Nemesis, an Escape Game in Graphs

Pierre Bergé
Université Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Antoine Dailly
Université Clermont Auvergne, INRAE, UR TSCF, 63000 Clermont-Ferrand, France

Yan Gerard[1]
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Abstract

We define a new escape game in graphs that we call *Nemesis*. The game is played on a graph having a subset of vertices labeled as exits and the goal of one of the two players, called the *fugitive*, is to reach one of these exit vertices. The second player, *i.e.* the fugitive adversary, is called *the Nemesis*. Her goal is to trap the fugitive in a connected component which does not contain any exit. At each round of the game, the fugitive moves from one vertex to an adjacent vertex. Then the Nemesis deletes one edge anywhere in the graph. The game ends when either the fugitive reached an exit or when he is in a connected component that does not contain any exit. In trees and graphs of maximum degree bounded by 3, Nemesis can be solved in linear time. For arbitrary graphs, we show that Nemesis is PSPACE-complete, and that it is NP-hard on planar multigraphs. We extend our results to the related Cat Herding problem, proving its PSPACE-completeness.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Graphs, Evasion and Pursuit Games, PSPACE-completeness, Quantified SAT, Canadian Traveler Problem, Cat Herding Problem

Digital Object Identifier 10.4230/LIPIcs.FUN.2026.7

Related Version The full version including a variant of the game and the NP-completeness of the computation of a binary escape tree is hosted on ArXiv.
Full Version: `https://arxiv.org/abs/2601.13841` [6]

Funding This work was supported by the project ANR SPAWN ANR-25-CE48-1750.

Acknowledgements We thank Claire Hilaire and Gaëtan Berthe for the discussions and the different ideas they brought to this work.

1 Introduction

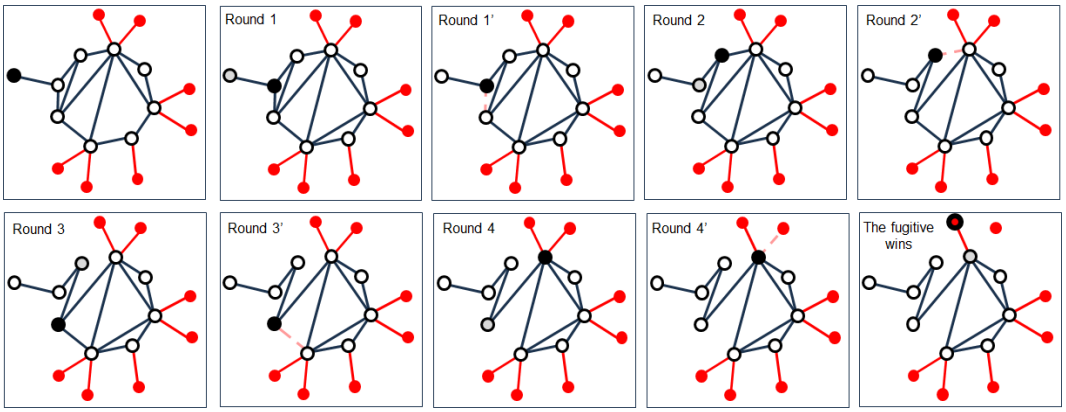
Nemesis escape game

Over the last twenty years, Escape Games have become a highly popular form of entertainment in many countries. These are collaborative games in which a participant or a team attempts to exit a room by solving a sequence of riddles. In this paper, we propose a similar framework, but set in a graph environment. An individual, whom we call the *fugitive*, seeks to escape from a graph. Some vertices are labeled as *exits*, and the objective of the fugitive is to reach any of them.

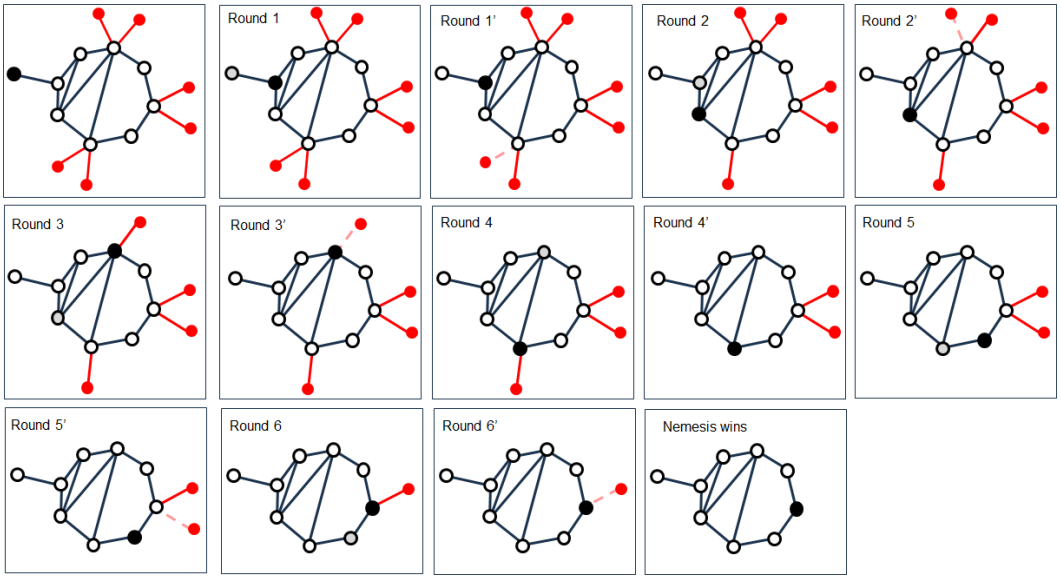
At each round of the game, the fugitive moves from his current vertex to an adjacent one by traversing an edge. If, at any round, he manages to reach an exit, he wins. In a static graph, a simple breadth-first search (BFS) computes the shortest path from the fugitive's starting vertex to an exit, giving the exact minimum number of rounds required to escape.

[1] Corresponding author

The game comes from the introduction of an adversary of the fugitive who has the magic power to alter the graph. We now define the new two player's escape game in a graph that we call *Nemesis*[2]. *The Nemesis*[3] is the name of the adversary of the fugitive. The Nemesis alters the structure of the graph in a very restricted and natural way. At each round, after the fugitive's move, the Nemesis permanently removes exactly one edge of the graph. The deleted edge can be chosen anywhere in the graph. The Nemesis wins if she succeeds in cutting all paths from the fugitive's position to every exit, thereby confining him in the graph forever. The game ends either when the fugitive reaches an exit or when the Nemesis traps him in a connected component without any exit (two plays are represented in Fig. 1).



(a) In the above game, the fugitive wins.



(b) In the above game, the Nemesis wins.

Figure 1 Two games of Nemesis. Exits are red vertices. At each round, the fugitive whose position is represented by the black vertex moves from one edge while the Nemesis removes one edge.

[2] In Greek mythology, Nemesis is a goddess of righteous anger and divine punishment.
[3] We add the article *the* into *the Nemesis* for referring to the fugitive adversary while NEMESIS without article is used as name of the game.

Despite the simplicity of this setting, and although many pursuit-and-evasion or blocking games on graphs have been studied, Nemesis is a natural escape game which, to the best of our knowledge, has never been investigated before. However, Nemesis is closely related to several well-studied problems.

Related Games

There is a rich literature on pursuit games on graphs, in particular on the classical cops and robber problem [26, 1, 22, 17, 9, 12], which has been proved to be PSPACE-complete in [23]. There are two major differences between cops and robber games and Nemesis. First, in the cops and robber problem, there are no exit vertices. Second, the pursuers move on the graph, whereas in our setting the Nemesis instead deletes edges.

Nemesis is more closely related to the guard problem, which is also PSPACE-complete [16, 18]. In the guard problem, the input is a graph with a subset of vertices defining a protected area, defended by a fixed number of guards. The goal of the guards is to prevent an intruder from reaching this protected area. If one replaces the intruder by a fugitive and the protected area by exits, the framework is very similar to Nemesis. However, the essential difference is that the guards move on the vertices of the graph, while in our simple game the Nemesis irrevocably deletes edges.

Another related problem is John Conway's Angel and Devil problem, in which an angel moves on an infinite two-dimensional grid while the devil attempts to trap it by progressively removing (or "eating") vertices [7, 13]. While it has been shown that the angel wins if they can move at distance at least 2 each round [19, 24, 28], several variants have been studied depending on the respective powers of the angel and the devil [11] or in other graphs such as 3D grids [20, 8]. In the same spirit of trapping a player moving in a graph, the Cat Herding problem was recently introduced in [2] and investigated in [3, 4]. Instead of catching an angel in an infinite grid, the herder tries to trap a cat maliciously wandering in a finite graph. This problem looks like Nemesis in that the herder deletes edges one at a time while the cat moves from one edge at each round. The significant difference is that the cat has no exit to escape to, which could be unsurprising for a kitten, but unfortunate for a rooftop cat. The absence of exit vertices makes Cat Herding fundamentally different from Nemesis. In the Cat Herding problem, the goal of the cat is to remain free for as long as possible, while the herder attempts to trap it as quickly as possible. A by-product of the results about Nemesis is the PSPACE-completeness of the Cat Herding problem. We also note the existence of the 2012 commercial board game Nowhere to Go [5], in which two players try to trap their opponent by removing edges in a finite hexagonal grid.

Nemesis may also be viewed as a connectivity game, close to Hex or Generalized Geography, since one player (the fugitive) attempts to build a connection between its initial vertex and an exit while avoiding the edges deleted by the Nemesis. The difference is that the fugitive must follow a single path, restricting his choices more than in unstructured vertex-selection games. Many connectivity games are known to be PSPACE-complete, such as Hex on general [15] and on planar graphs [27], or Havannah, Twixt, and Slither [10].

A final closely related problem is the Canadian Traveler Problem, introduced by Papadimitriou and Yannakakis in 1991 [25]. In this online problem, a traveler wishes to go from its initial position s to one target vertex t on a known graph, but each edge may be blocked by snow, and the traveler only learns whether an edge is passable upon reaching one of its endpoints. The task of designing a good strategy can be modeled as a game between the traveler and the storm. The traveler moves in the graph exactly as the fugitive

in our escape game, while the snowstorm plays the role of the adversary who determines the availability of edges adjacent to the traveler's current vertex. The traveler aims to keep the total length of his walk within a given competitive ratio compared to the optimal available path. Papadimitriou and Yannakakis proved that the Canadian Traveler Problem is PSPACE-complete [25]. The first difference with Nemesis is in the objective of the Canadian traveler which is to maintain an effective competitive ratio when it reaches the exit. In contrast, the goal of the fugitive is simply to reach to an exit: the notion of competitive ratio is thus irrelevant in Nemesis. Another difference is in the power of the adversary: while the Nemesis can remove an edge anywhere on the graph, the storm only impacts edges incident with the traveler's position when he first reaches it. A last difference is that the snow can remove many edges in one round while the Nemesis can only remove them one by one.

We also introduced a variant of Nemesis called Blizzard as nod of the Canadian Traveler Problem. The difference is that in Blizzard the Nemesis can only delete an edge which is incident with the fugitive's current vertex while in Nemesis, any edge can be deleted. This restriction on the power of the Nemesis makes Blizzard significantly more tractable but due to space limitation, this variant is only discussed in the full version of the paper [6].

Results

Nemesis is a zero-sum game. Our main question is to determine, for a given starting position of the fugitive, which player has a winning strategy. By solving the game, we mean deciding which of the two players can force a win from the given starting vertex.

An overview of our results on Nemesis is given in Table 1. They fall into two classes: polynomial-time solvable cases, and hardness results obtained by reductions from computationally hard problems.

A key concept throughout the paper is that of a *binary escape tree*. A binary escape tree is a full binary tree in which the root has degree 2, all internal vertices have degree 3, and all leaves are exits. The presence of a binary escape tree in the neighborhood of the fugitive's starting position guarantees a winning strategy for the fugitive. However, this condition is not universal: there exist instances in which the fugitive has a winning strategy despite the absence of any nearby binary escape tree. Another negative result relative to binary escape trees comes from the property that searching for a binary escape tree is itself computationally hard (this statement is presented in the full version of the paper [6]).

We however show that, for trees and for graphs of maximum degree 3 (without multiedges), the existence of a binary escape tree is both necessary and sufficient to certify that the fugitive wins. Moreover, binary escape trees can be detected in linear time in both graph classes. As a consequence, Nemesis can be solved efficiently on these graphs.

▶ **Theorem 1.** *For trees and graphs of maximum degree at most* 3*,* Nemesis *can be solved in linear time.*

We now present the main hardness contributions of the paper, which consist of two complexity results for Nemesis. We first show that the game becomes computationally hard on planar multigraphs.

▶ **Theorem 2.** *For planar multigraphs,* Nemesis *is NP-hard.*

However, as is often the case for two-player games, Nemesis is not known to belong to NP, even when restricted to planar multigraphs. Therefore, NP-completeness is not the appropriate complexity notion for this problem. In fact, the exact complexity of Nemesis on planar graphs (even without multiedges) remains open and appears to be a challenging question.

Table 1 A summary of the complexity results for NEMESIS.

	Trees	$\Delta \leq 3$	Planar graphs	Arbitrary graphs
Simple graphs	Linear (Th. 1)	Linear (Th. 1)	?	PSPACE-complete (Th. 3 + Th. 7)
Multi-graphs	?	?	NP-hard (Th. 2)	PSPACE-complete (Th. 3)

The proof of Theorem 2 relies on a reduction from a planar variant of SAT. Unfortunately, we are currently unable to reduce the seminal PSPACE-complete problem QUANTIFIED SAT (QSAT) to NEMESIS in planar multigraphs, and thus cannot establish PSPACE-completeness in this restricted setting.

Nevertheless, we provide a reduction from QSAT to NEMESIS in arbitrary multigraphs. The additional gadget required to simulate quantifier alternation is inherently non-planar. As a consequence, we obtain PSPACE-completeness only for arbitrary multigraphs and, after additional arguments, for arbitrary simple graphs.

▶ **Theorem 3.** *NEMESIS is PSPACE-complete.*

As a corollary of Theorem 3, we obtain the complexity of the CAT HERDING problem.

▶ **Corollary 4.** *The CAT HERDING problem is PSPACE-complete.*

The paper is organized as follows. In Section 2, we formally define the games, and give examples and basic observations. Section 3 is devoted to graph classes for which NEMESIS can be solved in polynomial time. Finally, Section 4 contains our hardness results.

2 Problem Statement and First Observations

We begin with a formal definition of NEMESIS. We also present preliminary remarks on straightforward cases or dealing with the number of exits.

2.1 The Game

We define the game NEMESIS on either a graph $G = (V, E)$ or a multigraph (where E is a multiset of edges). A subset of vertices is designated as *exits*, while the remaining vertices are called *regular*.

NEMESIS is a two-player, zero-sum game. Player 1 is the *fugitive*, and Player 2 is the *Nemesis*. The game is asymmetric: the fugitive moves along the graph, whereas the Nemesis removes edges. Their objectives are also different. The fugitive wins if he reaches an exit vertex, while the Nemesis wins if she isolates the fugitive in a connected component that contains no exit.

An instance of the game consists of a graph or multigraph G, a designated set of exits, and a starting vertex s indicating the initial position of the fugitive. We may assume without loss of generality that the exits form an independent set, since edges between exits play no role in the game.

The game proceeds in rounds. At each round, the fugitive first moves from his current vertex x to an adjacent vertex y by traversing an edge (x, y) still present in E. Then, the Nemesis removes one edge from E. In the case of multigraphs, if an edge e has multiplicity

$r \geq 1$, removing a copy of e decreases its multiplicity to $r - 1$. If $r - 1 = 0$, the edge disappears from the graph. Otherwise, it remains usable by the fugitive. In the language of games, edges can be seen as having a number of *health points* (HP).

Several variants of the game can naturally be defined. For instance, the Nemesis could remove an edge before the fugitive makes the first move, or both players could act simultaneously without knowing the adversary's action. Regardless of the precise timing rules, these variants are closely related. Throughout the paper, we adopt the following convention: the fugitive moves first, thereby triggering the wrath of the gods and setting the Nemesis into action.

Since one edge is removed at each round, the number of rounds is bounded by the number of edges, and the game is finite. NEMESIS satisfies the following two basic properties.

▶ Remark 5. If the fugitive has a winning strategy, then he has a winning strategy that never visits the same vertex twice. Indeed, suppose a winning strategy contains a loop between rounds i and j. Since fewer edges remain after round j than after round i, the fugitive can skip the loop without weakening his position. In other words, adding the rule that the fugitive cannot come back to a previously visited vertex does not change the outcome of the game. This additional rule is taken into account in the proofs. It follows that if the fugitive wins, he can do so in at most $|V|$ rounds. As a consequence, in multigraphs, any edge of multiplicity greater than $|V|$ cannot be fully removed by the Nemesis. Such edges behave as if they had infinite multiplicity and are called *unbreakable*.

▶ Remark 6. In simple graphs, many instances are trivial. If no vertex is adjacent to two exits, then the fugitive always loses, except in the trivial cases where he starts at an exit or at a neighbor of an exit. Indeed, whenever the fugitive reaches a vertex at distance 1 from an exit, the Nemesis can remove the unique edge leading to the exit, making it unreachable. Therefore, in simple graphs, non-trivial instances of NEMESIS must contain vertices adjacent to at least two exits.

2.2 Exercises

As a warm-up, we present several small instances of the game. Who wins in the instances depicted in Fig. 2? Solutions are provided in the footnote.[4]

2.3 About the Number of Exits

We now investigate how the number of exits influences the complexity of NEMESIS. As a first observation, there is a straightforward reduction from NEMESIS to NEMESIS WITH ONE EXIT in the case of multigraphs. Indeed, all exits can be merged into a single exit vertex t by replacing, for every vertex v, the edges from v to exits by a single edge from v to t whose multiplicity equals the sum of the original multiplicities.

In a later section, we prove that NEMESIS is PSPACE-complete on multigraphs (Theorem 3). The reduction above immediately implies that even NEMESIS WITH ONE EXIT is PSPACE-complete in this setting. Consequently, assuming P $\neq$ PSPACE, there cannot exist any tractable algorithm (neither FPT nor XP) parameterized by the number of exits.

The situation is different for simple graphs. With only one exit, NEMESIS becomes trivial. Indeed, there are two cases. If the starting vertex s is adjacent to the unique exit t, the fugitive wins immediately. Otherwise, as noticed in Remark 6, the Nemesis has a simple winning strategy: whenever the fugitive reaches a neighbor u of t, she removes the edge (u, t).

[4] a–Nemesis wins (N), b–Fugitive wins (F), c–N, d–N, e–F, f–N, g–F, h–F.

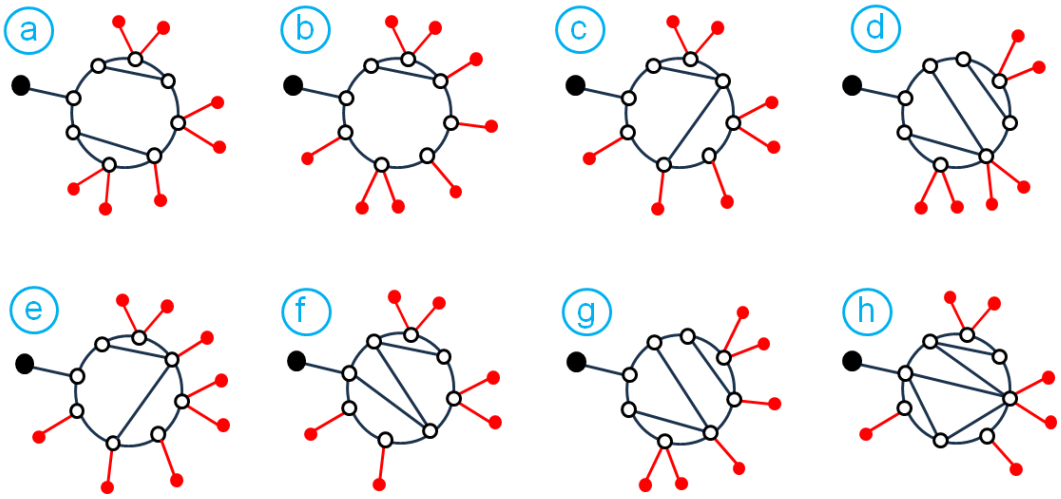


Figure 2 Eight instances of Nemesis. Can the fugitive, starting from the black vertex, escape?

It is therefore natural to ask whether this observation extends to a fixed number of exits. As we now show, the answer is negative: even in simple graphs, restricting the number of exits to two already captures the full complexity of the game.

▶ **Theorem 7.** *For simple graphs, there is a polynomial-time reduction from* Nemesis *to* Nemesis with two exits.

Proof. Let (G, s, X) be an instance of Nemesis, where $G = (V, E)$ is a simple graph, $s \in V$ is the starting vertex, and $X \subsetneq V$ is the set of exits, with $s \notin X$. We construct an instance $(G', s, \{t_1, t_2\})$ of Nemesis with two exits as follows.

- The vertex set of G' is $V' = V \cup Y \cup \{t_1, t_2\}$, where $Y = \{y_1, \dots, y_{n+1}\}$ is a set of $n+1$ new vertices.
- The edge set of G' is $E' = E \cup (X \times Y) \cup (Y \times \{t_1, t_2\})$, that is, every vertex of X is connected to every vertex of Y, and every vertex of Y is connected to both exits t_1 and t_2.
- The starting position remains s, and the exits are t_1 and t_2.

The graph G' has $2n+3$ vertices and at most $m + (n-1)(n+1) + 2(n+1)$ edges.

We now show that the two instances are equivalent. First, assume that the fugitive has a winning strategy in (G, s, X). By Remark 5, he reaches some exit $x \in X$ within at most n rounds. Up to this point, the Nemesis has removed at most n edges. Since $|Y| = n+1$, there exists a vertex $y \in Y$ such that no edge incident to y has been removed. In particular, y is still adjacent to x, t_1, and t_2. The fugitive can therefore move from x to y. As y has two exit neighbors, the Nemesis can block at most one of them, and the fugitive wins.

Conversely, if the Nemesis has a winning strategy in (G, s, X), the same strategy applied in $(G', s, \{t_1, t_2\})$ allows her to prevent the fugitive to reach the vertices of X and thus to reach any of the two exits $\{t_1, t_2\}$. ◀

Theorem 7 shows that, in simple graphs, two exits suffice to capture the computational complexity of Nemesis. Note however that this reduction does not, in general, preserve planarity.

3 Polynomial Time Games

For some classes of instances, the winner of the game can be determined efficiently by using classical graph structures. We now investigate several such cases.

3.1 Trees and Graphs of Maximum Degree 3

Before addressing the simplest graph classes, we introduce a graph simplification procedure.

Graph Simplification

Given a graph with exits and a starting vertex s, we introduce a first reduction performed in two phases (Fig. 3). The first phase consists in duplicating the exits so that each exit has degree 1. See for instance the transition from a) to b) in Fig. 3. This transformation simplifies the structure by avoiding spurious cycles passing through exits, and it preserves the outcome of the game.

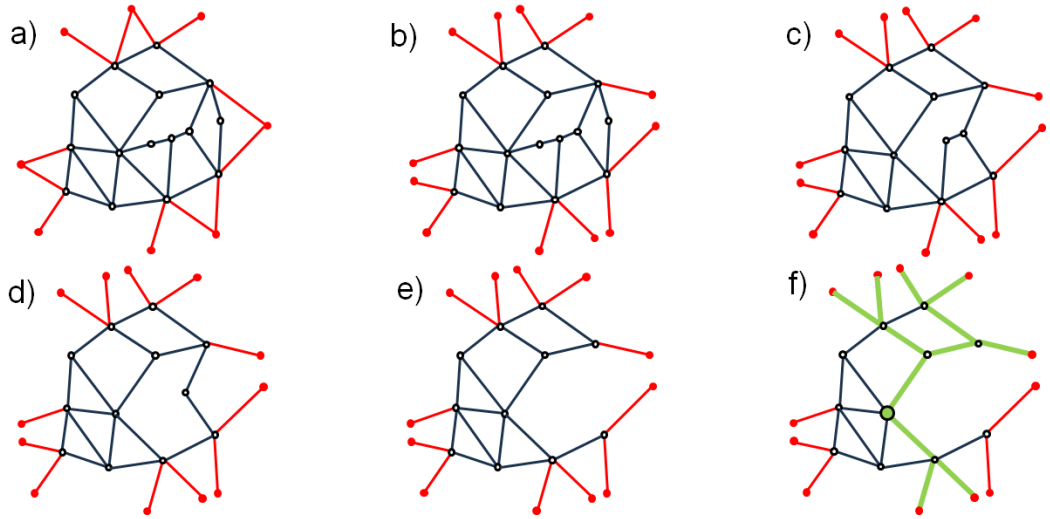


Figure 3 Simplification of an input graph. From a) to b), exits are duplicated. From b) to e), regular vertices of degree 1 or 2 are pruned. In f), a binary escape tree.

The second transformation consists in removing from the graph all vertices such that, if the fugitive ever reaches one of them, he is inevitably doomed to lose. More precisely, we iteratively prune all regular vertices of degree 1 or 2 (except the starting vertex s), since such vertices act as traps for the fugitive. As these vertices never play any role in a winning strategy, this pruning phase does not change the outcome of the game.

This pruning can be implemented using a Breadth-first Search (BFS) initialized with the initial set of vertices to be removed. We also discard any connected component that does not contain s. Overall, this preprocessing runs in linear time.

From now on, we therefore assume that all input graphs have been previously simplified. Note that this simplification never increases the degree of any vertex.

Binary Escape Trees

A key structure guaranteeing a win for the fugitive is the notion of a *binary escape tree*.

▶ **Definition 8.** *A* binary escape tree *of a graph G (given with a set of exits) is a rooted full binary tree (i.e., a rooted tree in which every internal node has exactly two children) whose vertices and edges belong to G, and whose leaves are exits.*

Binary escape trees provide the fugitive with a simple winning strategy.

▶ **Lemma 9.** *Given an instance of* Nemesis *with graph $G = (V, E)$, a set of exits, and a starting vertex s, if s or one of its neighbors in G is the root of a binary escape tree, then the fugitive has a winning strategy.*

Proof. The winning strategy is straightforward. If the root of the binary escape tree is a neighbor of s, the fugitive moves to this root in the first round. Otherwise, if the tree is rooted at s, he moves to one of its children.

At each round, the Nemesis can delete at most one edge. Consequently, from the fugitive's current position, at least one of the two subtrees of the current root remains intact. At the next round, the fugitive moves to the root of this unaltered subtree. Repeating this strategy ensures that after each of his moves, the fugitive occupies the root of a binary escape tree in the current graph. Since the height of the binary escape tree strictly decreases at each round, the fugitive eventually reaches an exit and wins the game. ◀

We call the *tree condition* the property that at least one vertex at distance at most 1 from the fugitive's current position is the root of a binary escape tree. Lemma 9 can then be restated as follows: if the tree condition is satisfied, the fugitive wins.

It is natural to ask whether this condition is not only sufficient but also necessary. If this were the case, it would yield a simple characterization of the starting positions from which the fugitive has a winning strategy. However, this is not true in general. Indeed, there exist instances in which the fugitive wins even though the tree condition is not satisfied. A counter-example occures with square grids.

Square Grids

Square grids deserve special attention, as readers familiar with Conway's Angel game may recognize in Nemesis a variant of the demon's strategy [7, 13]. The set of regular vertices of our instance forms an $m \times n$ square grid. Each corner vertex is connected to two exits, while each other boundary vertex is connected to exactly one exit. The instance and the corresponding strategies are illustrated in Fig. 4.

It is easy to see that if the fugitive starts at a vertex at distance at least 6 from the closest exit, then the Nemesis has a winning strategy by cutting the corner exits. Conversely, if the distance from the starting vertex to the boundary vertices is at most 5, the fugitive has a winning strategy by reaching the boundary and then moving around the grid (we do not formally prove this claim).

An interesting feature of the grid is that when the starting vertex s is at distance 5 from the exits, the fugitive has a winning strategy, even though s does not satisfy the tree condition. We now prove this claim by contradiction.

Assume that a binary escape tree is rooted at a vertex at distance at most 1 from s. Then the distance from this root to the exits is at least 4. This root has two children at distance at least 3 from the exits, each being the root of an independent binary escape tree. Each of these two nodes in turn has two children at distance at least 2 from the exits, again serving as roots of independent binary escape trees. At the next level, we therefore obtain 8 nodes at distance at least 1 from the exits, each being the root of a binary escape tree.

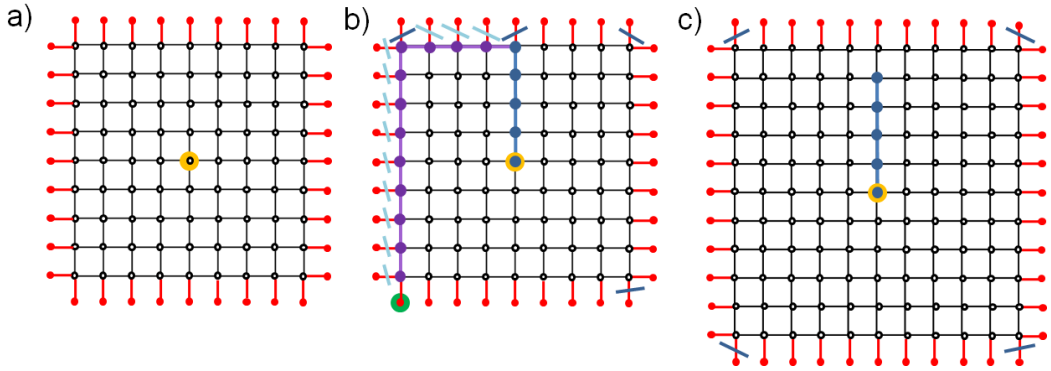


Figure 4 Nemesis on a square grid. In a), the input graph with the starting vertex in orange and exits in red. In b), with a starting vertex at distance 5 from the closest exit, the fugitive wins by moving around the grid. In c), with a starting vertex at distance 6 from the closest exit, the Nemesis wins by cutting one exit at each corner.

However, any binary escape tree that is not rooted at an exit must contain at least one vertex adjacent to two exits. Hence, this construction implies the existence of 8 distinct vertices each adjacent to two exits. Since the grid contains only 4 such vertices, we obtain a contradiction. Therefore, s does not satisfy the tree condition.

Tree Condition is Necessary and Sufficient for Trees and Graphs of Maximum Degree 3

For arbitrary graphs, the tree condition is sufficient but not necessary to guarantee a win for the fugitive. We now show that when the graph G is either a tree or has maximum degree at most 3 (after graph simplification), this condition is also necessary.

▶ **Lemma 10.** *Given a* Nemesis *instance with a set of exits and a graph $G = (V, E)$ that is either a tree or has maximum degree at most 3 after graph simplification, the fugitive has a winning strategy from the starting position s if and only if s or one of its neighbors in G is the root of a binary escape tree.*

Proof. Note that the simplified graph of a tree or a graph of max degree 3 is itself a tree or a graph of max degree 3. The condition that the graph has been previously simplified does not reduce the scope of Lemma 10 but it is necessary to avoid artificial cycles which might pass through the exits in the initial graph before the duplication step.

One direction of the equivalence is given by Lemma 9. We prove the converse, namely that if the tree condition is not satisfied, then Nemesis has a winning strategy.

We proceed by induction on the size $n = |V|$ of the graph G.

We start with trees. Our induction hypothesis is that for any tree with at most n vertices, the fugitive wins if and only if one of the vertices at distance at most 1 from his current position is the root of a binary escape tree.

Consider now a tree with $n + 1$ vertices, and assume that the fugitive has a winning strategy from s. By Remark 5, we may assume that this strategy never visits the same vertex twice. At his first move, the fugitive moves from s to a vertex v and never returns to s. Thus, the fugitive has a winning strategy in the subtree of G rooted at v, whose size is at most n.

By the induction hypothesis, either v itself or one of its neighbors is the root of a binary escape tree in this subtree. If v is such a root, then the tree condition holds at s. Otherwise, at least one child of v has this property.

Suppose that exactly one child v' of v is the root of a binary escape tree in this subtree. Then, immediately after the fugitive moves from s to v, Nemesis can delete the edge (v, v'), thereby destroying the tree condition at v. By the induction hypothesis, the fugitive then has no winning strategy, contradicting our assumption. Hence, v must have at least two children that are roots of binary escape trees.

Since G is a tree, these two subtrees are vertex-disjoint. It follows that v itself is the root of a binary escape tree. Thus, the tree condition is satisfied at s through its neighbor v.

We now consider graphs of maximum degree 3. The proof follows the same inductive framework, but requires an additional final argument.

The induction hypothesis is that for any graph of maximum degree at most 3 with at most n vertices, the fugitive wins if and only if the tree condition is satisfied. Let G be such a graph with $n + 1$ vertices, and let s be a winning starting position for the fugitive.

As before, we may assume that the fugitive follows a winning strategy without loops. Let v be the first vertex reached by the fugitive from s according to the winning strategy. After the first round (one move by the fugitive and one deletion by Nemesis), the remaining graph has size at most n, and the fugitive is at v. By the induction hypothesis, v or one of its neighbors is the root of a binary escape tree.

If v itself is such a root, the tree condition holds and we are done. Otherwise, let v' be a neighbor of v that is the root of a binary escape tree. Since, in this case, v is regular, v has two other neighbors: s, v' and v'' (after simplification the unique vertex of degree 2 might be s).

Nemesis may delete the edge (v, v') at the first round, forcing the fugitive to move toward v''. If v'' is not the root of a binary escape tree, then by the induction hypothesis the fugitive loses, contradicting our assumption. Hence, both v' and v'' must be roots of binary escape trees.

If the two binary escape trees rooted at v' and v'' are vertex-disjoint, then v is itself the root of a binary escape tree, and the tree condition holds. Otherwise, these two trees share vertices. We now show that such a configuration contradicts the initial assumption that the fugitive has a winning strategy starting by the move (s, v).

Since the graph has maximum degree 3 and the fugitive never revisits a vertex, whenever the fugitive arrives at a vertex, one incident edge has already been used and will not be crossed again. Of the remaining two edges, Nemesis can delete one, thereby forcing the fugitive's next move. If the two binary escape trees rooted at v' and v'' are not vertex-disjoint, then their union contains a cycle C passing through v, v', and v'', in which Nemesis can trap the fugitive. Indeed, she only has to remove, at each round, the single edge escaping from cycle C. As the graph is assumed to be simplified, the cycle C does not contain exits. It ensures the win of the Nemesis and contradicts the assumption that the fugitive can escape after initially moving to v. We conclude that whenever the fugitive has a winning strategy in a tree or in a graph of maximum degree 3, the tree condition must be satisfied. ◀

The tree condition can be checked in linear time for trees using a BFS starting from s. For graphs of maximum degree 3, the condition can also be checked in linear time, although the argument is slightly more involved. The algorithm also performs a BFS from the starting vertex s and labels each visited vertex according to the first neighbor of s through which it is reached. Since $\deg(s) \leq 3$, there are at most three such branches.

If a vertex is reached from two different branches, then both branches are disabled, as they necessarily share a cycle and cannot both belong to a binary escape tree. If a vertex is reached twice from the same branch, this branch is disabled as well.

After the traversal, the vertex s is the root of a binary escape tree if and only if at least two branches remain enabled. The correctness of the algorithm follows from the fact that in graphs of maximum degree 3, any branch containing a cycle cannot be part of a binary escape tree. Conversely, due to the preliminary simplification, all the regular vertices are of degree 3. Then any branch which does not contain a cycle is a binary exit tree. Hence, the existence of two enabled branches *i.e.* without cycles implies that s is the root of a binary escape tree.

4 Hardness Results

In this section, we prove the hardness results of the paper. We first present a reduction from Planar SAT to Nemesis on planar multigraphs, proving Theorem 2 (Section 4.1). Finally, we prove with Theorem 3 the PSPACE-completeness of Nemesis on arbitrary graphs (Section 4.3) using arbitrary multigraphs as an intermediary lemma (Section 4.2). As a corollary, we also derive the PSPACE-completeness of the Cat Herding problem (Section 4.4).

4.1 Hardness of Nemesis for Planar Multigraphs

We now prove Theorem 2. Readers familiar with the reduction of QSAT to Geography [21] or to the Canadian Traveler Problem [25] may recognize the general structure of the reduction from QSAT to graph problems. These classical reductions use a main central path around which local choices encode the Boolean variable assignment of a SAT instance.

Planar Monotone Rectilinear 3SAT

We do not directly reduce QSAT but rather the NP-complete problem Planar Monotone Rectilinear 3SAT [14], which we denote by PMR3SAT. A key feature of these 3SAT instances is that each clause is either all-positive (containing only non-negated literals) or all-negative (containing only negated literals). An important property of Planar Monotone Rectilinear 3SAT is that it admits a geometric representation by non-crossing line segments.

In this representation, Boolean variables appear as a sequence of axis parallel rectangles lying on the horizontal axis. Clauses correspond to axis parallel rectangles placed in the half-plane $y < 0$ for all-negative clauses, and in $y > 0$ for all-positive clauses. The participation of a variable in a clause is encoded by a vertical segment that connects the rectangle of the variable to that of the clause. An instance of PMR3SAT is shown in the upper part of Fig. 5. We do not rely on the fact that these segments are axis-parallel.

Construction of the Nemesis instance

We present the reduction of an instance of PMR3SAT to an instance of Nemesis in a planar multigraph. Let n be the number of variables and m the number of clauses of the PMR3SAT instance. The construction can be followed in Fig. 5.

Each Boolean variable is represented by a cycle of size $4K$, where K is a constant chosen larger than the maximum number of clauses in which any given variable appears (larger than m is sufficient). Thus, the ordered list of the n variables of the PMR3SAT instance is encoded by a sequence of cycles - one per variable - each connected to the next by an intermediate vertex.

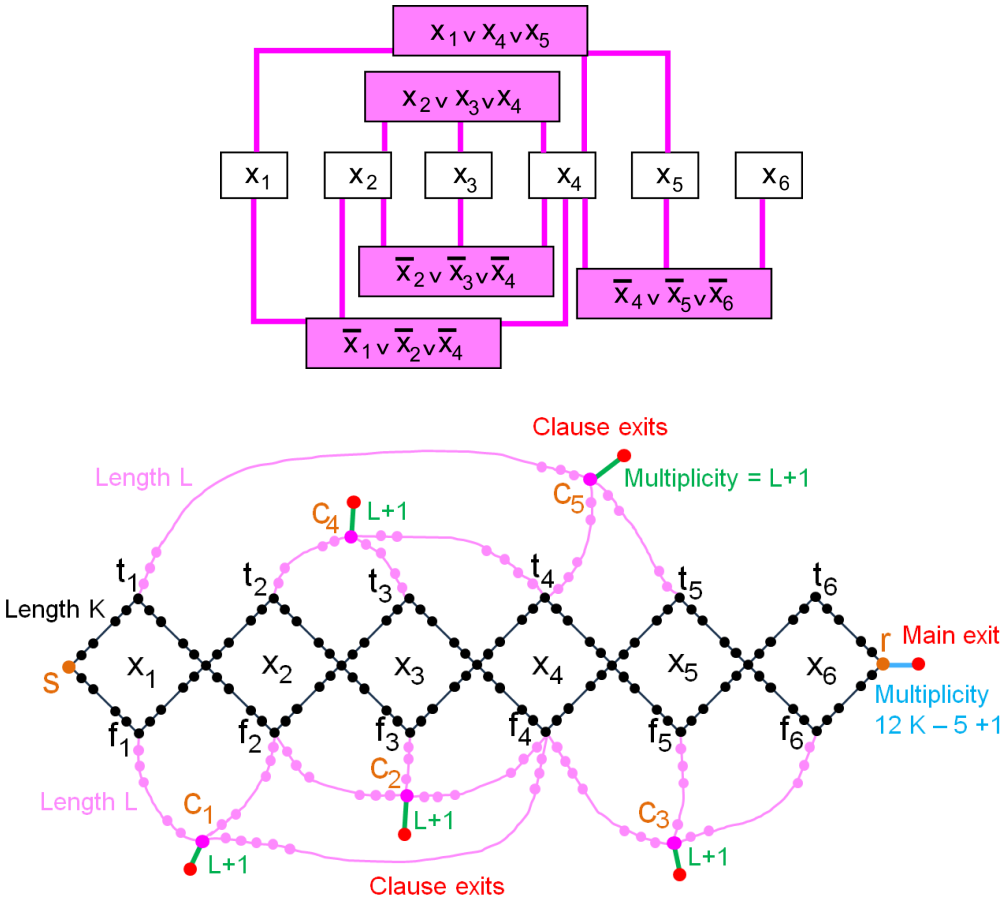


Figure 5 Reduction of a PMR3SAT instance to a Nemesis instance. We reduce the instance of PMR3SAT represented above. All the edges of the Nemesis instance are unbreakable except the edges adjacent to exits (in green). Their mulitplicity is $L + 1$ for the clause exists and it is $12K - 5 + 1$ for the main exit.

The key idea of the reduction is that, when traveling from the leftmost start vertex s to the rightmost regular vertex r, the fugitive must traverse each cycle on one side or the other, so that his path from s to r encodes an assignment of the n variables. For this purpose, we introduce intermediate vertices t_i (for true) and f_i (for false) on the two sides of each cycle. For instance, a path from s to r passing through vertices t_1, f_2, t_3, t_4, f_5, f_6 corresponds to the assignment where x_1, x_3 and x_4 are true, and x_2, x_5 and x_6 are false.

Observe that, regardless of the choice of side for each variable, all s-r paths passing through the variable cycles have length $2nK$ (in the figure, with $n = 6$, this length is $12K$).

Since this central path is crucial to the reduction, we ensure that the Nemesis cannot destroy its structure: all its edges are given multiplicities that make them unbreakable. The same applies to the additional edges introduced below. Almost all edges are therefore unbreakable. The only edges whose multiplicities allow the Nemesis to delete them are those incident with exits. The main exit is connected to the rightmost regular vertex r, and we set the multiplicity of this edge to $2nK - m + 1$. The remaining exits correspond to the clauses.

Each clause is represented by a vertex c_i connected to an exit by an edge of multiplicity $L + 1$, where L is chosen larger than $2nK$. Each clause vertex c_h is also connected to the cycles of its variables.

For an all-positive clause $x_i \vee x_j \vee x_k$, the vertex c_h is linked to t_i, t_j, and t_k by unbreakable paths of length L.

For an all-negative clause $\overline{x_i} \vee \overline{x_j} \vee \overline{x_k}$, the vertex c_h is linked to f_i, f_j, and f_k by unbreakable paths of length L.

The choice of L, which is larger than the distance from s to r, guarantees that these clause paths cannot create shortcuts between s and r.

How does it work?

We first consider the direct paths from s to r. For each variable x_i, the path goes through either t_i or f_i. Assume, without loss of generality, that the fugitive arrives at t_i. From there, he may attempt to use one of the corresponding clause paths to reach an exit. When the fugitive is at t_i, his distance to c_h is L, and the initial multiplicity of the exit edge from c_h is $L+1$. Thus, if this exit edge has not been attacked by the Nemesis at least once beforehand, the fugitive can reach the clause exit, since the Nemesis no longer has enough time to remove that edge.

This forces the Nemesis to anticipate: before the fugitive reaches t_i, he must traverse a path of length K. During these K rounds, the Nemesis must attack the exit edges of all clauses connected to t_i. Reducing the multiplicity of each such exit edge by one is sufficient to make the corresponding clause exit harmless. This reduction is unavoidable, because otherwise the fugitive could escape through that exit. Consequently, for each clause reachable from the fugitive's path, the Nemesis must spend one round reducing an exit edge's multiplicity from $L+1$ to L.

Equivalence between the PMR3SAT and the corresponding Nemesis instances

We now show that when the PMR3SAT instance is satisfiable, the fugitive has a winning strategy by following the path corresponding to a satisfying assignment. Along this path, the vertices t_i and f_i visited by the fugitive are connected to all m clauses. Hence, the Nemesis must spend m rounds preventing escape through the clause exits. This leaves only $2nK - m$ rounds during the fugitive's traversal from s to r (a path of length $2nK$) to destroy the main exit edge. Since its multiplicity is set to $2nK - m + 1$, that edge still has multiplicity 1 at the beginning of round $2nK - m + 1$ (after the fugitive reaches r), allowing the fugitive to exit.

We now show that if the PMR3SAT instance is not satisfiable, then the fugitive cannot escape. In this case, the fugitive's path from s to r visits at most $m - 1$ clauses, so the Nemesis has enough free rounds to remove the main exit edge.

Could the fugitive take a better path? Using a clause path is not advantageous: if he travels from t_i or f_i to c_h, then at every round the Nemesis can reduce the multiplicity of that clause's exit edge. The Nemesis has exactly enough time to finish removing the exit edge before the fugitive can traverse it. The fugitive must then return to the central path, during which the Nemesis has ample time to destroy the main exit and win.

Another possibility would be to backtrack within the variable cycles, but this would require the fugitive to visit some vertex twice. This cannot help: if the fugitive has a winning strategy at all, then he has one without loops (Remark 5).

Thus, regardless of the path chosen by the fugitive, he cannot reach an exit.

Let us check now that the reduction takes only a polynomial time. The number of vertices involved in the central path is lower than $4nK$ with $K \leq m$. The clause paths require $3(L+1)m$ vertices with L of the order of $2nK$. Then the total number of vertices is $O(m^2n^2)$. The Nemesis instance encoding the initial PMR3SAT intance is thus computed in polynomial time.

This completes the NP-hardness proof, showing Theorem 2.

4.2 PSPACE-completeness of Nemesis for Arbitrary Multigraphs

We now prove an intermediate result toward Theorem 3, namely the PSPACE-completeness of Nemesis for multigraphs.

The QSAT game between the Universal and Existential Players

To establish the PSPACE-hardness of Nemesis, we need to move from the previous reduction from 3SAT (or SAT in general) to a reduction from Quantified SAT (QSAT). This amounts to going from a formula φ with only existential quantifiers,

$$\exists x_1, \exists x_2, ..., \exists x_i, ..., \exists x_n, \varphi\left((x_i)_{1\leq i\leq n}\right)$$

to a formula with alternating existential and universal quantifiers,

$$\exists x_1, \forall x_2, ..., \exists x_{2i-1}, \forall x_{2i}, ..., \forall x_{2n}, \varphi\left((x_i)_{1\leq i\leq 2n}\right).$$

Variables quantified existentially are called *existential*, while the others are called *universal*.

The satisfiability of such a quantified formula can be viewed as a game between an *existential player*, who assigns values to the existential variables with the objective of satisfying the formula, and a *universal adversary*, who assigns values to the universal variables with the objective of preventing satisfaction. It is this game that we reduce to Nemesis in order to prove PSPACE-hardness.

The Electric Gadget and its Analysis

We now consider the initial reduction of SAT presented in Section 4.1 and explain how to update it in order to reduce the QSAT game. The main idea is to give the Nemesis control on the fugitive's path over every universal variable, thereby emulating the power of the universal player. To explain the construction, we use an electrical analogy.

A first idea is to introduce, on each path corresponding to a universal variable (the $true$ path and the $false$ path), an edge of multiplicity 1 that the Nemesis can use as a fuse. However, this naive approach potentially gives the Nemesis the ability to destroy both parallel fuses, which would allow her to win regardless of the fugitive's choices. Thus, while this idea provides useful intuition, it grants too much power to the Nemesis and must be refined. We therefore introduce a fuse on each path of the universal variables and complement it with two derivation paths. This construction yields the so-called *electric gadget*, illustrated in Fig. 6.

Let us now describe the gadget in its full details. The gadget consists of a fuse (an edge of multiplicity one) placed on each branch immediately after the first edge of the paths corresponding to a universal variable x_i. The two fuses connect the vertices u_i and v_i on the $true$ path, and vertices $\overline{u_i}$ and $\overline{v_i}$ on the $false$ path. We complete the electric gadget with two additional paths of length 2: one from $\overline{u_i}$ to v_i, and the other from u_i to $\overline{v_i}$. All edges on these two paths are unbreakable.

We now explain why this gadget gives the Nemesis control over the universal variable x_i. We first recall that, in the initial reduction from SAT to Nemesis, the race between the fugitive reaching the main exit and the Nemesis destroying the main exit edge is tight: if the fugitive loses even one single round, he is defeated. We return to the electric gadget and assume, without loss of generality, that the Nemesis has chosen the value of variable x_i to be true, meaning that she intends to forbid the fugitive from passing through f_i. When the fugitive reaches the cycle corresponding to the variable x_i, his first move must go either to u_i or to $\overline{u_i}$. We analyze both cases.

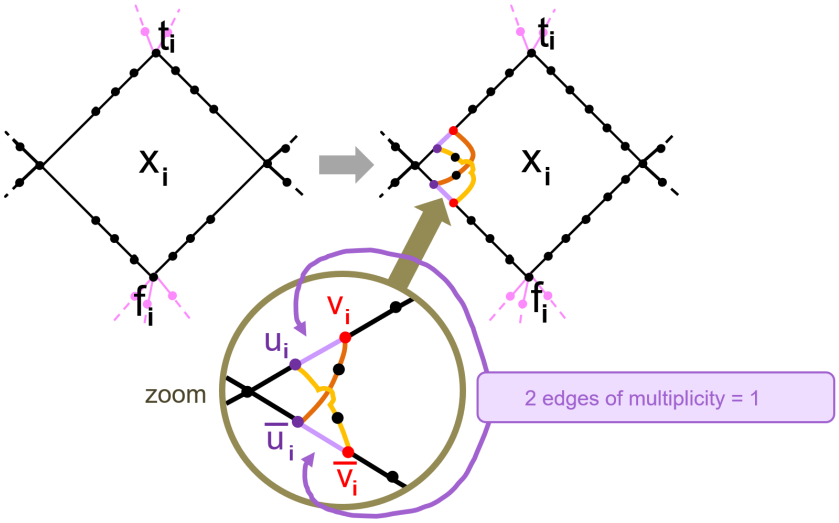


Figure 6 The Electric gadget to simulate a universal quantifier. On the left, the initial variable gadget used to reduce 3SAT. On the right, its update which allows the Nemesis to control the path followed by the fugitive.

- The fugitive moves to u_i. In this case, the Nemesis does not destroy the fuse u_iv_i. The fugitive then has two possible moves. If he goes to v_i, he effectively sets $x_i = true$, as expected by the Nemesis. If, on the contrary, he moves toward $\overline{v_i}$, he reaches $\overline{v_i}$ one round later than he would have by taking the direct path through $\overline{u_i}$. Since arriving at the main exit on time is a tight condition, this detour necessarily leads to defeat. Returning backwards would also result in defeat. Hence, the fugitive has no viable option other than proceeding toward t_i, as dictated by the Nemesis.
- The fugitive moves to $\overline{u_i}$. Since the Nemesis has decided that the fugitive must go through t_i rather than f_i, she immediately deletes the fuse between $\overline{u_i}$ and $\overline{v_i}$. The fugitive cannot turn back, as doing so would again condemn him. His only remaining option is to move to v_i, and thus toward t_i, as imposed by the Nemesis. This detour costs the fugitive one round, but the Nemesis has also spent one round deleting the fuse. These two rounds therefore cancel out, and, in terms of timing, the situation is equivalent to that in the original SAT reduction.

Another possible scenario is that the Nemesis deletes one or both fuses associated with variable x_i before the fugitive reaches u_i or $\overline{u_i}$. We now show that she cannot take advantage from such a strategy. By spending a round to delete at least one fuse prematurely, the Nemesis allows the fugitive to use either one of the unbreakable detours $u_i, \overline{v_i}, f_i$ or $\overline{u_i}, v_i, t_i$ without being condemned to lose. This gives the fugitive an advantage, making such a strategy suboptimal for the Nemesis.

This analysis shows that the electric gadget allows the Nemesis to choose whether the fugitive passes through t_i or f_i, a choice that directly corresponds to the value of the universal variable x_i. This control comes at no net cost to the Nemesis. Of course, she may choose not to exercise this power, but refraining from doing so does not give her additional rounds to destroy the main exit edge. Any such restraint is therefore useless. The control granted by the electric gadget over the universal variable cannot be exploited elsewhere in the game.

Proof of PSPACE hardness of Nemesis for Multigraphs

We now prove the PSPACE-hardness of Nemesis for multigraphs. The construction of the graph of Nemesis encoding a QSAT instance proceeds in two steps. First, we build the graph corresponding to the reduction from the SAT formula, as presented in Section 4.1. Second, we add the electric gadget at the entries of all universal Boolean variables.

This modification increases the size of the variable cycles by increasing the value of K by a constant factor, but it does not otherwise affect the behavior of the construction.

We explore both directions of the reduction.

($\Rightarrow$) Assume that the quantified formula is satisfiable. This means that, in the corresponding QSAT game, the existential player has a winning strategy that satisfies all m clauses. If the fugitive follows the same choices along the variable gadgets, he enforces the Nemesis to spend one round for each of the m clauses of the SAT formula in order to block the clause exits.

For universal variables, the Nemesis may force the fugitive to lose one round by means of the electric gadget; however, in each such situation, the Nemesis must also spend one round to delete a fuse. Consequently, these rounds compensate each other, and the Nemesis gains no advantage. It follows that the fugitive reaches vertex r before the Nemesis has completed the destruction of the main exit edge. At that point, this edge still has multiplicity 1, and the fugitive escapes and wins the game. In other words, the fugitive has a winning strategy.

($\Leftarrow$) Assume that the quantified formula is not satisfiable. Then, in the QSAT game, the universal player has a strategy that prevents the existential player from satisfying all clauses. By following an identical strategy, the Nemesis prevents the fugitive from being able to approach all m clause exits: at least one clause is necessarily missed. This missing clause yields one additional round that the Nemesis can devote to destroying the main exit edge. As a consequence, when the fugitive completes his path from s to r, the edge to the main exit is no longer passable. Therefore, the Nemesis necessarily wins the game. It means that the Nemesis has a winning strategy.

This completes the proof of PSPACE-hardness of Nemesis for multigraphs. To establish PSPACE-completeness, it remains to show that Nemesis belongs to PSPACE. Since Nemesis is a finite two-player game with perfect information, each game position can be encoded using polynomial space. Moreover, by Remark 5, if the fugitive has a winning strategy, then he has one that does not visit the same vertex twice. Therefore, it is sufficient to consider plays without repeated vertices, and the length of any play is bounded by the number of vertices of the graph. The winner of the game can thus be determined by a depth-first exploration of the game tree, evaluating positions recursively according to the current player. Although this procedure may require exponential time, it uses only polynomial space, which shows that Nemesis on multigraphs belongs to PSPACE.

Combining this result with the PSPACE-hardness, we conclude that Nemesis on multigraphs is PSPACE-complete.

4.3 PSPACE-completeness of Nemesis for Arbitrary Graphs

We now prove the PSPACE-completeness of Nemesis for arbitrary graphs (Theorem 3). Rather than redoing the proof from scratch, we explain how the PSPACE-completeness proof for Nemesis on multigraphs can be adapted to obtain the same result for simple graphs.

Our objective is therefore to start from a Nemesis instance obtained by reducing an instance of QSAT and to construct an equivalent Nemesis instance whose underlying structure is a graph (*i.e.*, without multiedges).

As a first step, we recall the key features of the multigraph instance produced by the QSAT reduction:

1. The construction uses only four distinct values of edge multiplicities. Edges of multiplicity 1 appear in the electric gadgets. Exit edges have multiplicities $L+1$ and $2nK-m+1$. All remaining edges are unbreakable: their multiplicity can be fixed to a value N larger than the total number of rounds of the game, so that the Nemesis has no reason to attempt deleting them. This value N is polynomial in the number of variables and clauses of the original QSAT instance.
2. A crucial property of the reduction is that the fugitive's winning path is tight. If the fugitive loses even a single round along his path from s to r, then he can no longer reach r before the Nemesis finishes destroying the main exit edge. This tightness is essential for the correctness of the reduction.

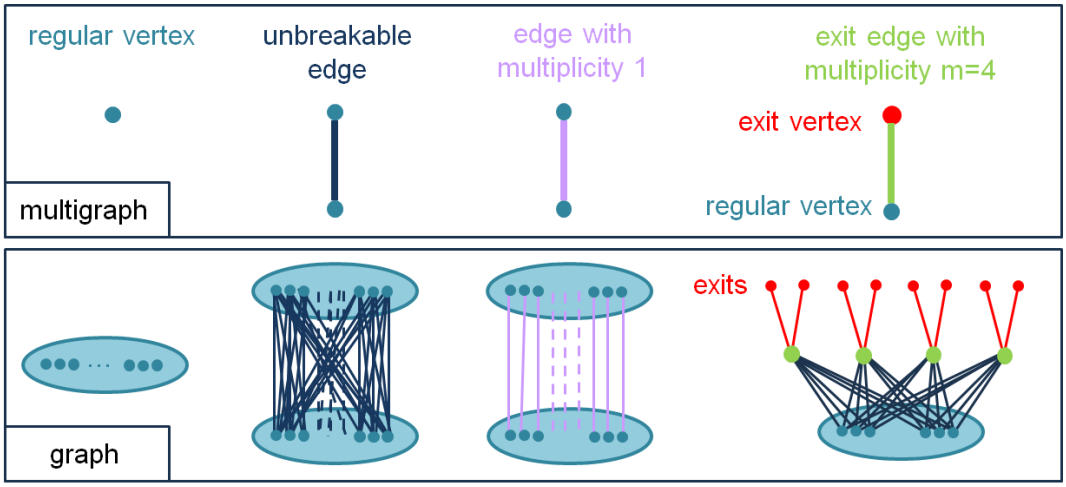


Figure 7 Gadgets for translating our multigraph instance of Nemesis into an equivalent Nemesis graph instance. Each regular vertex of the multigraph is translated into N copies of the vertex with larger than the initial number of vertices of the graph. Unbreakable edges are translated in bicliques. Edges of multiplicity 1 are translated in matchings. Exit edges of multiplicity k are encoded through k new regular vertices (the exit can be disabled by removing one exit edge of each new regular vertex).

We now explain how we translate each element of the reduced multigraph instance of Subsection 4.1 in order to build an equivalent simple graph instance (the translation gadgets are illustrated in Fig. 7).

- **Vertices.** Each vertex of the multigraph instance, except the exits, is replaced by N copies. More precisely, a vertex u in the multigraph is replaced by N vertices $u_1, \ldots, u_N$.
- **Unbreakable edges.** An unbreakable edge connecting vertices u and v in the multigraph is replaced by a complete bipartite graph between the sets $\{u_i\}_{1\leq i\leq N}$ and $\{v_j\}_{1\leq j\leq N}$. This yields N^2 edges instead of a single multiedge of multiplicity N.
  With such a construction, if the fugitive stands at some vertex u_i, the Nemesis cannot prevent him from reaching a copy of v, as blocking all N possibilities would require too many rounds. This simulates the unbreakable edges of the multigraph instance.
- **Edges of multiplicity 1.** An edge of multiplicity 1 between vertices u and v in the multigraph is replaced by a matching connecting each u_i to the corresponding v_i. This produces exactly N simple edges.

The key point is that if the fugitive stands at a vertex u_i, the Nemesis can delete the edge (u_i, v_i) in a single round. Although the fugitive could then attempt to reach another copy v_j via u_j, the Nemesis can repeat the same action. Since the timing of the reduction is tight, such repeated attempts would inevitably cause the fugitive to lose. Consequently, once the edge (u_i, v_i) is deleted while the fugitive stands at u_i, the fugitive must follow an alternative path (in the electric gadget, this corresponds to an unbreakable derivation path, now implemented via a complete bipartite graph).

Note that the Nemesis cannot efficiently delete these edges in advance. If she deletes (u_i, v_i) prematurely, the fugitive may simply use another pair (u_j, v_j). This limitation does not affect the correctness of the reduction. Indeed, in the multigraph construction, the winning strategy of the Nemesis deletes edges of multiplicity 1 only when the fugitive stands at one of their endpoints. Therefore, this simulates edges of multiplicity 1 from the multigraph instance.

- **Exit edges of arbitrary multiplicity.** To replace an exit edge of multiplicity k between a vertex u and an exit t, we introduce an *exit gadget*. We need to take into account the fact that the vertex u of the multigraph is represented in the graph by N copies u_i. We additionally introduce k intermediate vertices p_j, for $1 \leq j \leq k$. Each vertex p_j is connected to exactly two exits.

  This gadget simulates an exit edge of multiplicity k as follows. If the Nemesis has spent k rounds deleting exit edges, she can deactivate the gadget by removing one exit edge incident with each p_j. In this case, whenever the fugitive reaches some p_j, the remaining exit edge can be deleted immediately, preventing escape.

  If, on the other hand, the Nemesis has used fewer than k rounds on this gadget, then at least one vertex p_j still has both exit edges intact. When the fugitive stands at some u_i, he can move to such a vertex p_j and escape before the Nemesis can react.

  The Nemesis might attempt to delete edges between the vertices u_i and the p_j in advance. However, since the fugitive can choose among the N copies of u, he can always reach a copy from which all remaining p_j are still accessible. Hence, the Nemesis has no incentive to attack these edges.

  Overall, this exit gadget simulates an exit edge of multiplicity k.

The reduction of a QSAT instance is performed in two steps. First, the QSAT instance is reduced to an equivalent instance of Nemesis on a multigraph. Then, this multigraph instance is translated into an equivalent instance of Nemesis on a simple graph.

This translation is achieved by replacing each vertex and each multiedge of the multigraph by the corresponding vertex and edge gadget described above. Each gadget simulates the behavior of its multigraph counterpart.

▶ **Lemma 11.** *The translation from the multigraph instance encoding an initial QSAT instance to the graph instance preserves the existence of a winning strategy for the fugitive or for the Nemesis.*

As a consequence, the fugitive has a winning strategy in the multigraph instance if and only if he has a winning strategy in the resulting graph instance. Since the size of the constructed graph is polynomial in the size of the initial QSAT instance, the reduction runs in polynomial time. Together with the PSPACE-hardness of Nemesis on multigraphs and the fact that Nemesis belongs to PSPACE, this proves that Nemesis is PSPACE-complete on graphs (Theorem 2).

4.4 PSPACE-completeness of the Cat Herding Problem

The PSPACE-completeness of Cat Herding follows as a corollary of the PSPACE-completeness of Nemesis with two exits. In the Cat Herding problem, the fugitive is a cat and the Nemesis is the cat herder. Unlike Nemesis, the underlying graph has no exits. The goal of the cat herder is to trap the cat on a single vertex in as few rounds as possible, while the goal of the cat is to remain free for as many rounds as possible.

To establish PSPACE-completeness, we consider the associated decision problem: given an integer k, does the cat have a strategy to remain free for more than k rounds? We reduce Nemesis to Cat Herding. An instance of Nemesis is given by a graph G with m edges, a set of exits E, and a starting vertex s. By Theorem 7, we may assume without loss of generality that $|E| = 2$. Let $\text{cat}_s(G)$ denote the maximum number of rounds the cat can remain free in G when starting from s. Clearly, $\text{cat}_s(G) \leq m$, since the cat herder deletes one edge per round.

We now construct an instance G' of Cat Herding from G. The graph G' is obtained by attaching a clique K_{2m} to each exit of G, identifying the exit vertex with one vertex of the corresponding clique. The size of each clique is polynomial in m and sufficiently large to ensure that, once the cat enters a clique, it can remain free for strictly more than $2m$ rounds, even if the cat herder has previously deleted up to m edges of the clique.

If the fugitive has a winning strategy in G, then the cat can reach one of the two cliques in G'. Once inside a clique, the cat can remain free for more than $2m$ rounds, and thus $\text{cat}_s(G') > 2m$.

Conversely, if the Nemesis has a winning strategy in G, then the cat herder can prevent the cat from reaching either clique. This takes at most m rounds. Once the cat is confined to G, the cat herder can delete the remaining edges of G and trap the cat in at most m additional rounds. Therefore, $\text{cat}_s(G') \leq 2m$.

It follows that the answer to the decision problem asking whether the cat can remain free for more than $2m$ rounds in G' is positive if and only if the fugitive has a winning strategy in the original Nemesis instance. This establishes a polynomial-time reduction from Nemesis to Cat Herding.

Finally, Cat Herding is a finite two-player game with perfect information and polynomially bounded play length. Hence, determining the winner can be done in polynomial space. Since Nemesis is PSPACE-complete (Theorem 3), we conclude that Cat Herding is PSPACE-complete.

Conclusion

We introduced Nemesis, with the somewhat surprising observation that this very simple game has not been previously investigated. We established several complexity results for different classes of graphs, but many open questions remain. Can the PSPACE-completeness of Nemesis be established for planar simple graphs? Do polynomial-time algorithms exist for planar graphs? Or for planar graphs with a fixed number of exits? The key feature of Nemesis and Blizzard is their simplicity, which suggests that understanding their structure may help shed light on other problems, as illustrated by the complexity result about the Cat Herding problem. Finally, we hope that these results illustrate how deceptively simple games on graphs can provide a rich playground for complexity theory while remaining fun to play.

References

1 Martin Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984. doi:10.1016/0166-218X(84)90073-8.

2 Rylo Ashmore, Danny Dyer, Trent Marbach, and Rebecca Milley. Cuts, Cats, and Complete Graphs. *Theoretical Computer Science*, page 115393, 2025. doi:10.1016/J.TCS.2025.115393.

3 Rylo Ashmore, Danny Dyer, and Rebecca Milley. Extremal Cat Herding, 2025. arXiv:2505.07588.

4 Rylo Ashmore and Sophie Pinchinat. Cat Herding Game Played on Infinite Trees. In *45th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2025)*, pages 10–1. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.

5 Hank Atkins. Nowhere to Go, 2012.

6 Pierre Bergé, Antoine Dailly, and Yan Gerard. Nemesis, an escape game in graphs, 2026. arXiv:2601.13841.

7 Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy. *Winning Ways for your Mathematical Plays*. Academic Press, 1982.

8 Béla Bollobás and Imre Leader. The Angel and the Devil in three dimensions. *Journal of Combinatorial Theory, Series A*, 113(1):176–184, 2006. doi:10.1016/J.JCTA.2005.03.009.

9 Anthony Bonato and Richard J. Nowakowski. *The game of cops and robbers on graphs*. American Mathematical Soc., 2011.

10 Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. On the complexity of connection games. *Theoretical Computer Science*, 644:2–28, 2016. doi:10.1016/J.TCS.2016.06.033.

11 Brian H Bowditch. The Angel Game in the Plane. *Combinatorics, Probability and Computing*, 16(3):345–362, 2007. doi:10.1017/S0963548306008297.

12 Nathan Bowler, Joshua Erde, Florian Lehner, and Max Pitz. Bounding the cop number of a graph by its genus. *SIAM Journal on Discrete Mathematics*, 35(4):2459–2489, 2021. doi:10.1137/20M1312150.

13 John H Conway. The Angel Problem. *Games of no chance*, 29:3–12, 1996.

14 Mark de Berg and Amirali Khosravi. Optimal Binary Space Partitions in the Plane. In *Computing and Combinatorics*, pages 216–225. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-14031-0_25.

15 Shimon Even and Robert Endre Tarjan. A Combinatorial Problem Which Is Complete in Polynomial Space. *Journal of the ACM (JACM)*, 23(4):710–719, 1976. doi:10.1145/321978.321989.

16 Fedor V Fomin, Petr A Golovach, Alexander Hall, Matúš Mihalák, Elias Vicari, and Peter Widmayer. How to guard a graph? In *International Symposium on Algorithms and Computation*, pages 318–329. Springer, 2008. doi:10.1007/978-3-540-92182-0_30.

17 Fedor V Fomin, Petr A Golovach, Jan Kratochvíl, Nicolas Nisse, and Karol Suchan. Pursuing a fast robber on a graph. *Theoretical Computer Science*, 411(7-9):1167–1181, 2010. doi:10.1016/J.TCS.2009.12.010.

18 Fedor V Fomin, Petr A Golovach, and Daniel Lokshtanov. Guard games on graphs: Keep the intruder out! *Theoretical computer science*, 412(46):6484–6497, 2011. doi:10.1016/J.TCS.2011.08.024.

19 Oddvar Kloster. A solution to the Angel Problem. *Theoretical Computer Science*, 389(1-2):152–161, 2007. doi:10.1016/J.TCS.2007.08.006.

20 Martin Kutz. Conway's angel in three dimensions. *Theoretical Computer Science*, 349(3):443–451, 2005. doi:10.1016/J.TCS.2005.08.034.

21 David Lichtenstein and Michael Sipser. GO Is Polynomial-Space Hard. *Journal of the ACM (JACM)*, 27(2):393–401, 1980. doi:10.1145/322186.322201.

22 Malaz Maamoun and Henry Meyniel. On a game of policemen and robber. *Discrete Applied Mathematics*, 17(3):307–309, 1987. doi:10.1016/0166-218X(87)90034-5.

23 Marcello Mamino. On the computational complexity of a game of cops and robbers. *Theoretical Computer Science*, 477:48–56, 2013. doi:10.1016/J.TCS.2012.11.041.

24 András Máthé. The Angel of Power 2 Wins. *Combinatorics, Probability and Computing*, 16(3):363–374, 2007. doi:10.1017/S0963548306008303.

25 Christos H Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991. doi:10.1016/0304-3975(91)90263-2.

26 Alain Quillot. *Jeux et points fixes sur les graphes*. PhD thesis, Université Paris VI, 1978. Thèse d'État.

27 Stefan Reisch. Hex ist PSPACE-vollständig. *Acta Informatica*, 15(2):167–191, 1981. doi:10.1007/BF00288964.

28 Johan Wästlund. A weaker winning angel, 2008.