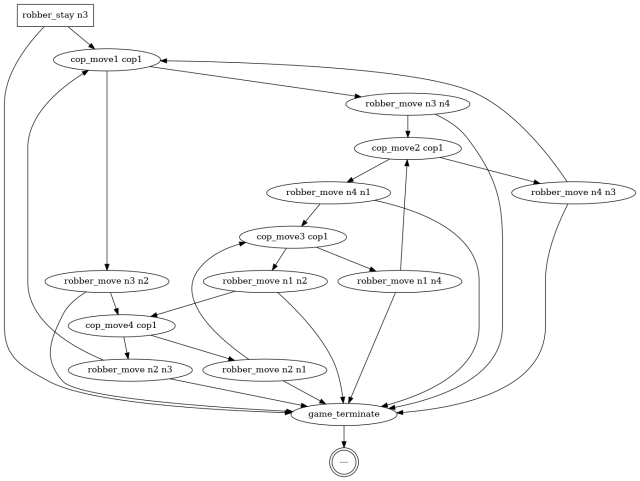

Figure 2: Policy to avoid capture on a square

longest by adding nodes so that the line is longer. The second version adds a loop of cycle 4 to the branch. This not only makes it the longest but ensures that if the robber were to reach the cycle they have a guaranteed win.

Unfortunately, the implementation of the domain and problem make the robber incapable to computing the backtracking required to descend down this path. The way to access the longest branch would be to wait for the cop to descend down a branch the robber did not access and backtrack to the longer one. In both cases the planner failed to discover this policy.

This failure to generate the policy also extended through parameter tuning. Increasing the depth of the search and the number of trials, as well as disabling known state returns all failed to cause the planner to backtrack. This proves that in its current state, the planner will not survive for the longest possible time.

This does not suggest that the planner cannot find solutions, however. In all backtracking cases, there is a possibility of guaranteed failure should a cop immediately descend down the chosen branch of the robber. Due to this, there is no strong cyclic solution that exists. We were unable to develop a problem that requires backtracking but also has a strong cyclic solution. Therefore, it is inconclusive if any solutions may be missed due to this aspect of the program.

Summary

The cops and robbers problems are a class of problems that are of significant interest in graph theory. This paper presents 3 main results. The first is the creation and implementation of a PDDL version of the cops and robbers problem through automatic domain and problem engineering. The addition of the non-deterministic cop and variable starts allows for an exhaustive exploration of the problem proposed. The second contribution is a proof of concept and exploration of property testing for classes of problems. We design a generation of domain problems based on graphs and exhaustively cover the search space. The planner is able to run on classes of instances and test hypotheses. It can also be

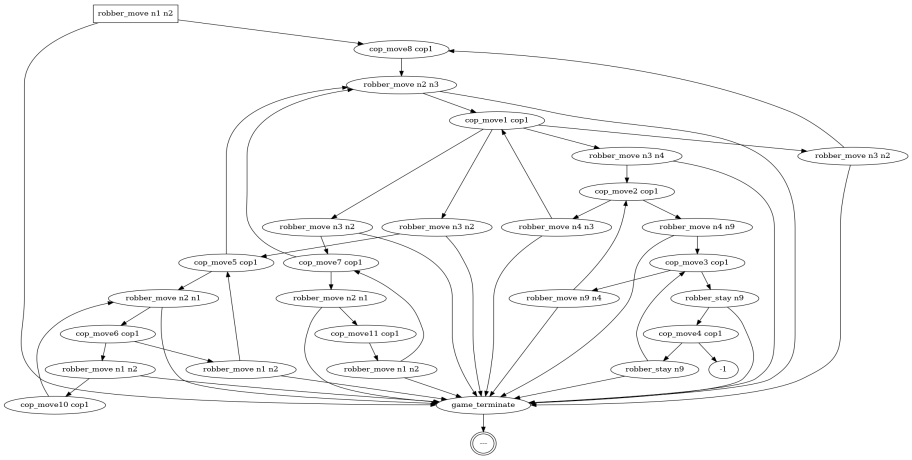

Figure 3: Policy to avoid capture on a Tree

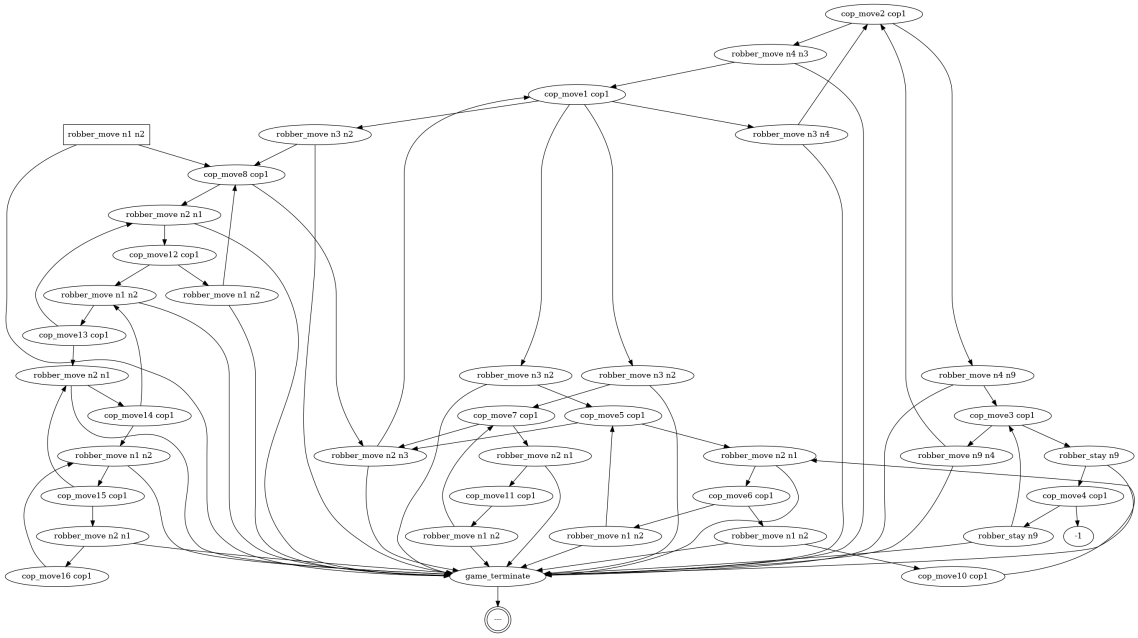

Figure 4: Policy to avoid capture on a tree with backtracking

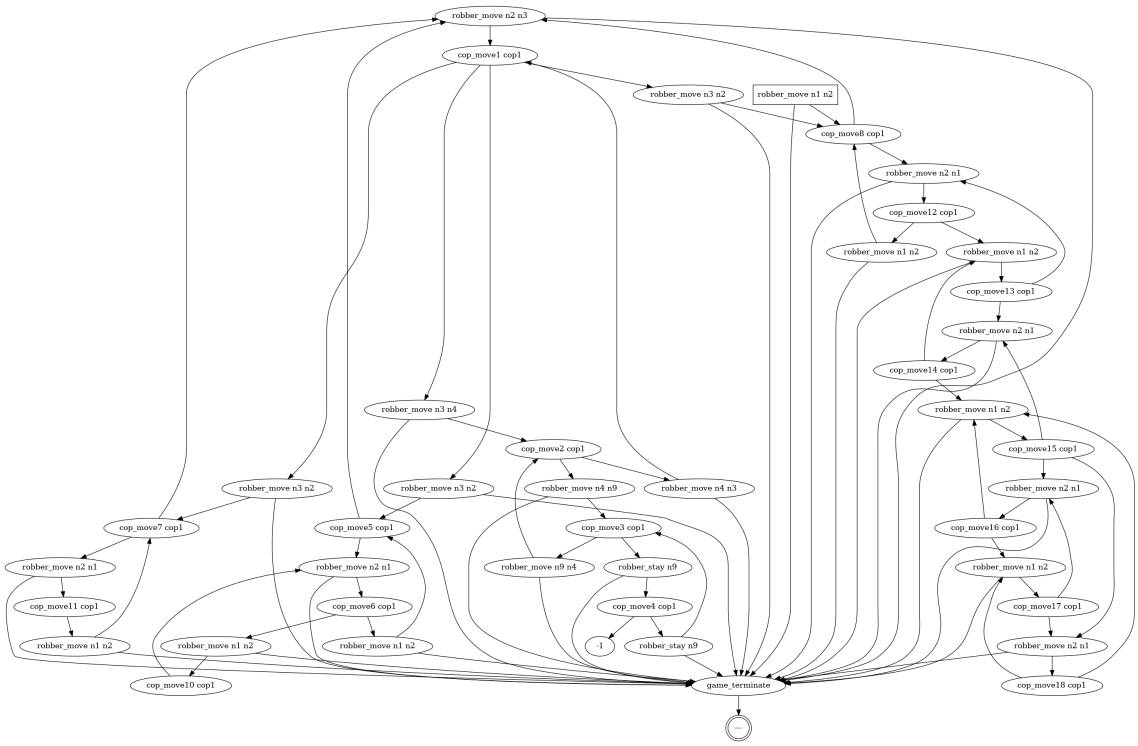

Figure 5: Policy to avoid capture on a tree with a cycle

used to determine bounds. The implementation is efficient when applied to small instances to explore entire classes or large individual problems. The final contribution is regarding policy interpretation. The policies can be extracted and understood while being intuitive. They follow the rules one would expect a robber to follow.

Ultimately, we have demonstrated how a systematic planning modeling of a mathematical problem setting can be used to explore properties of interest to the mathematics community. Our approach to the cops and robbers problem offers a unique perspective on the problem and allows researchers to prove properties of problems.

Moving forward we hope to explore additional variants of the cops and robbers problem. In this paper, we cover the friendly robber and the variable speed robber. While some variants such as the drunk robber would be superfluous with our implementation, most others should be integratable into our framework without substantial adjustment, such as a dynamic cop problem or an infinite speed robber. Many more variants are included in the literature and pose interesting problems in their own right. Further optimizations to the plan generation would also allow larger search spaces to be explored. Improving the implementation to explore larger classes of problems would provide much stronger use cases for the model. The current model is only tested on graphs with small nodes, and while graphs that contain more nodes are feasible, the time complexity of the problem is exponential and solving the whole class of graphs becomes difficult. We have shown that there are some cases in which human intuition does not match the plan, such as in the tree search with backtracking. With further exploration, we believe this can be improved.

References

Berarducci, A.; and Intrigila, B. 1993. On the Cop Number of a Graph. *Advances in Applied Mathematics*, 14(4): 389–403.

Bonato, A. 2011. *The game of cops and robbers on graphs*. American Mathematical Soc.

Frieze, A.; Krivelevich, M.; and Loh, P. S. 2012. Variations on cops and robbers. *Journal of Graph Theory*, 69: 383–402.

Guibas, L. J.; Latombe, J.-C.; LaValle, S. M.; Lin, D.; and Motwani, R. 1999. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry & Applications*, 9(04n05): 471–493.

Harris, P.; Insko, E.; Prieto-Langarica, A.; Stoisavljevic, R.; and Sullivan, S. 2020. Tipsy cop and drunken robber: a variant of the cop and robber game on graphs.

Kehagias, A.; Mitsche, D.; and Prałat, P. 2013. Cops and invisible robbers: The cost of drunkenness. *Theoretical Computer Science*, 481: 100–120.

Komarov, N.; and Winkler, P. 2013. Capturing the Drunk Robber on a Graph. *Electronic Journal of Combinatorics*, 21.

Macindoe, O.; Kaelbling, L. P.; and Lozano-Pérez, T. 2023. POMCoP: Belief Space Planning for Sidekicks in Cooperative Games.

McKay, B. 2023. Combinatorial Data. http://users.cecs.anu.edu.au/~bdm/data/. Accessed: 2023-11-12.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2023. PRP Rebooted: Advancing the State of the Art in FOND Planning. (arXiv:2312.11675). ArXiv:2312.11675 [cs].

Nowakowski, R.; and Winkler, P. 1983. VERTEX-TO-VERTEX PURSUIT IN A GRAPH.

Nussbaum, D.; and Yörükçü, A. 2015. Moving Target Search with Subgoal Graphs*.

Quilliot, A. 1986. Some Results about Pursuit Games on Metric Spaces Obtained Through Graph Theory Techniques. *European Journal of Combinatorics*, 7(1): 55–66.

Stiffler, N. M.; and O'Kane, J. M. 2020. Planning for robust visibility-based pursuit-evasion. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6641–6648.