Finding Shortest Walks in Kuru Kuru Kururin

Mickaël Laurent
Charles University, Prague, Czech Republic

Maher Mallem
Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP, UMR 5668, 69342, Lyon cedex 07, France

Abstract

This paper serves as a celebration of the twenty-fifth anniversary of *Kuru Kuru Kururin*. Although this video game is presented as a collection of two-dimensional puzzles based on rotation, it naturally invites players to complete its levels as quickly as possible. This has led to a surprisingly rich and challenging playing field to finding foremost temporal walks. In this work, we tackle this problem both in theory and in practice. First, we introduce a model for the game and provide an in-depth complexity analysis. Most notably, we show how each gameplay mechanic independently brings a layer of NP-hardness and/or co-NP-hardness. We also provide a pseudo-polynomial time algorithm for the general problem and identify several cases which can be solved in polynomial time. Along the way, we discuss connections to the more established framework of temporal graphs, both in the point model and the interval model. Then, we propose simple and flexible algorithmic techniques to reduce state space and guide the search, offering trade-offs between precision and computation speed in practice. These techniques were implemented and tested using a full recreation of the game physics and the levels from the original game. We demonstrate the efficiency of our framework in several settings - with or without taking damage, with or without unintended game mechanics - and relate empirical struggles which we encountered in practice to our complexity analysis. Our implementation is open source and fully available online, offering a novel and amusing setting to benchmark shortest path algorithms.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Shortest paths

Keywords and phrases shortest path, complexity

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Kuru Kuru Kururin is a video game published by Nintendo for the Game Boy Advance in 2001. This game is composed of a succession of levels, in which the player (the *helirin*) has to walk through a maze, from the *start area* to (one of the) *ending area(s)*.

Figure 1 The helirin never stops rotating (here, clockwise)

Figure 2 When hitting a wall, it loses a heart...

Figure 3 ...and is repelled (both in position and angle)

While studying the computational complexity of video games is nothing new (e.g., see [3, 9, 10] for classic video game series and [8, 17, 23] for pathfinding in video games), this game is unique in that the helirin constantly spins around at a fixed angular speed. The

player cannot control its rotation speed, but can move the helirin up, down, left, right, and diagonally at three possible speeds (slow, medium, or fast). The helirin initially has three hearts. When the helirin hits a wall, it loses one heart and gets repelled (both in position and angle) as shown in Figures 1, 2 and 3. When reaching zero hearts, the level must be restarted from the start area. The level may contain some healing areas, which restore all hearts and prevent the helirin from taking damage. The starting area also has this effect.

Some additional mechanics are introduced in later levels. For instance, *springs* that reverse the direction of rotation of the helirin (clockwise / counterclockwise) when hit, or *moving objects* (pistons and spiked balls) that follow a predefined path and must be avoided (cf. Figures 4, 5 and 6).

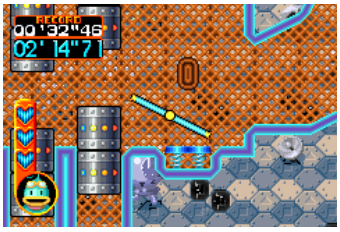

Figure 4 Some moving pistons and springs

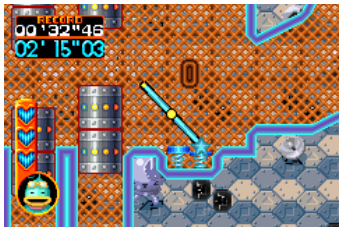

Figure 5 The helirin hits a spring...

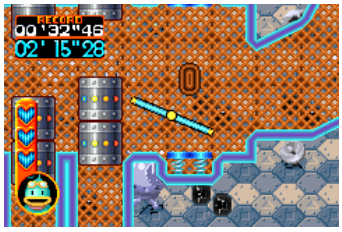

Figure 6 ...which reverses its rotation direction

This paper is organized as follows. In Section 2, we propose a simplified model of the base gameplay, and extend it with additional mechanics (springs and moving objects). Then, we study complexity of each variant in Section 3. Section 4 then studies the actual game implementation and shows how to design an approximate path finding algorithm despite the gigantic number of states. Finally, in Section 5 we introduce KuruBot, our path-finder implementation for *Kuru Kuru Kururin*, then we elaborate on how it can be configured to achieve different goals.

2 Preliminaries

▶ **Definition 1.** *$\mathbb{N}, \mathbb{Z}$, and $\mathbb{Q}$ are respectively defined as the set of positive integers, integers and rational numbers. $\mathbb{Z}^+$ and $\mathbb{Q}^+$ are respectively defined as the set of non-negative integers and non-negative rational numbers.*

2.1 Base Gameplay

▶ **Definition 2.** *A **helirin** $\mathcal{H}$ is defined as a tuple $(\ell, \nu_{card}, \nu_{diag}, \omega)$ which represents a half-length $\ell \in \mathbb{N}$ a cardinal speed $\nu_{card} \in \mathbb{N}$ a diagonal pointwise speed $\nu_{diag} \in \mathbb{N}$ and an angular speed $\omega\pi$, $\omega \in \mathbb{Q}^+$.*

▶ **Definition 3.** *A **base helirin state** $\mathcal{S}$ is defined as a tuple (x, y, α, b) representing a center position $(x, y) \in \mathbb{Z}^2$, an angle $\alpha\pi \in \mathbb{Q}\pi$, $0 \leq \alpha < 1$ and a boolean $b \in \{0, 1\}$ indicating whether the helirin is turning counterclockwise.*

We treat angles modulo π. Like in the unit circle, angles are measured from segment $[(0, 0), (0, 1)]$ and increase as you turn counterclockwise. By default, we assume that a helirin is turning counterclockwise - i.e., $b = 1$.

▶ **Definition 4.** *The **base moveset** $\mathcal{M}$ is $\{\emptyset, N, NE, E, SE, S, SW, W, NW\}$, i.e., a stand-still and the eight cardinal and diagonal directions "north", "north-east", etc..*

*A **base move** is a couple $(\mu, d) \in \mathcal{M} \times \mathbb{N}$ where d is the **duration** of the move.*

During a move, the helirin is constantly spinning around by an angle $\omega\pi$ (resp. $-\omega\pi$) per time unit if it is turning counterclockwise (resp. clockwise).

▶ **Definition 5.** *A **base helirin walk** $\mathcal{W} = (\mathcal{S}_0, t_0, (\mu_1, d_1), \ldots, (\mu_k, d_k))$ is a sequence of base moves from an initial state $\mathcal{S}_0$ and time t_0. The **duration** of walk $\mathcal{W}$ is defined as $d(\mathcal{W}) = d_1 + \ldots + d_k$.*

▶ **Definition 6.** *Let $s \in \mathbb{N}$. A **tile** is a $s \times s$ square in the plane which can interact with the helirin. The **base tileset** Σ features a **goal tile** kind, which only interacts with the center of the helirin, and **wall tile** kinds, which forbid a polygonal zone within the tile to intersect with any part of the helirin. There are five of them here: the plain square, and the four triangles filling half of the square.*

*A **base tile** is a tuple $(\kappa, s, x, y) \in \Sigma \times \mathbb{N} \times \mathbb{Z} \times \mathbb{Z}$ where κ is the tile kind, s is the side length and (x, y) are the coordinates of the bottom-left corner of the $s \times s$ square.*

We allow tiles to overlap each other. Plus, like in *Kuru Kuru Kururin*, we consider that goal tiles have priority over wall tiles. In other words, collisions with wall tiles are ignored if the center of the helirin is in a goal tile.

▶ **Definition 7.** *A base helirin state $\mathcal{S}$ is said to be **valid** if and only if the helirin does not intersect with a forbidden square of a wall tile. A base helirin walk $\mathcal{W}$ is said to be **valid** if and only if all intermediate states during the walk are valid.*

▶ **Definition 8.** *Problem* BaseKururin.

Input: A helirin $\mathcal{H}$, a set of base tiles $\mathcal{T} = \{(\kappa_1, s_1, x_1, y_1), \ldots, (\kappa_N, s_N, x_N, y_N)\}$, a base helirin state $\mathcal{S}_0$, times $t_0, D \in \mathbb{Z}^+$.

Question: Starting from state $\mathcal{S}_0$ and time t_0, is there a valid base helirin walk to a goal tile of duration at most D?

Let N be the number of tiles. For the sake of simplicity, in the rest of the paper, we assume that we have pre-implemented routines to check whether a given base helirin state is valid in $\mathcal{O}(N)$ time, and whether a given base helirin walk is valid in polynomial time.

2.2 Additional Gameplay Mechanics

▶ **Definition 9.** *A segment in the plane has the **mirroring property** if the following behavior occurs. Let q be the denominator of angular speed factor ω. If any part of the helirin would intersect the helirin after a unit-time move, the helirin rotation is reverted back by steps of $1/q$ until it no longer intersects with the spring tile edge - say, after an angle p/q. Then the rotation is reverted back by an additional angle p/q, and the helirin now turns the other way around.*

*A **spring tile** is a tile where exactly one edge of the square has the mirroring property.*

When BaseKururin is augmented with spring tiles, four tile kinds are added to base tileset Σ: one for each choice of edge which has the mirroring property.

▶ **Definition 10.** *A **piston** is defined as a tuple $(s, x, y, \tau, t_{on}, t_{off}, (\mu, d), \nu)$ which represents a square of side length $s \in \mathbb{N}$, the starting coordinates $(x, y) \in \mathbb{Z}^2$ of the bottom-left corner, a time period $\tau \in \mathbb{N}$, a periodic activating time $t_a \in \{0, \ldots, \tau - 1\}$, a periodic deactivating time $t_d \in \{0, \ldots, \tau - 1\}$, a base move $(\mu, d) \in \mathcal{M} \times \mathbb{N}$ and a (pointwise) speed $\nu \in \mathbb{N}$.*

▶ **Definition 11.** *A **spiked ball** is defined as a tuple $(r, x, y, \tau, t_0, (\mu, d), \nu)$ which represents a disk of radius $r \in \mathbb{N}$, the starting coordinates $(x, y) \in \mathbb{Z}^2$ of the bottom-left corner of the square which inscribes the disk outer circle, a time period $\tau \in \mathbb{N}$, a periodic starting time $t_0 \in \{0, \ldots, \tau - 1\}$, a base move $(\mu, d) \in \mathcal{M} \times \mathbb{N}$ and a (pointwise) speed $\nu \in \mathbb{N}$.*

In the presence of pistons and/or spiked balls, a base helirin walk is valid if and only if, on top of wall tiles, the helirin never intersects with them during the walk.

2.3 Other Problems

▶ **Definition 12.** *Problem* SubsetSum.
Input: Elements $a_1, \ldots, a_n \in \mathbb{N}$, target $B \in \mathbb{N}$.
Question: Is there a set $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = B$?

▶ **Definition 13.** *Let V be a set of vertices. A **timed arc** is a tuple (u, v, t, δ) representing a directed arc $(u, v) \in V^2$, a departure time $t \in \mathbb{Z}^+$ and a travel time $\delta \in \mathbb{N}$. A **point temporal graph** $\mathcal{G}$ is of the form $(V, \mathcal{A})$ where $\mathcal{A}$ is a set of timed arcs.*
*An **interval timed arc** is a tuple (u, v, t, t', δ) representing a directed arc $(u, v) \in V^2$, an earliest and a latest departure time $t, t' \in \mathbb{Z}^+$ and a travel time $\delta \in \mathbb{N}$. An **interval temporal graph** $\mathcal{G}$ is of the form $(V, \mathcal{A})$ where $\mathcal{A}$ is a set of interval timed arcs.*

▶ **Definition 14.** *A **temporal walk** W in a (point or interval) temporal graph $\mathcal{G} = (V, \mathcal{A})$ is a sequence of timed arcs $((u_1, v_1, t_1, \delta_1), \ldots, (u_k, v_k, t_k, \delta_k))$ such that, for all $i \in \{1, \ldots, k-1\}$, $v_i = u_{i+1}$ and $t_i + \delta_i \leq t_{i+1}$. In the point model, arcs must belong to $\mathcal{A}$. In the interval model, for every timed arc $(u_i, v_i, t_i, \delta_i)$ in the walk, there must be an interval timed arc (u, v, t, t', δ) in $\mathcal{A}$ such that $t \leq t_i \leq t'$. The walk is called **restless** if $t_1 = 0$ and $t_i + \delta_i = t_{i+1}$ for all i. The **arrival time** of W is $(t_k + \delta_k)$.*

▶ **Definition 15.** *Problem (*Restless*)ForemostTemporalWalk.*
Input: A (point or interval) temporal graph $\mathcal{G} = (V, \mathcal{A})$, two vertices $u, v \in V$, a time $D \in \mathbb{Z}^+$.
Question: Is there a (restless) temporal walk from u to v with arrival time at most D?

RestlessForemostTemporalWalk is in P on point temporal graphs [2], whereas it is weakly NP-hard on interval temporal graphs [22, 28], even with constant vertex-interval-membership-width [6]. ForemostTemporalWalk is in P with both models [5].

3 Computational Complexity

In this section, we establish the computational complexity of problem BaseKururin. We start from the original problem, then consider several gameplay mechanics one by one and study their respective impact on the problem complexity. Omitted proofs are available in Appendix A.

3.1 Base Gameplay

First, we show that the base problem is in PSPACE.

▶ **Theorem 16.** *BaseKururin is in* PSPACE.

Indeed, recall that PSPACE = NPSPACE from Savitch's theorem [25]. So, if there exists a valid base helirin walk $\mathcal{W} = (\mathcal{S}_0, t_0, (\mu_1, d_1), \ldots, (\mu_k, d_k))$, we can find it by guessing moves

(μ_i, d_i) in order, performing them on-the-fly while checking the validity of the corresponding walk prefixes. We do so while keeping track of the current base helirin state, and of time with a non-negative integer variable, checking that it never goes above D.

Next, we show that the base problem is NP-hard, regardless of the value of the angular speed.

▶ **Theorem 17.** *BaseKururin is weakly* NP*-hard even when* $\omega = 0$.

We reduce from weakly NP-hard problem SubsetSum [12]. We take inspiration from the NP-hardness proof of problem RestlessForemostTemporalWalk on interval temporal graphs by Zeitz [28]. Figure 7 illustrates the created instance of BaseKururin. For each choice of element a_i in SubsetSum, we have two branching paths: one where you go east for $2a_i$ time units, and another where you go north-east from a_i time units, then south-east for a_i time units. Both paths are then merged. Element a_i is included in the subset if and only if we chose the path going north-east then south-east.

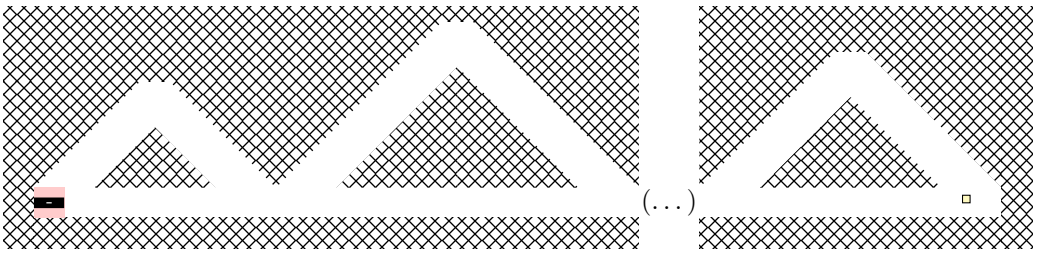

Figure 7 Layout of the NP-hardness reduction of BaseKururin. The isolated square at the right represents the single 1×1 goal tile

Note that this reduction would work with any angular speed ω: it purely relies on the offset between cardinal and diagonal speed values, which gets magnified over time. As such, time D needs to be encoded in binary for our reduction to work. In fact, if time D is given in unary in the input, then BaseKururin can be solved in polynomial time.

▶ **Theorem 18.** *BaseKururin is pseudo-polynomial-time solvable.*

We show this by encoding our instance as a point temporal graph with $\mathcal{O}(D^4)$ vertices and $\mathcal{O}(D^5)$ timed arcs. Each accessible center position can be encoded as a tuple $(n_N, n_{NE}, n_E, n_{SE})$ of values in $\{-D, \ldots, D\}$ giving a signed number of unit-time base moves in each direction in order to reach it. And timed arcs are of the form $(u, v, t, 1)$, where u represents a center position accessible at time t and v represents a center position accessible from u with one of the nine unit-time base moves. If N is the number of tiles in our instance I of BaseKururin then, in $\mathcal{O}(D^5 N)$ time, we can compute all accessible center positions, including whether they are in a goal tile, and compute an equivalent point temporal graph $\mathcal{G}_I$. Then problem ForemostTemporalWalk can be solved in linear time on $\mathcal{G}_I$ to look for a walk from the starting center position to all accessible center positions in goal zones simultaneously [4].

▶ Remark 19. Conversely, ForemostTemporalWalk on point temporal graphs can be reduced to BaseKururin.

Additionally, in the proof of Theorem 17, note that wall tiles were crucial to enforce specific durations for the diagonal base moves which corresponded to the elements in the instance of SubsetSum. If there are no such walls then, for each goal tile, one can compute a base

helirin walk reaching it in minimum duration via an integer linear program with a constant number of variables and constraints. We have eight non-negative integer variables indicating the number of unit-time base moves in each of the eight directions. We aim to minimize their sum while ending the walk in the square goal tile (i.e., with four constraints). Each system can be solved in $\mathcal{O}(1)$ time (e.g., see [1]), so we can solve our problem in $\mathcal{O}(N)$ time.

▶ **Theorem 20.** *BaseKururin with no wall tiles is polynomial-time solvable.*

3.2 Base Gameplay with Diagonal Speed Restrictions

In this subsection, we restrict the value of diagonal pointwise speed ν_{diag} such that we cannot reuse the trick used to prove the NP-hardness of BaseKururin in Theorem 17. In particular, if $\nu_{diag} \in \{0, \nu_{card}\}$, then one can no longer create a position offset other than a multiple of ν_{card}. In fact, the graph of accessible center positions with unit-time move neighborhood is a grid graph, with a four-neighborhood if $\nu_{diag} = 0$ and an eight-neighborhood if $\nu_{diag} = \nu_{card}$. With such neighborhoods, it is not hard to see that the triangle inequality is satisfied. As such, if additionally the helirin does not spin around, then the instance of BaseKururin can be easily turned into such grid graphs and solved efficiently. E.g., by using the visibility graph induced by the corners of the wall tiles [18, 21, 27], we can solve this particular case in $\mathcal{O}(N^2)$ time.

▶ **Theorem 21.** *BaseKururin with $\nu_{diag} \in \{0, \nu_{card}\}$ and $\omega = 0$ is polynomial-time solvable.*

However, if the helirin is allowed to spin around, then we believe that our problem is again unlikely to be polynomial-time solvable.

▶ **Conjecture 22.** *BaseKururin is weakly* co-NP*-hard even when $\nu_{diag} \in \{0, \nu_{card}\}$.*

This would show that both speed offset and rotation independently make our problem difficult.

3.3 Base Gameplay with Spring Tiles

If spring tiles are available, then the reduction of Theorem 17 can be adapted by using the mirroring property to create an angle offset in the rotation of the helirin between branching paths. Arbitrary angle offsets can be obtained within constant duration, so this reduction does not require time D to be given in binary in the input.

▶ **Theorem 23.** *BaseKururin augmented with spring tiles is weakly* NP*-hard even when $\nu_{diag} \in \{0, \nu_{card}\}$ and time D is given in unary.*

Still, the algorithm from Theorem 18 can be adapted if, on top of time D, the denominator q_ω of angular speed factor ω is given in unary in the input. Then the number of vertices in the equivalent point temporal graph is in $\mathcal{O}(D^4 q_\omega)$.

▶ **Theorem 24.** *BaseKururin augmented with spring tiles is pseudo-polynomial-time solvable.*

3.4 Base Gameplay With Pistons and Spiked Balls

If pistons or spiked balls are available, then the algorithm from Theorem 18 can be easily adapted with minor time overhead. Indeed, these elements are time-indexed so, at each time, their location can easily be determined and taken into account when computing the set of accessible center positions.

▶ **Theorem 25.** *BaseKururin augmented with pistons and spiked balls is pseudo-polynomial-time solvable.*

However, having these moving elements makes our problem difficult independently from all previous gameplay mechanics - namely speed offset, rotation and spring tiles.

▶ **Theorem 26.** *BaseKururin augmented with pistons or spiked balls is weakly* NP*-hard and weakly* co-NP*-hard even when* $\nu_{diag} \in \{0, \nu_{card}\}$*,* $\omega = 0$ *and half-length* ℓ*, cardinal speed* ν_{card} *and tile sizes* s_i *are given in unary.*

Indeed, note that spiked balls with standstill base moves and pistons with unit-time base moves essentially act as periodic on/off switches, with periods written in binary. So, these periods can be given exponential values within logarithmic working space [7, 24], which is the key ingredient in our reductions. For instance, this allows us to test all valuations of a boolean formula - and thus encode both Sat and DNF-Tautology.

▶ Remark 27. (Restless)ForemostTemporalWalk on interval temporal graphs can be reduced to BaseKururin augmented with spiked balls.

▶ Remark 28. Finally, since the problem is both NP-hard and co-NP-hard, it is unlikely that it belongs to either class. Indeed if, e.g., it belonged to NP, then co-NP-complete problem DNF-Tautology would also belong to NP. This would mean that NP = co-NP, which would imply a collapse of the Polynomial Hierarchy to the second level [26].

4 Algorithmic Techniques

In this section, we provide algorithmic techniques aimed at tackling real instances of *Kuru Kuru Kururin*.

4.1 State Space

Recall that a base helirin state features a center position (x, y), an angle α, and a boolean b indicating whether the helirin is turning counterclockwise. In the actual implementation of *Kuru Kuru Kururin*, these are stored as follows:

56 bits position: 32 bits for x, 32 bits for y, with the 4 most-significant bits unused in practice. The 16 less-significant bits are sub-pixel precision—the position in pixels can be retrieved by only reading the 16 most-significant bits,

16 bits angle (evenly captures the $[0, 2\pi[$ interval),

1 bit rotation direction (clockwise / counterclockwise).

In addition, when colliding with walls, the helirin gets repelled[1]. This mechanism is modeled by adding the following components to the state:

40 bits bump speed: 32 bits for x, 32 bits for y, with the 12 most-significant bits unused in practice. Bump speed is added to the helirin when hitting a wall and decreases with time,

12 bits rotation speed: 16 bits, with the 4 most-significant bits unused in practice. Rotation speed is added to the helirin when hitting a wall, and stabilizes towards the base rotation speed with time.

[1] For the sake of simplicity, in Section 2 we did not model this knock-back mechanism, and instead focused on a variant where collisions with walls are not allowed. Though it is clear that it would make the base problem more difficult, at least by considering angle offsets in the helirin rotation similarly to springs.

This amounts to a total of 125 bits, i.e., roughly $6 \cdot 10^{37}$ states, which would be too massive to explore exhaustively. Encoding base helirin states with a temporal graph as in Theorem 18 would help in shorter levels, though not nearly enough. Indeed, *Kuru Kuru Kururin* level durations typically range from a couple seconds to a minute, so consider a level which can be finished in ten seconds. Since the game runs at roughly sixty frames per second, this would still amount to $600^4 \cdot 2^{52} \simeq 6 \cdot 10^{26}$ states. Consequently, looking for optimal resolution by exhaustive search seems impractical. In response, we propose a custom implementation of the A* algorithm, which we detail in the next section.

Note that, for some levels and applications, even more components must be added to the state (e.g., hearts and invulnerability frames for non-damageless gameplay as in Section 5.2, or the position of the moving objects for maps that contain some). More information about the state and how it evolves can be found in the notes of our first *tool-assisted speedrun* [15].

4.2 Custom A* Algorithm

In order to find short walks in spite of the very large state graph, we propose a custom implementation of the A* search algorithm. First, a heuristic function is computed by computing distance of every point to the target in an approximated variant of the problem (we call this heuristic function the "cost map", Section 4.2.1). Then, this heuristic function is used to guide a custom A* search algorithm (Section 4.2.2).

4.2.1 Cost map

The heuristic function used to guide the A* search is computed using a Dijkstra algorithm that starts at the end of the level and computes, for each state, its distance to the end. However, the state space used for this heuristic function is harshly approximated: it assimilates the helirin with a point (thus making all angle-related state irrelevant), and ignores movements due to the collision with walls (thus eliminating the state related to bump speed). Depending on the application, walls can either be considered impassable, or passable under certain conditions and at a certain cost (cf. Section 5). In this context, as walls are aligned on pixels, the precision of the position can be reduced to the pixel unit, thus saving 32 bits of state. The state is thus only composed of the x and y position of 12 bits each, yielding 2^{24} states (which is perfectly fine to explore exhaustively).

This yields a cost map that associates to each pixel of the map a distance to the end of the level (or to a custom target). It is then used to guide the A* algorithm for the resolution with the full state. This cost map is not necessarily an under-approximation of the real cost, as considering the helirin as being punctual allows turning closer to the walls. Consequently, we cannot guarantee that the A* search that find an optimal path. Also note that the cost map can be multiplied by a constant factor to parametrize the influence it has on the search: a factor higher than 1 will often explore fewer states, resulting in a faster resolution but a solution that is less optimal.

4.2.2 Reducing the search space

The algorithm we use to find a minimal path is based on the A* search algorithm [11] with the heuristic function h induced by our cost map:

1. Let Q be a priority queue. For each starting state s, the pair $(s, 0)$ is added to Q with priority $h(s)$.
2. The pair (s, l) of minimum priority is extracted from Q.

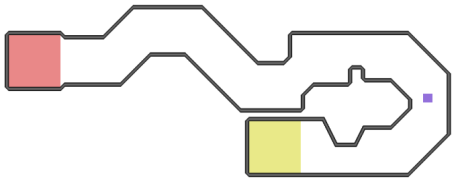

Figure 8 Map of the first level: Grasslands 1 (start area, ending area)

Figure 9 Associated cost map (unpassable walls)

- If this is a final state, then the algorithm terminates: a path of length l has been found.
- Otherwise, the successors of s are computed, and for each such successor s', the pair $(s', l+1)$ is added to Q with priority $l + 1 + h(s)$. This step is then repeated.

In practice, this algorithm does not terminate even for simple levels. Thus, we implemented two main strategies for reducing the search space.

Equivalence classes

A first way to reduce the search space is to define equivalence classes regrouping states that are *sufficiently similar*. This is done by defining a state normalization function $n(s)$ that truncates some of the state components to the desired precision (that can be configured depending on the application, cf. Section 5). Two states s_1 and s_2 are in the same equivalence class if and only if $n(s_1) = n(s_2)$.

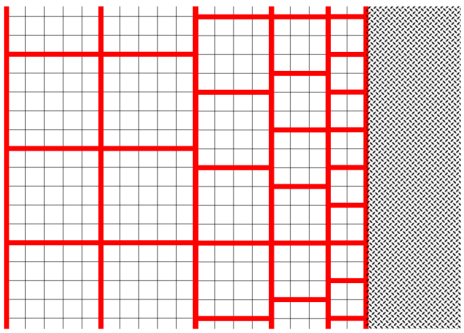

Figure 10 Equivalence classes for position (smaller near the wall on the right)

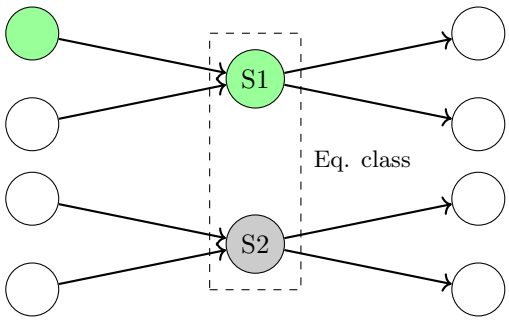

Figure 11 Path finding: node S2 is disabled because S1 has been reached first

The searching algorithm is then amended to never explore two states of the same equivalence class. For that, it maintains a set of normalized states S: when a state s is extracted from the priority queue Q, $n(s)$ is added to S – if it was already present, then s is removed from Q without exploring it. This is illustrated by Figure 11.

Also note that we may want a dynamic resolution for the search: states that are close to a wall may benefit from a higher resolution because of the complex dynamic of collisions (in particular when solving for unrestricted gameplay, cf. Section 5.2). Thus, our normalization function n implements adaptive precision: the closer to a wall a state is, the fewer digits are truncated. This is illustrated in Figure 10.

Piecewise solving

Another way to reduce the search space is to perform a piecewise solving: *checkpoints* are inserted in the level, fragmenting the map into segments that are solved successively. Once a checkpoint has been reached, no other path to this checkpoint will be explored, and the search for a path to the next checkpoint starts.

This strategy can be achieved in two ways. First, it can be achieved by manually performing multiple searches: a first one from the starting state to an arbitrary area (checkpoint 1), then another search from the resulting state to another area (checkpoint 2), and so on. Second, it can be achieved in an automated way by modifying the cost map so that reaching a checkpoint area induces a huge decrease of the estimated cost. This is illustrated in Figures 12, 13 and 14 where checkpoints have been added on healing areas (these are good spots to insert checkpoints as they offer a great mobility: when the helirin is in a healing area, it can easily change its angle by hitting a wall without taking damage).

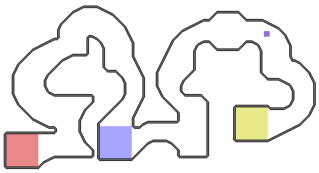

Figure 12 Map (start area, healing area, ending area)

Figure 13 Cost map (no checkpoint)

Figure 14 Cost map (with checkpoint)

5 Applications

5.1 Implementation: KuruBot

The exact physics of the game Kuru Kuru Kururin as well as the path finding algorithm described in Section 4 have been implemented in KuruBot [14], a C# application of about 4000 lines of code. It can also communicate with the emulator BizHawk in order to retrieve the current map and state from a live session, find a path, and play it on the emulator. Several configurations can be loaded to achieve different goals that we present in this section.

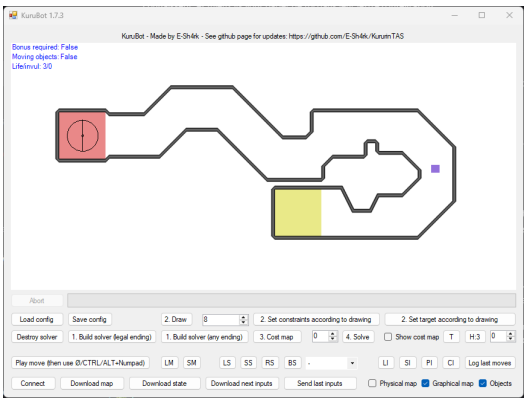

Figure 15 Interface of KuruBot

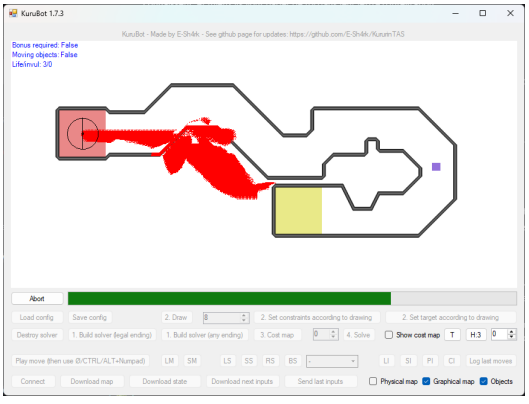

Figure 16 KuruBot while searching for a path

5.2 Categories

Levels can be solved with different constraints ("*categories*"). The main categories are presented below. Solving each category requires using an adapted configuration for KuruBot, which are summarized in Figure 17.

Category	Cost map	Resolution (base, near walls)	Checkpoints
Damageless	No wall crossing	2 px, 2 px	Automated, at healing areas
Regular intended	No wall crossing	2 px, 1 px	Automated, at healing areas
Regular unrestricted	Wall crossing if hearts > 1	1 px, 1/8 px	Manual, short segments

Figure 17 Configuration adapted for each category

Damageless completion

The damageless completion category consists in finishing every level without taking any damage. Note that hitting a wall may still be permitted if the helirin is in a healing zone (thus, the bump speed and rotation speed components of the state cannot be neglected). A *tool-assisted speedrun* of this category made using KuruBot is available on TASVideos.org [16].

Regular completion, intended gameplay

For regular completion, the helirin is allowed to hit walls, which can be used to walk through narrow spaces, to have a boost due to the speed bump, and to change the angle of the helirin. When hitting a wall outside a healing area, the helirin looses a heart and becomes invulnerable for 20 frames. Thus, the following bits must be added to our state:

2 bits remaining hearts (max: 3),

5 bits invulnerability frames (max: 20).

Regular completion, unrestricted gameplay

The unrestricted category allows exploiting the physics of the game, in particular the way collisions are handled, to break through walls. However, such manipulations are only possible for some wall placements, and require a very precise combination of position, angle, and bump speed to be performed. Consequently, for this category, the resolution of the solver (cf. Section 4.2.2) must be increased (in particular the resolution near walls).

The cost map computation is also modified: crossing walls is allowed, but at a constant cost to account for the number of frames it takes to cross a wall in average (e.g. 10 frames), and only if the helirin has at least 2 hearts (otherwise, hitting a wall will lose the level). Consequently, two cost maps must be computed: one that will be used when the helirin has at least 2 hearts left, and one when it has only one heart (cf. Figures 18 and 19). Positions that are inside walls also get a constant cost reduction to account for the fact that these are advantageous areas inside which the helirin should stay as long as possible (in particular, the helirin can move faster when it is inside a wall). A *tool-assisted speedrun* of this category made using KuruBot is available on TASVideos.org [20].

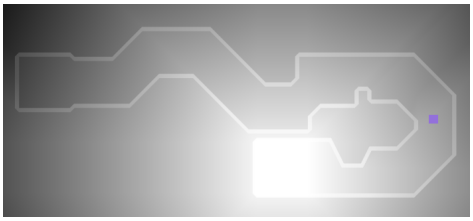

Figure 18 Unrestricted gameplay cost map (with extra hearts)

Figure 19 Unrestricted gameplay cost map (no extra heart)

Other incentives

We may want to search for solutions that minimize measures other than the total number of frames. For instance, minimizing the number of input changes (i.e. the number of times a button must be pressed or released) can also be of interest in order to find *humanly-viable* ways to cross a wall for speedrunners. Such paths have already been found by using KuruBot with an adapted configuration [19].

6 Conclusion

In this work, we presented an algorithmic and complexity study of *Kuru Kuru Kururin*, a puzzle-action game whose distinctive mechanics naturally give rise to rich shortest-walk problems. By formalizing the game's behavior and progressively extending the model with additional mechanics, we revealed how each gameplay element (rotation, speed asymmetries, spring tiles, moving objects, etc.) introduces its own source of computational hardness. Our results show that even the base problem is weakly NP-hard, and that various mechanics independently raise the difficulty to NP-hard or even co-NP-hard levels. Despite these hardness results, when time or mechanical constraints are restricted, several variants admit pseudo-polynomial-time or even polynomial-time algorithms.

In the second part of this article, focusing this time on approximate resolution, we introduced practical algorithmic techniques, including a state-space reduction strategy and a custom A* search guided by approximate cost maps, that enable efficient solving of real game levels despite an astronomically large underlying state space. Finally, we implemented these ideas in KuruBot, a full physics-accurate path-finding framework capable of exploring a wide variety of gameplay settings. Overall, this work bridges complexity theory, temporal graph reasoning, and game physics modeling, showing how a charming 2001 Game Boy Advance title can serve as a surprisingly deep computational playground.

Going forward, in the short term we plan to benchmark the performance of our path finding algorithm in the presence of the different elements introduced in the complexity analysis. Using a level editor that we created [13], we plan to generate multiple levels, each focusing on a specific mechanic: base gameplay without diagonal moves, base gameplay with diagonal moves, springs, moving objects, etc. The performance on these benchmark levels will then be evaluated using a specific configuration of KuruBot, and compared to the conclusions of our complexity analysis. Moreover, such levels could also serve as an amusing playing field to compare state-of-the-art pathfinding algorithms.

Finally, in the future, our complexity analysis could be extended to model the whole game, including knock-back, the life system and healing areas, as well as the sequels *Kururin Paradise* and *Kururin Squash!*, hidden gems which were only released in Japan and feature even more unique game mechanics to explore.

References

1. Nina Amenta, Jesús A De Loera, and Pablo Soberón. Helly's theorem: new variations and applications. *Algebraic and geometric methods in discrete mathematics*, 685:55–95, 2017.
2. Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(1):73, 2020.
3. Jeffrey Bosboom, Josh Brunner, Michael J. Coulombe, Erik D. Demaine, Dylan H. Hendrickson, Jayson Lynch, and Elle Najt. The legend of zelda: The complexity of mechanics. *CoRR*, abs/2203.17167, 2022. URL: https://doi.org/10.48550/arXiv.2203.17167, arXiv:2203.17167, doi:10.48550/ARXIV.2203.17167.
4. Filippo Brunelli and Laurent Viennot. Computing temporal reachability under waiting-time constraints in linear time. In David Doty and Paul Spirakis, editors, *2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023)*, volume 257 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:11, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAND.2023.4.
5. Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003. doi:10.1142/S0129054103001728.
6. Justine Cauvi and Laurent Viennot. Parameterized restless temporal path. In Artur Jeż and Jan Otop, editors, *Fundamentals of Computation Theory*, pages 82–93, Cham, 2026. Springer Nature Switzerland. doi:10.1007/978-3-032-04700-7_7.
7. Yiping Cheng. Space-efficient Karatsuba multiplication for multi-precision integers. *arXiv preprint arXiv:1605.06760*, 2016. URL: https://arxiv.org/abs/1605.06760.
8. Xiao Cui and Hao Shi. Direction oriented pathfinding in video games. *International Journal of Artificial Intelligence & Applications*, 2(4):1, 2011.
9. Erik D. Demaine, Holden Hall, Hayashi Layers, Ricardo Ruiz, and Naveen Venkat. You can't solve these super mario bros. levels: Undecidable mario games. *Theoretical Computer Science*, 1060:115549, 2026. URL: https://www.sciencedirect.com/science/article/pii/S0304397525004876, doi:https://doi.org/10.1016/j.tcs.2025.115549.
10. MIT Hardness Group, Erik D. Demaine, Holden Hall, and Jeffery Li. Tetris with few piece types. *Theoretical Computer Science*, 1059:115581, 2026. URL: https://www.sciencedirect.com/science/article/pii/S0304397525005195, doi:https://doi.org/10.1016/j.tcs.2025.115581.
11. Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.
12. Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.
13. Mickaël "E-Sh4rk" Laurent and contributors. Repository of ROM-hacking tools for the GBA Kururin games, 2021. URL: https://github.com/E-Sh4rk/KuruTools.
14. Mickaël "E-Sh4rk" Laurent and contributors. Repository of KuruBot and other TAS tools, 2024. URL: https://github.com/E-Sh4rk/KururinTAS.
15. Mickaël "E-Sh4rk" Laurent and Maher "mohoc" Mallem. Author notes of the tool-assisted speedrun of Kuru Kuru Kururin on TASVideos.org, 2019. URL: https://tasvideos.org/6314S.
16. Mickaël "E-Sh4rk" Laurent and Maher "mohoc" Mallem. Tool-assisted speedrun of Kuru Kuru Kururin - baseline category, 2019. URL: https://tasvideos.org/3933M.
17. Sharmad Rajnish Lawande, Graceline Jasmine, Jani Anbarasi, and Lila Iznita Izhar. A systematic review and analysis of intelligence-based pathfinding algorithms in the field of video games. *Applied Sciences*, 12(11), 2022. URL: https://www.mdpi.com/2076-3417/12/11/5499, doi:10.3390/app12115499.

18 Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

19 Maher "mohoc" Mallem. Kuru Kuru Kururin speedrunning guide - pause frame document, 2024. URL: `https://www.speedrun.com/kuru_kuru_kururin/resources/ecdzb`.

20 Maher "mohoc" Mallem. Tool-assisted speedrun of Kuru Kuru Kururin - 100% category, 2024. URL: `https://tasvideos.org/6116M`.

21 Marcell Missura, Daniel D. Lee, and Maren Bennewitz. Minimal construct: Efficient shortest path finding for mobile robots in polygonal maps. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7918–7923, 2018. `doi:10.1109/IROS.2018.8594124`.

22 Ariel Orda and Raphael Rom. Traveling without waiting in time-dependent networks is NP-hard. Technical report, Dept. Electrical Engineering, Technion - Israel Institute of Technology, Haifa, Israel 32000, 1989.

23 Sara Lutami Pardede, Fadel Ramli Athallah, Yahya Nur Huda, and Fikri Dzulfiqar Zain. A review of pathfinding in game development. *CEPAT] Journal of Computer Engineering: Progress, Application and Technology*, 1(01):47, 2022.

24 Daniel S. Roche. Space- and time-efficient polynomial multiplication. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, page 295–302, New York, NY, USA, 2009. Association for Computing Machinery. `doi:10.1145/1576702.1576743`.

25 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. URL: `https://www.sciencedirect.com/science/article/pii/S002200007080006X`, `doi:https://doi.org/10.1016/S0022-0000(70)80006-X`.

26 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976. `doi:https://doi.org/10.1016/0304-3975(76)90061-X`.

27 Tansel Uras, Sven Koenig, and Carlos Hernandez. Subgoal graphs for optimal pathfinding in eight-neighbor grids. *Proceedings of the International Conference on Automated Planning and Scheduling*, 23(1):224–232, Jun. 2013. URL: `https://ojs.aaai.org/index.php/ICAPS/article/view/13568`, `doi:10.1609/icaps.v23i1.13568`.

28 Tim Zeitz. NP-hardness of shortest path problems in networks with non-FIFO time-dependent travel times. *Information Processing Letters*, 179:106287, 2023. `doi:https://doi.org/10.1016/j.ipl.2022.106287`.

A Complexity Proofs

A.1 Theorem 17

Proof. We reduce from weakly NP-hard problem SubsetSum [12]. Let $a_1, \ldots, a_n$ be the elements and let B be the target. Let $A = \sum_{1 \leq i \leq n} a_i$ and $L = 2A + 1$.

We propose the following instance of BaseKururin. We set the helirin properties to $(\ell, \nu_{card}, \nu_{diag}, \omega) = (L, 2L, 2L - 1, 0)$. We set $D = 2A + n - 1$ and $\mathcal{S}_0 = (L, L, 0, 1)$. We set wall tiles to form the instance illustrated in Figure 7. For each choice of element a_i in SubsetSum, we have two branching paths: one, which we call $P_{i,0}$, where you go east for $2a_i$ time units, and another, which we call $P_{i,1}$, where you go north-east for a_i time units, then south-east for a_i time units. Both paths are then merged. Finally, there is a single 1×1 goal tile at position $((2D + 1)L - 2B, L)$.

If our instance of SubsetSum has a solution $S \subseteq \{1, \ldots, n\}$, we propose base helirin walk $\mathcal{W}$ where, for all $i \in \{1, \ldots, n\}$, we take path $P_{i,1}$ if $i \in S$ and path $P_{i,0}$ otherwise. This walk is valid and ends at position $((2D + 1)L - 2(\sum_{i \in S} a_i), L) = ((2D + 1)L - 2B, L)$. Conversely, if we have a valid base helirin walk $\mathcal{W}$, by our choice for speed values ν_{card}, ν_{diag}

and time D, only the east, north-east and south-east directions can be used. Therefore, according to the position of wall tiles, the underlying path from base helirin walk $\mathcal{W}$ is necessarily of the form $P_{i,j_1} \cdots P_{n,j_n}$ with $j_i \in \{0,1\}$ for all i. We propose $S = \{i \in \{1,\dots,n\} | j_i = 1\}$. Then, because $\mathcal{W}$ is valid and ends at position $((2D+1)L - 2B, L)$, we have: $2B = 2(\sum_{i \in S} a_i)$. ◀

A.2 Remark 19

The main idea is to keep track of time with the rotation angle. By taking a large enough half-length ℓ of the helirin compared to the denominator q_ω of angular speed factor ω, we can discriminate between angles which are multiples of π/q_ω, e.g., by setting wall tiles in a similar way as in the proof of Theorem 23. Using this trick, having the time span of the input temporal graph fit within a single half-rotation, we can replicate the layout of the input temporal graph, and only make the start and the end of the representation of each timed arc available at specific angles - and thus at specific times.

A.3 Theorem 20

Proof. We have eight non-negative integer variables $(x_\mu)_{\mu \in \mathcal{M} \setminus \{\emptyset\}}$, representing the time spent going in each of the eight directions. For each goal tile of side length $s^{(i)}$ and bottom-left corner coordinates $(x^{(i)}, y^{(i)})$, we propose the following integer linear program:

$$
\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{\mu \in \mathcal{M} \setminus \{\emptyset\}} x_\mu \\
\text{subject to} & \nu_{card}(x_E - x_W) + \nu_{diag}(x_{NE} + x_{SE} - x_{NW} - x_{SW}) \geq x^{(i)}, \\
& \nu_{card}(x_E - x_W) + \nu_{diag}(x_{NE} + x_{SE} - x_{NW} - x_{SW}) \leq x^{(i)} + s^{(i)}, \\
& \nu_{card}(x_N - x_S) + \nu_{diag}(x_{NE} + x_{NW} - x_{SE} - x_{SW}) \geq y^{(i)}, \\
& \nu_{card}(x_N - x_S) + \nu_{diag}(x_{NE} + x_{NW} - x_{SE} - x_{SW}) \leq y^{(i)} + s^{(i)}, \\
& x_\mu \in \mathbb{Z}^+, \mu \in \mathcal{M} \setminus \{\emptyset\}.
\end{array}
$$

This ensures that we land on the goal tile as soon as possible. We have a constant number of variables and constraints, so the system can be solved in $\mathcal{O}(1)$ time (e.g., see [1]). Thus, by taking the minimum over all goal tiles, we can conclude whether, starting from base helirin state $\mathcal{S}_0$, there is a valid base helirin walk to a goal tile with duration at most D. ◀

A.4 Theorem 23

Proof. (Sketch.) We reduce from weakly NP-hard problem SubsetSum [12]. Let $a_1, \dots, a_n$ be the elements and let B be the target. Let $A = \sum_{1 \leq i \leq n} a_i$ and $L = 16(A + 2n + 1)$.

We propose a similar reduction to the proof of Theorem 17, except we rely on rotation angle offsets instead of position offsets. We set $D = 6n + 1$, $q_\omega = L/2$ and $p_\omega = 1 + q_\omega/8$. We set the helirin properties to $(\ell, \nu_{card}, \nu_{diag}, \omega) = (L, 2L, 0, p_\omega/q_\omega)$ and we set $\mathcal{S}_0 = (L, L, 0, 1)$. We set wall tiles to form the instance illustrated in Figure 20. For each choice of element a_i in SubsetSum, we have two branching paths: one, which we call $P_{i,0}$, where you go south, then east for 2 time units, then north, and another, which we call $P_{i,1}$, where you go north, then east for 2 time units, then south. Both paths are then merged. Plus, there is a single $2L \times 2L$ goal tile at position $((4n+1)2L, 0)$.

Now, let $\mathcal{P} = \{0, \dots, q_\omega/2\}$. Given $p \in \mathcal{P}$, let (x_p, y_p) be the coordinates of the right extremity of the helirin at angle $p\pi/q_\omega$. Since $L > 2q_\omega/\sqrt{2}$, function $(p \mapsto (x_p, y_p))$ is injective over $\mathcal{P}$, and sequences $(x_p)_{p \in \mathcal{P}}$ and $(y_p)_{p \in \mathcal{P}}$ are respectively non-decreasing and

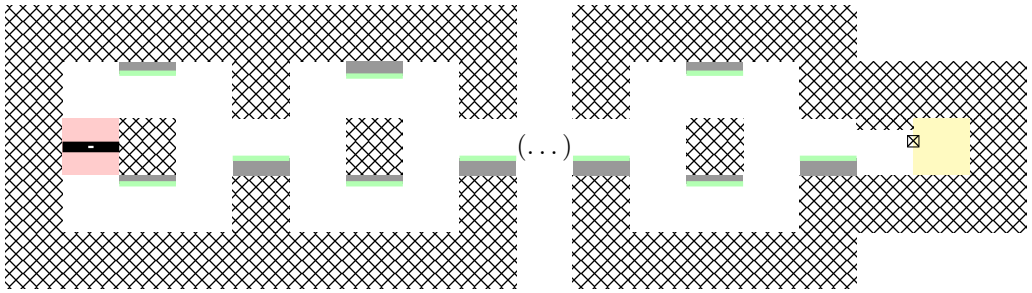

Figure 20 Layout of the NP-hardness reduction with spring tiles. The isolated square at the left of the goal tile is a wall tile which is part of the angle check gadget, which only allows for angle $(B + 2n)\pi/8L$.

non-increasing. Plus, for each $p \in \mathcal{P}$, couple (x_p, y_p) is merely a rational approximation of the cosine and sine of angle $p\pi/q_\omega$, and thus can be computed in polynomial time.

Knowing this, we can use these values to position our spring tiles accordingly. In bottom paths $P_{i,0}$, we rely on numerator $(p = q_\omega/4+1)$ to align the edges with the mirroring property. In bottom paths $P_{i,1}$, we rely on numerator $(p = q_\omega/4 + 1 - a_i)$ instead. And, once both paths are merged, we rely on numerator $(p = -q_\omega/8)$. Finally, right before the goal tile, we set two wall tiles in order to block positions (x_p, y_p) for every $p \in \mathcal{P}\backslash\{B + 2n\}$. Then the correspondence between valid base helirin walks and solutions of the SubsetSum instance unfolds similarly to the proof of Theorem 17. Finally, one can easily adapt the reduction to case $\nu_{diag} = \nu_{card}$.

◀

A.5 Theorem 26

We detail proofs which make use of spiked balls with standstill base moves. In both of them, we set the helirin properties to $(\ell, \nu_{card}, \nu_{diag}, \omega) = (1, 2, 0, 0)$ and base helirin state $\mathcal{S}_0$ to $(x, y, \alpha, b) = (1, 1, 0, 0)$. Plus, all spiked balls will have radius 1, $\mu = \emptyset$ in their base move, and $\nu = 1$. So, all relevant elements will be set along a grid of 2×2 squares.

NP-hardness

Proof. We reduce from 3-Sat. Let $\varphi = \bigwedge_{1 \le i \le m}(l_{i,1} \vee l_{i,2} \vee l_{i,3})$ be a 3-CNF formula with m clauses and n variables, w.l.o.g. with exactly three literals per clause. We encode the formula as an obstacle course going from left to right. Figure 21 represents the base layout, which can be easily obtained with square wall tiles. Each vertical layer represents a clause and has three available ways, each associated to a literal in this clause. And we set a single goal tile of side length 2 at $(x, y) = (4m + 4, 0)$. Then, going from the start area (in red) to the goal tile (in yellow) with no wait takes time at most $T = 6m + 6$.

The plan is to dedicate time T for each of the 2^n valuations of the variables to give them a chance to reach the goal tile from the start, which will correspond to making φ true. First, we only allow the helirin to leave the start area at periodic times $(1 \mod T)$. We do so with a spiked ball at $(x, y) = (2, 0)$ with $\tau = T, t_0 = 2$ and $d = T - 1$. Plus, once the helirin leaves the start area, we force it to reach the goal tile by the end of the time period. We do so by setting spiked balls for every $2 \le i \le 2m + 2$ and $0 \le j \le 4$ at $(x, y) = (2i, 2j)$ with $\tau = T, t_0 = 0$ and $d = 1$.

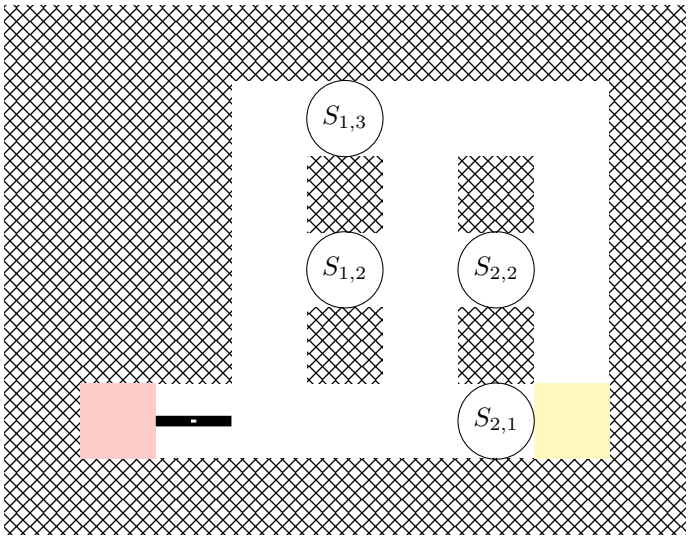

Figure 21 Layout of the NP-hardness reduction with spiked balls, illustrated with input formula $\varphi = (x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \neg x_4)$. We are at time $\Delta + 1$, i.e., about to check valuation $(x_4, x_3, x_2, x_1) = (0, 0, 0, 1)$, which makes φ true by choosing literals x_1 then $\neg x_4$.

Now, for $1 \leq k \leq n$ we tie variable x_k to spiked balls of time period $\tau = T2^k$. More precisely, given $1 \leq i \leq m$ and $1 \leq j \leq 3$, we set a spiked ball $S_{i,j}$ at $x = 4i + 2$ and $y = 4j - 4$, with move duration $d = T2^{k-1}$ and:

- if $l_{i,j} = x_k$ is a positive literal: $\tau = T2^k, t_0 = 0, d = T2^{k-1}$,
- if $l_{i,j} = \neg x_k$ is a negative literal: $\tau = T2^k, t_0 = d = T2^{k-1}$.

In other words, spiked balls tied to literal x_k (resp. $\neg x_k$) block their respective path during periodic times $(0, \ldots, T2^{k-1} - 1 \mod T2^k)$ (resp. $(T2^{k-1}, \ldots, T2^k - 1 \mod T2^k)$).

We now piece everything together. Let us consider valuations $(x_n, \ldots, x_1)$ by lexicographic order. Given $0 \leq h \leq 2^n - 1$, let v_h be the h^{th} valuation by lexicographic order. Then, v_h matches with the binary representation of h from the right (e.g., the value of x_1 in v_h is the last digit). Knowing this, consider time units $Th, \ldots, T(h+1) - 1$. During them, by the definition of spiked balls $S_{i,j}$, the latter block their respective path if and only if:

- $l_{i,j} = x_k$ and the $(k-1)^{th}$ digit of h from the right is 0, or
- $l_{i,j} = \neg x_k$ and the $(k-1)^{th}$ digit of h from the right is 1.

Thus the missing spiked balls $S_{i,j}$ exactly correspond to the literals which are made true by valuation v_h. As a result, starting from time Th, we can reach the goal tile by time $T(h+1)$ if and only if v_h makes φ true. Finally, we have that φ is satisfiable if and only if, starting from base helirin state $\mathcal{S}_0$, there is a base helirin walk reaching the goal tile by time $D = T2^n$. ◀

co-NP-hardness

Proof. We reduce from 3-DNF-Tautology. Let $\varphi = \bigvee_{1 \leq i \leq m} (l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$ be a 3-DNF formula with m conjunctions and n variables, again w.l.o.g. with exactly three literals per conjunction. We encode the formula as an obstacle course going from left to right with parallel subpaths, one per conjunction in φ. Then, instead of reaching a goal tile, we allow the helirin to loop back to the start area. Figure 22 represents the base layout, which can be easily obtained with square wall tiles.

The plan is to dedicate time T for each of the 2^n valuations of the variables, this time forcing each one of them to loop through the structure, essentially choosing a subpath - i.e.,

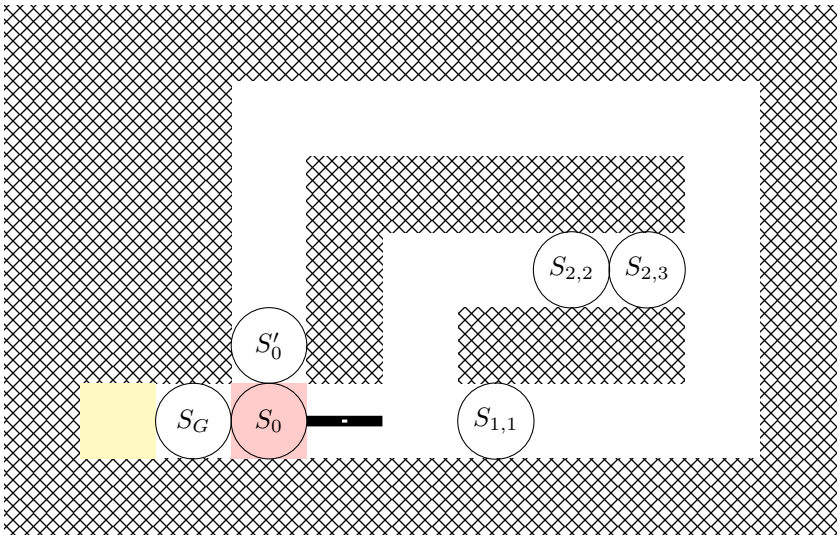

Figure 22 Layout of the co-NP-hardness reduction with spiked balls, illustrated with input formula $\varphi = (\neg x_1 \wedge \neg x_2 \wedge \neg x_4) \vee (x_1 \wedge x_3 \wedge x_4)$. We are at time $T+1$, i.e., about to check valuation $(x_4, x_3, x_2, x_1) = (0, 0, 0, 1)$, which makes φ false.

a conjunction in φ. First, we set $T = 4m + 6$, $D = T2^n + 2$ and set a spiked ball S_G at $(x, y) = (-2, 0)$ with $\tau = D, t_0 = 0$ and $d = D - 2$. Then, at the beginning of each period, we force the helirin to leave the start area and head to the right. We do so with two more spiked balls:

- S_0 at $(x, y) = (0, 0)$ with $\tau = T, t_0 = 1$ and $d = T - 1$,
- S'_0 at $(x, y) = (0, 2)$ with $\tau = T, t_0 = 0$ and $d = 2$.

Plus, we force the helirin to reach the end of our loop by the end of each time period. We do so by setting spiked balls for every couple $(i, j) \in \{0, \dots, 6\} \times \{0, \dots, 2m + 1\} \backslash \{(0, 0), (0, 1)\}$ at $(x, y) = (2i, 2j)$ with $\tau = T, t_0 = T - 1$ and $d = 2$. Then, other than time $D - 1$, the helirin is necessarily at positions $(1, 3), (1, 1), (3, 1)$ at respective periodic times $(T - 1, 0, 1 \mod T)$.

Now, for $1 \leq k \leq n$ we tie variable x_k to spiked balls of time period $\tau = T2^k$. More precisely, given $1 \leq i \leq m$ and $1 \leq j \leq 3$, we set a spiked ball $S_{i,j}$ at $x = 2j + 4$ and $y = 4i - 4$, with move duration $d = T2^{k-1}$ and:

- if $l_{i,j} = x_k$ is a positive literal: $\tau = T2^k, t_0 = 0, d = T2^{k-1}$,
- if $l_{i,j} = \neg x_k$ is a negative literal: $\tau = T2^k, t_0 = d = T2^{k-1}$.

Then, we have the same correspondence as in the previous proof between spiked ball appearances and valuations considered by lexicographic order. So, starting from time Th with $0 \leq h \leq 2^n - 1$, we can reach position $(1, 3)$ by time $T(h + 1)$ if and only if the h^{th} valuation v_h by lexicographic order makes φ true. Thus, we have that φ is a tautology if and only if, starting from base helirin state $\mathcal{S}_0$, there is a base helirin walk reaching the goal tile by time D. ◀

Finally, it is clear that both reductions also work if $\nu_{diag} = \nu_{card}$. Furthermore, the role of spiked balls with standstill base moves can be easily achieved using pistons with unit-time moves. Indeed, given such a spiked ball $(1, x, y, \tau, t_0, (\emptyset, d), 1)$, it blocks the square delimited by cells (x, y) and $(x + 2, y + 2)$ in periodic times $(t_0, \dots t_0 + d - 1 \mod \tau)$. So, e.g., we can define piston $(2, x, -2, \tau, t_0, t_0 + d, (N, 1), y + 2)$ to achieve the same result - i.e., when it is

inactive it is completely out of the way, and when it is active it blocks the square delimited by cells (x, y) and $(x + 2, y + 2)$.

A.6 Remark 27

Instead of using rotation angles and wall tiles to keep track of time like in Appendix A.2, similarly to the proofs of Theorem 26, we use spiked balls with standstill base moves to set intervals of time at which each edge in the underlying static graph is available. Furthermore, we use moving spiked balls along the representation of each edge in order to dictate exactly the travel time of the helirin along this edge. Not only does this allow us to encode interval temporal graphs, it can also be used to restrict the waiting time of the helirin at each vertex.