The complexity of snake and undirected NCL variants

The Complexity of Snake and Undirected NCL Variants

Marzio De Biasi

No affiliation

Tim Ophelders

Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands

Abstract

Snake and Nibbler are two well-known video games in which a snake slithers through a maze and grows as it collects food. During this process, the snake must avoid any collision with its tail. Various goals can be associated with these video games, such as avoiding the tail as long as possible, or collecting a certain amount of food, or reaching some target location. Unfortunately, like many other motion-planning problems, even very restricted variants are computationally intractable. In particular, we prove the NP-hardness of collecting all food on solid grid graphs; as well as its PSPACE-completeness on general grid graphs. Moreover, given an initial and a target configuration of the snake, moving from one configuration to the other is PSPACE-complete, even on grid graphs without food, or with an initially short snake.

Our results make use of the nondeterministic constraint logic framework by Hearn and Demaine, which has been used to analyze the computational complexity of many games and puzzles. We extend this framework for the analysis of puzzles whose initial state is chosen by the player.

Keywords: Games, Puzzles, Motion Planning, Nondeterministic Constraint Logic, PSPACE

1. Introduction

Recently the study of the complexity of puzzles and video games has gained a lot of popularity [2, 5, 13]. These puzzles are often based on motion planning problems. We will consider puzzles that can be modeled using paths or entities that move on planar graphs. A few such *motion planning* problems are the train marshaling problem [1], the robot and multi-robot path planning problems [7, 14], and the self-reconfiguring robot problem [8]. As a real-world example we can consider a set of linked wagons towed by a locomotor that must reach a

Email addresses: marziodebiasi@gmail.com (Marzio De Biasi), t.a.e.ophelders@tue.nl (Tim Ophelders)

target configuration by moving through a narrow environment. More geometric variants have also been studied, such as motion planning of deformable snake-like paths [6] in the Euclidean plane with obstacles. The problems we will consider arise from the popular games Snake and Nibbler.

Snake is a well-known video game with simple rules that dates back to 1978. It was inspired by the 1976 game Blockade. Since its original release, many variants of Snake have been created, implemented over a wide range of platforms. The simplicity of Snake has led to implementations for graphing calculators and cellphones in the late 90's. Despite its age, the popularity of Snake has hardly decreased, as new variants still appear to this day. A variant we focus our analysis on in this paper is the 1982 arcade game Nibbler.

The objective of Nibbler is to collect a set of items (food), placed on vertices of a graph, by maneuvering a simple path (a snake) through that graph. This path grows by a constant number of vertices per collected item, and the path can move only by extending the front of the path (the head) or removing vertices from the end of the path (the tail). So once a vertex is part of the snake, it is removed only after all vertices towards the tail have been removed. As a result, such vertices may trap the head of the snake, preventing it from reaching particular items. The challenge of Nibbler is to route the head without trapping it before collecting all food. We define moves between states of Nibbler in Definition 1, and the food collection problem in Definition 2. In our analysis of this problem, the growth rate g, the initial length $|P|$ of the snake, and the amount of food $|F|$ are treated as parameters.

Definition 1 (Valid moves between Nibbler states)**.** Consider a graph G and a *growth rate* $g \in \mathbb{N}$. A *snake* is a sequence P of vertices forming a simple path in G. A Nibbler *state* (P, F, d) is a snake P, a set $F \subseteq V_G$ of *food* vertices, and an integer $d \geq |P|$ representing the target length of the snake, see also Figure 1. We denote $s_1 + s_2$ the concatenation of two sequences s_1, s_2 of vertices. A *move* $(P, F, d) \to (P', F', d')$ between Nibbler states is *valid* if and only if $F' = F \setminus P'$, and $d' = d + g \cdot |F \setminus F'|$, and $t + P' = P + h$, with $|h| = 1$ and $|t| = 1$ unless $|P| < d'$, in which case $|t| = 0$. Here, h and t capture the movement of the snake's head and tail.

Definition 2 (Nibbler food collection problem (NIBBLER))**.**

Input. A graph G, growth rate g, and Nibbler state (P, F, d) with $F \cap P = \emptyset$ and $d = |P|$.

Output. Is there a sequence of valid moves that reaches a state (P', F', d') with $|F'| = 0$?

Figure 1 illustrates valid moves (or the absence thereof) for small instances of Nibbler. Generally, Nibbler takes place on a rectangular grid, possibly containing walls. We distinguish the variant without walls (solid grid graphs) from the one with walls (grid graphs), and discuss them separately in Sections 2

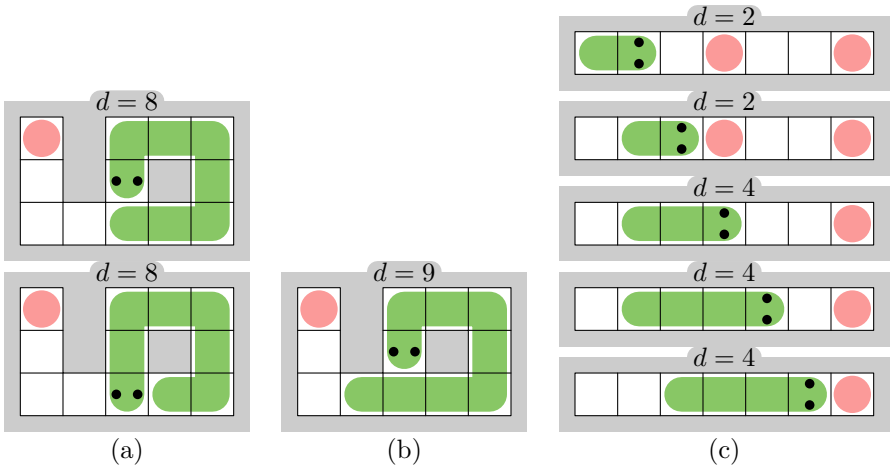

Figure 1: (a) A valid move. (b) A snake that cannot move. (c) Collecting food with $g = 2$.

and 4. For the latter, we use reductions from PSPACE-complete problems introduced in Section 3. These problems may be of independent interest for proving PSPACE-hardness of puzzles in which no initial position can be enforced.
A *grid graph* is a finite node-induced subgraph of the infinite two-dimensional integer grid, see Definition 3. The Hamiltonian cycle and path problems are NP-complete even when restricted to grid graphs [9]. A *solid grid graph* is a grid graph without holes. Formally, all points $p \in \mathbb{Z} \times \mathbb{Z}$ that are not vertices of a solid grid graph lie in its outer face. The Hamiltonian cycle problem on solid grid graphs is solvable in polynomial time [11]. In contrast, the complexity of the Hamiltonian path problem on solid grid graphs is still open.

Definition 3 (Grid graph)**.** A finite undirected graph (V, E) with $V \subseteq \mathbb{Z} \times \mathbb{Z}$ and $(u, v) \in E$ if and only if $\|u-v\| = 1$. So all edges are of the form $\{(x, y), (x+1, y)\}$ or $\{(x, y), (x, y+1)\}$.

An *orthogonal grid embedding* of a planar graph is a drawing in which each vertex is a distinct vertex of a grid graph, and each edge is represented as a path of edges of that grid graph. Given a planar graph whose vertices have degree at most 4, the algorithm of Tamassia [10] computes an orthogonal grid embedding with area $O(n^2)$ in polynomial time.

2. Nibbler without walls

In Theorem 2.1, we show that it is NP-hard to decide whether a snake with growth rate $g \geq 1$, moving on a solid grid graph can consume all food.

Theorem 2.1. NIBBLER *on solid grid graphs is NP-hard for any constant growth rate* $g \geq 1$.

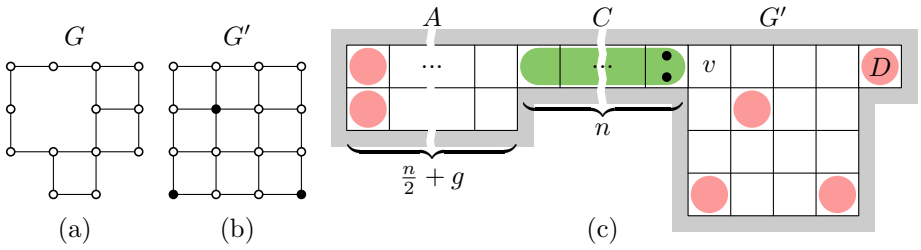

Figure 2: An instance of NIBBLER (c) derived from a Hamiltonian cycle problem instance (a).

Proof. We reduce from the NP-complete Hamiltonian cycle problem on grid graphs with holes [9]. Let the graph G be an instance of this problem. We include G as a subgraph of a rectangular grid graph G', see Figure 2 (a) and (b).

The two leftmost vertices in the topmost row of G must be part of any Hamiltonian cycle (if there is only one vertex, then there is no Hamiltonian cycle), so we can attach a path (that is two vertices wide) on top of these vertices, and route it to the top left corner of G'. The resulting graph has a Hamiltonian cycle if and only if G has one. Hence, without loss of generality, we assume that the two topmost vertices in the leftmost column of G' are vertices of G and denote the topmost one by v. We may also assume that the rectangular grid graph G' has even width. We place food on all vertices of G' except those of G, and attach an extra vertex D with food to the right of the top-right corner of G. Because D is a dead end, the snake must collect the food in D last.

Let n be the (even) number of vertices of G, and attach a path C containing an initial snake P of length n to the left of v, such that the head of P faces v, see Figure 2 (c). On the opposite end of path C, we attach a rectangular area A of height 2 and width $n/2 + g > 2g$ with two vertices with food in the leftmost column of A.

The snake P is forced to enter G', and it must find a way to turn around in order to reach the food in A. If the snake consumes any food in G' before consuming the food in A, the snake will be trapped in A, and hence unable to consume the food in D. The reason the snake gets trapped is that the snake will be of length greater than n when entering A, and after eating the food in A, the tail of P will still be be blocking the exit of A when the head reaches the exit. Hence, the snake can reach A with length at most n if and only if it uses a Hamiltonian cycle in G. If and only if it reaches A with length at most n, it can consume the food in A and return to G'. It can then consume all remaining food in G' using a zig-zag motion (going up and down in columns of G' from left to right). Eventually, the head of the snake will be able to consume the food in D after consuming all other food. An analogous argument shows that if $g \geq 2$, a snake of (odd) length $n-1$ must use a Hamiltonian cycle to exit G' to consume all food. The case where $g = 1$ and the snake has odd length is handled using 3 food vertices in A. Hence, NIBBLER is NP-hard on solid grid-graphs for $g \geq 1$. □

Although the above construction shows hardness for snakes of arbitrarily large initial (even) length $|P| = n$ and odd length $|P| = n - 1$, the construction extends to a setting in which the initial snake is short ($|P| \geq 3$). For this, the short snake is placed at the start of path C, and we place $\lfloor (n - |P|)/g \rfloor$ pieces of food in front of the snake, which the snake is then forced to consume. This grows the snake to length n or $n-1$, and inevitably results in the initial position of Theorem 2.1, so Corollary 2.1 follows.

Corollary 2.1. NIBBLER *on solid grid graphs is NP-hard for any* $|P| \geq 3$ *and any constant growth rate* $g \geq 1$.

A *rectangular grid graph* is a (solid) grid graph whose vertex set is a complete rectangle $[1, \ldots, w] \times [1, \ldots, h]$. A second extension shows that consuming all food on a rectangular solid grid graphs is NP-hard for $g \geq 2$.

Theorem 2.2. *It is NP-hard to decide if a snake with growth rate* $g \geq 2$ *can consume all food on a rectangular grid graph.*

Proof. As in Theorem 2.1, let grid graph G be an instance of the Hamiltonian cycle problem. Let G be a subgraph of a rectangular grid graph G' of width w and height h, with food on all vertices except those of G. We denote to the two topmost vertices of the leftmost row of G' as v_1 and v_n, and assume without loss of generality they are vertices of G, and thus empty. Let n be the number of vertices in G. If G has a Hamiltonian cycle, then it has a Hamiltonian cycle that starts at v_1, and v_n is visited last before returning to v_1.

We lay out the initial snake P of length $3n + 2h$ in a spiral of height h and width $n + 2$ next to G', such that its head lies next to v_1 and its last $n + 1$ vertices lie next to v_n; so if the snake does not consume food during the first n moves, the tail will lie next to v_n. Finally, we place a single food in the top-left corner of an area A of height h and width $3n + 2h$ that is placed to the left of the snake, see Figure 3.

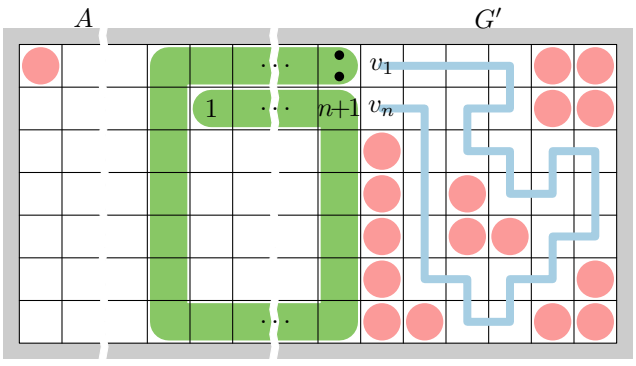

Figure 3: Snake on rectangular grid graphs.

We claim that if G' does not contain a Hamiltonian cycle, the snake cannot exit G'. Indeed, the first opportunity for the snake to exit occurs next to v_n,

and only after $n+g\cdot k \geq n+2k$ moves, where k is the number of items consumed. Hence, to exit G', the snake must occupy at least $n+2k$ vertices of G', which it can only do if $k=0$ since there will only be $n+k$ vertices without food in G'. So to consume the food in A, the snake head must be on v_n after n moves without consuming food. This is possible only if G' contains a Hamiltonian cycle, which we may assume is $\langle v_1, \ldots, v_n, v_1\rangle$. In that case, the snake can follow this cycle up to v_n; chase its tail until it can enter area A and consume the food contained in it and finally re-enter G' to collect the remaining food using a zig-zag motion as in Theorem 2.1. In the opposite direction, if G has a Hamiltonian cycle, then the snake can follow the same pattern described above to consume all food. □

Corollary 2.2 follows from extending area A and placing more food in it.

Corollary 2.2. *For any positive constant fraction, it is NP-hard to decide if a snake with growth rate $g \geq 2$ can consume at least that fraction of food on a rectangular grid graph.*

3. Nondeterministic Constraint Logic

Nondeterministic constraint logic (NCL) is a framework by Hearn and Demaine [4] for proving the complexity of reconfiguration problems. An *NCL graph* is a graph whose edges all have weight 1 or 2 and all vertices have an *inflow constraint*, which is either 1 or 2. When drawing an NCL graph, we color the (thin) edges of weight 1 red, and the (fat) edges of weight 2 blue. We will use two types of vertices with an inflow constraint of 2, namely AND and OR vertices. An AND *vertex* has two red edges (of weight 1), and one blue edge (weight 2), whereas an OR *vertex* has three blue edges, see Figure 4. An *oriented* NCL graph is a directed version of an NCL graph.

The *inflow* of a vertex is the sum of weights of inward directed edges, and an oriented NCL graph is *valid* if and only if each vertex has as inflow at least its inflow constraint. So for AND vertices, the blue edge can be directed outward only if both red edges are directed inward. For OR vertices, at least one edge is directed inward. We say that an NCL graph is in *normal form* if it uses only AND and OR vertices.

(a) AND (b) OR

Figure 4: The two vertex types of NCL graphs in normal form.

A *move* is an operation on a valid NCL graph that reverses the direction of one edge and a move is *valid* if it results in a valid NCL graph. Since reversing an edge twice does not change the graph, any valid move can be executed twice in succession to return to the original graph. Define the *configuration graph* of an

NCL graph to be the graph of valid oriented NCL graphs, with an edge between two graphs if and only if a valid move between them exists; that is, if exactly one edge direction is different.

Given an initial NCL graph, the problem of reversing the direction of a target edge through a sequence of valid moves (see Definition 4) is PSPACE-complete as proven by Hearn and Demaine [4] using a reduction from the quantified Boolean formula problem (QBF); the result holds even if the NCL graph is planar (see Theorem 3.1).

Definition 4 (NCL edge reversal problem: NCLREV(G, e^*))**.**

Input. A valid NCL graph G with orientation o and an edge $e^* \in E_G$.

Output. Is there a sequence of valid moves, starting from o, that eventually reverses edge e^*?

Theorem 3.1 ([4])**.** NCLREV *is PSPACE-complete, even for planar graphs in normal form.*

We generalize Theorem 3.1 to a setting where the edge directions of the initial graph can be chosen arbitrarily. In that case, finding an initial graph from which a given edge can be reversed is only NP-complete by reduction from the Boolean satisfiability problem (SAT). In contrast, we prove that reversing two edges in this setting (Definition 5) is PSPACE-complete in Theorem 3.2. We remark that this theorem was already known to hold for the instances resulting from the reduction from QBF [3]. In contrast, our alternative construction makes no assumptions on the NCL graphs used, and can hence generalize to other reductions, such as NCL graphs whose bandwidth is bounded by a constant, for which it was recently shown [12] that the NCL problem remains PSPACE-complete. Additionally, in Theorem 3.3, we show that reversing *all* edges at least once (Definition 6) is also PSPACE-complete for NCL graphs in normal form. Because any initial configuration is of polynomial size, and can hence be guessed, this result immediately reduces to the analogous case where the initial graph is fixed (Corollary 3.1).

Definition 5 (Free NCL edge reversal problem: FREENCLREV(G, e^*, f^*))**.**

Input. An NCL graph G without orientation and two edges $e^*, f^* \in E_G$.

Output. Does there exist a valid initial orientation for the edges of G for which a sequence of valid moves reverses both e^* and f^* at least once? Note that we do not require a configuration in which both edges are reversed simultaneously.

Definition 6 (Free NCL complete reversal problem: FREENCLREVALL(G))**.**

Input. An NCL graph G without orientation.

Output. Does there exist a valid initial orientation for the edges of G for which a sequence of valid moves reverses all edges of G at least once?

Theorem 3.2. FreeNclRev *is PSPACE-complete, even for planar graphs in normal form.*

Theorem 3.3. FreeNclRevAll *is PSPACE-complete, even for planar graphs in normal form.*

Corollary 3.1. FreeNclRevAll *is PSPACE-complete, even for planar graphs in normal form whose initial orientation is fixed.*

Before we prove Theorem 3.2, we construct gadgets that enforce the directions of a subset of edges in an NCL graph if a given edge is directed outward with respect to such gadget. We use these gadgets to enforce the initial edge directions of NclRev whenever a specific edge f^* is directed outward. Because the configuration graph has strongly connected components only, it is then PSPACE-complete to reverse both f^* and e^*.

3.1. Enforcing edge directions

Given an instance (G, e^*) of NclRev with NCL graph G in normal form with orientation o, we derive an instance (H, e^*, f^*) of FreeNclRev that has a solution if and only if the instance to NclRev has a solution. For any edge e of G, we create a copy $\phi(e)$ of that edge in H using the transformation of Figure 5. If $e \in E'$ is red (a), we first transform it into a blue edge (b), after which we transform that blue edge (c) into $\phi(e)$ (d). In the resulting graph, we have an edge f_e for each edge e of the original graph. If f_e is directed outward, then the direction of $\phi(e)$ must correspond to the direction of e in orientation o of G. If on the other hand f_e is directed inward, then $\phi(e)$ behaves exactly as e does in G. We connect all f_e to a single edge f^* using $|E'| - 1$ Branch gadgets (see Figure 6), such that if f^* is directed outward (to the right in the figure), all f_e are directed outward; and if f^* is directed inward (to the left in the figure), then all f_e can be directed inward as well.

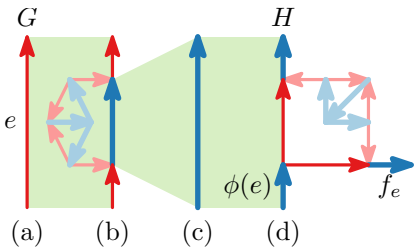

Figure 5: An edge e of weight 1 (a) or weight 2 (c) is transformed into edge $\phi(e)$ whose direction is fixed if edge f_e is directed outward (d). Light edges ensure that H is in normal form.

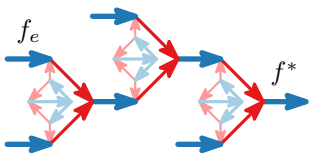

Figure 6: Three Branch gadgets can be used to connect four edges to f^*.

We ensure that the left endpoint of f^* can always be directed inward by connecting it to a so-called *free edge terminator* gadget. Planarity is ensured using

crossover gadgets. Both the crossover and the free edge terminator gadgets were introduced by Hearn and Demaine [4]. These gadgets ensure that H is planar and uses only AND and OR vertices, without changing the behavior of the NCL graph, or the edges representing $\phi(e^*)$ or f^*. The behavior of these gadgets is independent of their initial configurations.

Proof of Theorem 3.2. We use a reduction from the PSPACE-complete NCLREV to prove that reversing two edges is PSPACE-complete if the initial orientation of an NCL graph can be chosen arbitrarily. Consider an NCL graph G with initial orientation o and edge e^* given by an instance of NCLREV. Construct the graph H as described above with edge f^* forcing for each e of G the direction of $\phi(e)$ of H to be that of e as given by o. We show that it is PSPACE-hard to reverse both f^* and $\phi(e^*)$ from an arbitrary initial orientation of H. Indeed, if f^* is reversed at some point, then the orientation of H at some point corresponds to the orientation of o for G. The sequence of moves connecting this orientation in H and the reversal of $\phi(e^*)$ corresponds to a sequence of valid moves in G that reverse e^* in G and vice-versa. Hence, deciding whether two edges can be reversed from an arbitrary initial orientation is PSPACE-hard. The problem is clearly in NPSPACE, and since PSPACE = NPSPACE by Savitch's theorem, FREENCLREV is PSPACE-complete. □

Remark 1. The problem FREENCLREV is related to the PSPACE-complete EDGE-TO-EDGE (E2E) problem defined in [3]: "Given two edges e_A and e_B of an NCL machine, and orientations for each, are there configurations A and B such that e_A has its desired orientation in A, e_B has its desired orientation in B, and there is a sequence of moves from A to B?". These problems are related in the sense that any sequence of moves that reverses both e_A and e_B must pass through a configuration in which e_A has the desired orientation, and one where e_B has the desired orientation. A subtle difference between these problems is that there could exist a configuration where e_A and e_B simultaneously have the desired orientations, but those edges can never be reversed. However, the reduction of [3] can be adapted to show that FREENCLREV is PSPACE-complete, since the reduction prevents e_A and e_B from having the desired orientations simultaneously. In contrast, using our reduction we can show that FREENCLREV remains PSPACE-complete for graphs of bounded bandwidth.

3.2. Relaxing edge directions

From now on, we base our constructions on the graph H constructed above, and refer to $\phi(e^*)$ simply as e^*. We use a second construction to allow arbitrary orientations for certain subsets of edges if a given edge is directed inward, and use this to prove Theorem 3.3.

The main tool used in the construction is the RELAX gadget. The RELAX gadget comes in two types, red and blue, shown in Figure 7 (a) and (b), respectively. Each RELAX gadget has an input and an output edge, and these are said to be *disabled* if they are directed as in Figure 7 (a) or (b), and *enabled* if

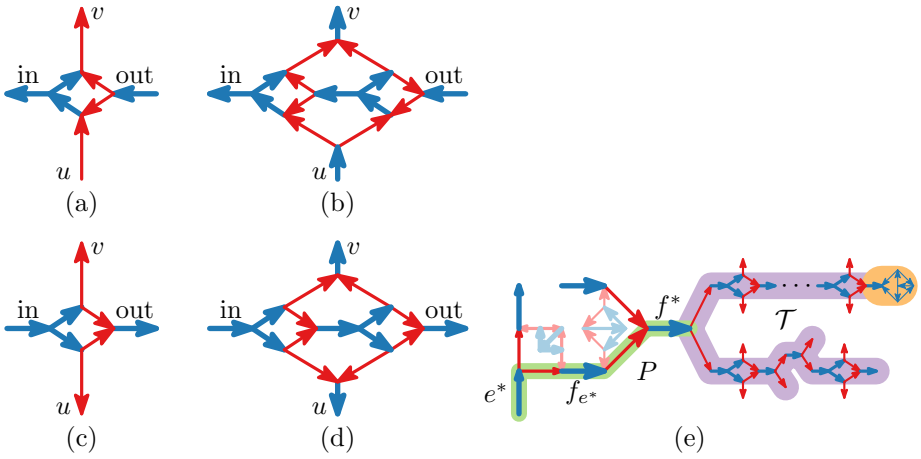

Figure 7: A (a) red and (b) blue RELAX gadget with its input and output disabled (a) and (b) or enabled (c) and (d). (e) Connecting f^* to a tree of RELAX gadgets using AND vertices to connect outputs to inputs of children.

they are directed as in Figure 7 (c) or (d). Any edge between u and v can be replaced by a RELAX gadget (of matching color) between u and v.

If the input of a RELAX gadget is disabled, edges of the gadget cannot direct towards u and v simultaneously, and its output is also disabled. So the RELAX gadget acts like a normal edge with respect to u and v if its input is disabled. On the other hand, if it is enabled, the output can be enabled and edges of the RELAX gadget can direct into both u and v simultaneously. Multiple RELAX gadgets can be connected to form a tree, illustrated in purple in Figure 7 (e). Inputs of gadgets in the tree can all be enabled if (and only if) the first is enabled. We connect the outputs of leaves of this tree to free edge terminators (highlighted in orange), such that these outputs can always be put in a disabled state.

Consider the graph H obtained from G in the construction for Theorem 3.2. We may assume that e^* and f^* lie on the outer face of the planar graph, and that e^* is connected to the BRANCH gadget closest to f^*, as illustrated in Figure 7 (e). We construct a graph H' by replacing the free edge terminator gadget connected to f^* in H by a tree $\mathcal{T}$ of RELAX gadgets, such that only if f^* is directed into $\mathcal{T}$, inputs of RELAX gadgets in $\mathcal{T}$ can be enabled. Let $\mathcal{T}$ denote the tree of RELAX gadgets including the free edge terminators at its leaves. Let P be the path on BRANCH gadgets from f^* to e^* (highlighted in green). We will replace each edge of H', except those of P and $\mathcal{T}$, by three RELAX gadgets of $\mathcal{T}$ (see Figure 8 (b)). We route the tree $\mathcal{T}$ such that planarity is preserved. To do this, we construct an ordinary tree T that acts as a backbone of the tree $\mathcal{T}$ of RELAX gadgets. Take any spanning subtree T of the dual graph of a planar drawing of $H' \setminus \mathcal{T}$, such that T does not use any (dual) edges of P. Observe that such a tree exists since P is a strict subpath of the boundary of one face.

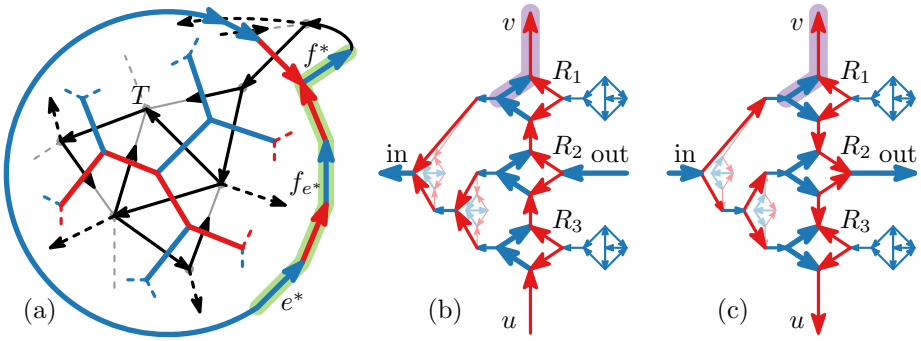

Figure 8: Schematically, tree T in black (a). A black arrow of (a) that crosses an edge (u, v) represents three RELAX gadgets of $\mathcal{T}$, shown in detail (for the case of red RELAX gadgets) in (b) and (c). The case of blue RELAX gadgets is implemented analogously, but is not illustrated.

For each edge of $H' \setminus (P \cup \mathcal{T})$ that is not yet crossed by T, add an edge from either of the two incident vertices (of the dual graph) to a new leaf to T, so that the tree T is planar and crosses exactly the edges of $H' \setminus (P \cup \mathcal{T})$. Let the root of T be in the (primal) face adjacent to (both sides of) f^*, such that f^* can be connected to the root of T; see Figure 8 (a). To obtain $\mathcal{T}$ from T, consider an edge t of T that crosses an edge (u, v) of H'. Replace the edge (u, v) by three RELAX gadgets, and combine their inputs into one using mirrored BRANCH gadgets, see Figure 8 (c). The output of the middle RELAX gadget along (u, v) is used to reach descendants of edge t of T (using AND vertices if the target of t has multiple children). The other two RELAX gadgets (and also the middle one if t ends in a leaf) are leaves of $\mathcal{T}$, so their outputs are connected to free edge terminators. Call the resulting graph H''.

Proof of Theorem 3.3. We claim that all edges of H'' can be reversed if and only if e^* and f^* can be reversed in H. If not both e^* and f^* can be reversed in H, then e^* and f^* in H'' can also not both be reversed in H'' because they are directly connected through P in H'', and no edge of P was replaced by a RELAX gadget. To prove FREENCLREVALL is PSPACE-complete, we show that if both e^* and f^* can be reversed, so can all edges of H''.

Consider a sequence of moves in H that reverses both e^* and f^*, and apply the equivalent sequence to H''. This reverses e^* and f^* and therefore all edges of P in H''. Because all edges of H'' except those of P lie on $\mathcal{T}$, it suffices to show that all edges of $\mathcal{T}$ can be reversed. Because f^* was reversed, the first input of $\mathcal{T}$ can be reversed and hence enabled.

We show how the first edges of $\mathcal{T}$ can be reversed, and how they allow descendants to be reversed. For this, consider Figure 8 (b) and (c), which show in detail how an edge (u, v) was replaced in (a). Assume without loss of generality that the edges are directed as in (b), and that the leftmost edge can be reversed. In that case, we can reverse all edges but the three highlighted ones, and then direct the edges as in (c), such that the output of RELAX gadget R_2

is enabled, and the descendants in $\mathcal{T}$ can be reversed in a similar way[1].

Doing so, we have for all of the edges (u, v) of H that were replaced in H'', that all but the highlighted edges near the target vertex (without loss of generality, v) have reversed in H''. If v does not lie on P, then it is incident to only RELAX gadgets, that are now all directed into v, so v has sufficient inflow to reverse the highlighted edges in incident RELAX gadgets and return them to state (c). If on the other hand, v lies on P, then the blue edge incident to v has reversed at some point, so the highlighted edges could also be reversed. □

Remark 2. Theorems 3.2 and 3.3 remain true for graphs with bandwidth bounded by a constant. To see this, observe that each edge of G is locally replaced by gadgets (transformed edges and crossovers) of constant size in H. Additionally, each BRANCH gadget can be placed near a distinct such transformed edge, and BRANCH gadgets can be connected in a linear fashion, so that the bandwidth of H grows by at most a constant factor with respect to that of G.

4. Nibbler with walls

On (non-solid) grid graphs, we prove that several variants of NIBBLER are PSPACE-hard using reductions from the problems of Section 3. Let G_{NCL} be a planar NCL graph (of n vertices) that we reduce from, and consider an orthogonal grid embedding G'_{NCL} of G_{NCL}, so that each edge of G_{NCL} is represented by a path of edges in G'_{NCL}. We may assume that this embedding uses only $O(n^2)$ edges [10]. Moreover, after a constant factor of scaling, we ensure that the two red edges incident to any AND vertex in G_{NCL} meet at 180 degrees.

Scaling the embedding further by some factor w, all faces in the resulting embedding contain $\Omega(w^2)$ grid points, and the resulting embedding uses only $O(wn^2)$ edges. If we take w to be sufficiently large, say $w = \Omega(n^4)$, then we can ensure that all paths (representing edges of G_{NCL}) between adjacent AND and OR vertices of G_{NCL} differ in length by at most a constant by introducing detours on short paths. The resulting embedding uses $O(wn^3)$ edges, and the number of grid points in each face remains $\Omega(w^2)$.

We will use this embedding of G_{NCL} to obtain an instance G of NIBBLER as follows. First, we scale G'_{NCL} by a constant factor and replace all paths (edges of G_{NCL}) by *edge* gadgets, see Figure 9 (a). Each edge gadget consists of two *half-edges* (separated by a dashed line, about halfway along the length of the edge). Each half-edge contains two bundles of two or three (blue) paths. Separating the two bundles of a half-edge, there is a *connector port pair* (orange). The bundles of blue paths are routed along the paths given by the embedding (b), in such a way that the blue paths in the resulting instance differs in length by at most a constant.

[1]Note that if we had only replaced edges by a single RELAX gadget instead of three,

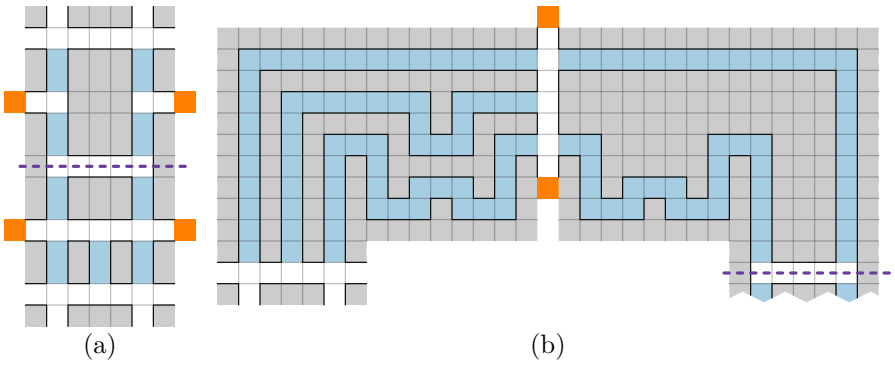

Figure 9: (a) Two half-edges of an edge gadget. (b) Routing blue paths around corners.

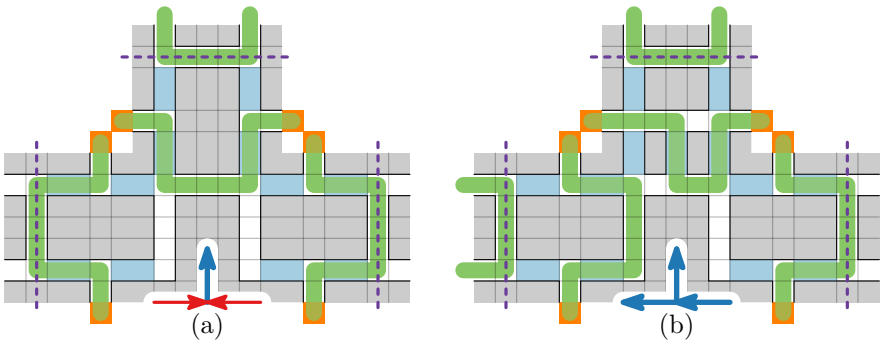

Figure 10: AND (a) and OR (b) gadgets in NIBBLER, separated by dashed lines.

Figure 10 shows AND and OR gadgets of G, corresponding to vertices of G_{NCL}, and illustrates how these gadgets connect half-edges in G at these vertices. The figure also shows green paths spanning the connector port pair of each half-edge.

We denote the graph of half-edges and AND and OR gadgets as $\mathcal{G}$. In addition to $\mathcal{G}$, the graph G contains so-called *connector paths* (which we place in the faces of G'_{NCL}). These connector paths are long compared to the size of $\mathcal{G}$, and each of them connects two (orange) connector ports. In our constructions, the key idea is to lay out the snake P (green) to form a long cycle alternating between all the half-edges and connector paths of G, forcing the head of the snake to chase its tail in order to not get stuck. The snake's head may reroute the segments of the cycle inside the two half-edges of an edge gadget to reverse edges.

descendants would not always be able to reverse since we may need the edges of u and v to be reversible for descendants. Using three RELAX gadgets allows the outputs of the R_1 and R_3 to be disabled during this propagation.

We place the connector paths in such a way that they, together with the segments of P inside the half-edges, form a simple closed path. By Lemma 4.1, a simple closed path in the plane exists that crosses each path of G'_{NCL} twice. This path can be drawn on the grid such that each crossing coincides with the connector port pair of a half-edge of $\mathcal{G}$, and the connector paths are routed along segments between two such crossings, see Figure 11. We introduce detours on connector paths, so that they all have length $\Omega(\frac{w^2}{wn^3}) = \Omega(\frac{w}{n^3})$, where w is the scaling factor of G'_{NCL}. We take $w = \Theta(n^4)$, so that all connector paths can be made significantly longer than the size of $\mathcal{G}$.

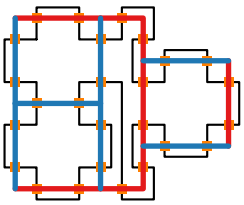

Figure 11: Black connector paths.

Lemma 4.1. *For a planar embedding of a graph G, and a subset E' of its edges that are connected in its dual graph, there exists a simple closed path in the plane that intersects the edges of E' exactly twice, and no other edges.*

Proof. By induction on $|E'|$. The case where E' has at most one edge is trivial. In the case where $|E'| > 1$, there is an edge $e \in E'$ for which $E' \setminus \{e\}$ is connected in the dual graph. So by hypothesis, a simple closed curve intersects all edges of E' except e twice. This curve has a segment in a face adjacent to e, so we can deform the curve to also cross e twice. □

We are now ready to consider the problem of collecting two items ($|F| = 2$) with an initially long snake. Assume the growth rate g is any constant. Let S_P be the set of segments of P in $\mathcal{G}$. Call a path P *stable* if its head and tail lie outside $\mathcal{G}$, so all segments of S_P connect two connector ports. If the head and tail of a stable path P lie on the same connector path C, and $|S_P| \geq 1$, define the *head-tail distance* of P as the number of vertices in $C \setminus P$. Let h be the number of half-edges in $\mathcal{G}$. We call a stable path P *valid* if S_P contains h segments, each of which connects two connector ports of a single half-edge and each of which contains exactly two blue paths.

Recall that all blue paths differ in length by at most a constant, and denote by W the length of the shortest blue path, and let the longest blue path have length $W + c$ for some constant c. We derive a valid initial path P from the NCL instance, and ensure that the head-tail distance is $D = g \cdot |F| + b \cdot c$, where b is the number of blue paths in $\mathcal{G}$. The head-tail distance for any valid path reachable from P will then be between 0 and $D + b \cdot c \leq 2D$.

Observe that for some segment $s \in S_P$ of a valid path P, any connected component of $(\mathcal{G} \setminus S_P) \cup s$ consists of at most $R = 11(W + c) + 25$ vertices. We will ensure that connector paths are all longer than $R + 2D$ vertices, so if the head of P moves into a half-edge (through a connector port), then P can be reconfigured into stable a path only by reaching the other connector port of the same half-edge. If it does so by moving through fewer than two blue paths, then it cannot use any blue paths, so the head-tail distance will have decreased by at least $2W$ when the head reaches the other connector port.

We will ensure that $2W > 2D$, so the snake cannot reach a stable path before using at least two blue paths, since the head-tail distance cannot be negative. Observe that there are no simple paths (connecting the two connector ports at the ends of $s \in S$) that use more than two blue paths in $(\mathcal{G} \setminus S_P) \cup s$. Hence, for any stable path P' reachable from a valid path P, the segments of $S_{P'}$ all use exactly two blue paths. So we conclude that all such P' are valid based on the following assumptions, both of which can be satisfied by our choice of $w = \Theta(n^4)$, which scales G'_{NCL} sufficiently.

- The initial valid path P has a head-tail distance of $D = g \cdot F + b \cdot c < W$.
- Each connector path is longer than $R + 2D = 11(W + c) + 25 + 2D < 13W + 11c + 25$.

We refer back to Figure 10 and argue that all moves leading from one valid path to another correspond to valid edge reversals of the NCL graph. The direction of an edge of the NCL graph is encoded in P by the routes taken by the segments of S_P in the corresponding half-edges. This encoding is as follows: whenever an edge of the NCL graph is directed into a vertex, the corresponding segment is routed away from the corresponding vertex in G, and vice-versa. It suffices to observe that for an OR gadget of G, the only constraint is that the segment of one of the incident half-edges is always routed away from the gadget. For an AND gadget, the segments of both red half-edges must be routed away from the gadget whenever the blue half-edge is not routed away from it (as in Figure 10). Lemma 4.2 follows.

Lemma 4.2. *It is PSPACE-complete to reach a valid path in which the segment of a half-edge (of the target edge of the NCL graph) is rerouted from a valid initial path in* NIBBLER.

Corollary 4.1. *Reaching a path P' from P is PSPACE-hard on grid graphs, even if $|F| = 0$.*

Using this lemma, we show that collecting two items is PSPACE-complete. Place one item in a blue path of the target half-edge, such that this item is collected when the segment of the half-edge changes its route. We use the second item to verify that a valid position was reached after collecting the first item. To do so, we place it in a dead end branching from a connector path, so that the second item must be collected last. Then both items can be collected if and only if a valid position is reached after reversing the target edge. Theorem 4.1 follows, but requires the initial snake to be long.

Theorem 4.1. NIBBLER *is PSPACE-complete for any constant growth rate $g \geq 0$ if $|F| = 2$.*

We wish to extend this theorem to a setting in which the initial snake is short, say $|P| = 1$, and the growth rate is positive. The idea is to use a lot of food (items) to grow the snake, and force it into a valid position, from which it must

collect remaining food. As before, an item in a dead end branching from a connector path must be the final item consumed in any solution to NIBBLER.

Let C and $C + c'$ (for some constant c') be the lengths of the shortest and longest connector paths, respectively, and assume without loss of generality that C is greater than $4|\mathcal{G}|$. We place the initial snake in a long path (with one dead end) that leads to the middle of a connector path, and fill this long path with $(h - \frac{1}{4})(C + c')/g$ food items (where h is the number of connector paths). When the tail of the snake leaves this path, the snake P' has length $(h - \frac{1}{4})(C + c')$. The snake must use part of each connector path, since otherwise more than $|\mathcal{G}|$ vertices of P' would not lie in a connector path, which is impossible. Moreover, the head and tail of P' lie in the same connector path.

During the first traversal, whenever its head leaves a connector path C_i, the snake is forced to choose a *bridge path* between the exit endpoint p_i of C_i and another endpoint p_{i+1} of the connector path C_{i+1} that it will traverse next. But the distance between the head and the tail will never be greater than C; so after the first tour the snake will not be able to change the endpoints of the bridge-paths, indeed when the heads leaves corridor C_i, its tail will stay in corridor C_{i+1} for more than $|\mathcal{G}|$ steps, so all endpoints, except the previously choosen one p_{i+1}, will remain blocked. In other words it will be able to modify the bridge path, but the sequence of the traversed corridor paths will remain fixed.

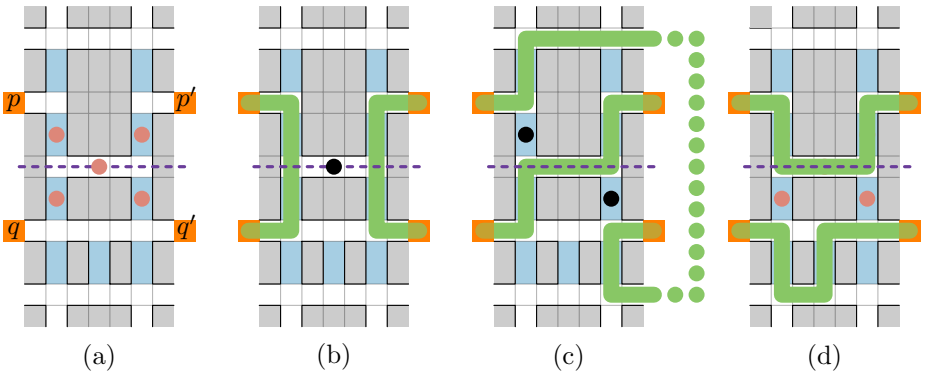

Figure 12: (a) Placing food in half-edges. (b-c) Invalid connections. (d) Valid connections.

For each edge we place a food item in the middle corridor and in the four blue corridors surrounding it, as illustrated in Figure 12 (a). The snake must use an average of at least two blue corridors per bridge path (otherwise it has not enough space and collides with itself). If some bridge path uses more than two blue corridors, then this would block a connector port, so all bridge paths use exactly two blue corridors.

We argue that if a bridge path connects connector ports of different edges, then some food cannot be consumed. Indeed, if food in the blue corridors adjacent to those connector ports is consumed, the bridge path that collects

the food must use at least three blue corridors in order to connect its connector ports. Hence, each bridge path connects connector ports of the same edge.

Moreover, we show that for each edge, a bridge path must connect the endpoints of the connector paths of the same half edge. For instance, in Figure 12, this would mean that a bridge path connects p and p', and a bridge path connects q and q'. We distinguish two cases and reason that it would be impossible to collect the food in the inner part of the half edges otherwise. Suppose p connects to q, as in Figure 12 (b). Then the food on the dashed line cannot be consumed without getting stuck. If instead p connects to q', as in Figure 12 (c), then q must connect to p', and (by planarity) one of the connections must use the dashed line. Now, to consume the remaining food, this connection must reconnect in such a way that the dashed line is not used, which would require the two connections to cross. Hence, for each edge, p must be connected to p', and q to q', meaning we have only valid connections, see for instance Figure 12 (d).

Now it is clear that, since the two bridge paths between p and p' and between q and q' cannot occupy the central (dashed) corridor at the same time, collecting all food in some edge gadget requires it to be reversed. In fact, if all edges are reversed, all food is automatically consumed. So all food can be consumed only if P' is a valid path and if and only if there is a solution to FreeNclRevAll.

Theorem 4.2. Nibbler *is PSPACE-complete for any growth rate $g \geq 1$, even if $|P| = 1$.*

5. Snake with walls

Snake is a variant of Nibbler in which not all food is initially on the board. We consider a variant where there is at most one item on the board at any given time. Initially one item is on the board, and the next item spawns only whenever the previous item was consumed. Here, a degeneracy occurs when a new item spawns on a location currently occupied by the snake. For simplicity, we will assume this item is consumed immediately in such case.

To model Snake, we pair the food vertices F with a permutation, representing the order in which items spawn. In the presence of walls, it turns out that Snake remains PSPACE-complete. We show this using minor modifications to the construction of Theorem 4.2.

Theorem 5.1. Snake *is PSPACE-complete for any growth rate $g \geq 1$, even if $|P| = 1$.*

Proof. We reuse the instance of Theorem 4.2, and set the permutation of items in the following way. As long as the snake is in the initial corridor, the food spawns just ahead of the snake, so that when the snake reaches the end of the initial corridor, its target length is as before. The item that appears last is the one in the dead end. Other items (those in $\mathcal{G}$) can be ordered arbitrarily.

We claim that all items can be consumed if and only if all items could be consumed in the Nibbler instance. We first argue that if all items can be consumed in the Snake instance, they can also be consumed in the Nibbler

instance. For this, the NIBBLER instance can be solved using the same sequence of edge-flips as the SNAKE solution, and all food is consumed this way.

Conversely, to obtain a solution for SNAKE from NIBBLER, recall that any sequence of edge reversals in NCL can be undone, and hence replayed as often as necessary, allowing all food to be consumed. □

6. Conclusion

In this paper we analyzed the computational complexity of deciding whether in arbitrarily sized levels of the games Snake and Nibbler, there is a way to collect a certain number of items. It turns out that both games are PSPACE-complete, and even some restricted versions in which there are no walls are NP-hard.

In order to prove our results, we used the Nondeterministic Constraint Logic (NCL) framework by Hearn and Demaine, which has been used to analyze the computational complexity of many (video)games and puzzles; and we extended it to the case in which the initial configuration is not given but it is chosen arbitrarily by the player. Given a configuration and an edge, the problem of deciding whether that edge of the NCL graph can be reversed is PSPACE-complete. Moreover, if we require that two edges must be reversed then the problem remains PSPACE-complete even if the player can choose the initial configuration. These variants of NCL remain PSPACE-complete even for NCL graphs in normal form that have bounded bandwidth.

An open question is whether SNAKE and NIBBLER remain hard for instances of bounded bandwidth. This is not immediate in our reduction, as the transformation of an NCL instance to a Nibbler or Snake instance does not preserve bounded bandwidth. For instance, the length of connector paths is polynomial in the number of vertices of the NCL graph.

For rectangular grid graphs, it seems that SNAKE is significantly easier than NIBBLER. This leads to the open question whether NIBBLER is polynomial time solvable on rectangular grid graphs if the snake is initially short.

Acknowledgments. Tim Ophelders is supported by the Netherlands Organization for Scientific Research (NWO) under project no. 639.023.208.

References

[1] Elias Dahlhaus, Peter Horák, Mirka Miller, and Joseph F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1–3):41–54, 2000.

[2] Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In *Games of No Chance 3*, volume 56 of *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009.

[3] Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint

logic model of computation. *Theoretical Computer Science*, 343(1):72–96, 2005.

[4] Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009.

[5] Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

[6] Irina Kostitsyna and Valentin Polishchuk. Simple wriggling is hard unless you are a fat hippo. *Theory of Computing Systems*, 50(1):93–110, 2012.

[7] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[8] Amit Pamecha, Imme Ebert-Uphoff, and Gregory S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13(4):531–545, 1997.

[9] Christos H. Papadimitriou and Umesh V. Vazirani. On two geometric problems related to the traveling salesman problem. *Journal of Algorithms*, 5(2):231–246, 1984.

[10] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[11] Christopher Umans and William Lenhart. Hamiltonian cycles in solid grid graphs. In *FOCS*, pages 496–505, 1997.

[12] Tom C. van der Zanden. Parameterized Complexity of Graph Constraint Logic. In *IPEC*, pages 282–293, 2015.

[13] Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.

[14] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10:399, 2013.