The Parameterized Complexity of Motion Planning for Snake-Like Robots

Siddharth Gupta[1*], **Guy Sa'ar**[1†], **Meirav Zehavi**[1‡]
[1]Ben-Gurion University of the Negev, Israel
{siddhart, saag}@post.bgu.ac.il, meiravze@bgu.ac.il

Abstract

We study a motion-planning problem inspired by the game Snake that models scenarios like the transportation of linked wagons towed by a locomotor to the movement of a group of agents that travel in an "ant-like" fashion. Given a "snake-like" robot with initial and final positions in an environment modeled by a graph, our goal is to decide whether the robot can reach the final position from the initial position without intersecting itself. Already on grid graphs, this problem is PSPACE-complete [Biasi and Ophelders, 2018]. Nevertheless, we prove that even on general graphs, it is solvable in time $k^{\mathcal{O}(k)}|I|^{\mathcal{O}(1)}$ where k is the size of the robot, and $|I|$ is the input size. Towards this, we give a novel application of color-coding to sparsify the *configuration graph* of the problem. We also show that the problem is unlikely to have a polynomial kernel even on grid graphs, but it admits a treewidth-reduction procedure. To the best of our knowledge, the study of the parameterized complexity of motion problems has been largely neglected, thus our work is pioneering in this regard.

1 Introduction

A basic *single-agent movement problem* can be modeled by an *agent* (representing a robot or a person) that has an initial *state* (also called *configuration*), a description of valid *transitions* between states, and a task to accomplish. Common tasks are to reach some geographical location while avoiding unwelcome (mobile or static) obstacles, collecting or distributing a set of items, or rearranging the environment to be of a specific form. The agent itself might have various features or restrictions, which are reflected in the definition of states and transitions. When several agents are present (in a multi-agent movement problem), the coordination between them might also play a major role (see [Demaine *et al.*, 2018] and references within). Arguably, given that we handle physical objects, the most basic requirement is that the agent must never *intersect* itself as well as other objects. Problems based on motion planning are ubiquitous in various aspects of modern life. In recent years, the study of such problems has gained increasing interest from both practical and theoretical viewpoints [Schwartz and Sharir, 1988; Galceran and Carreras, 2013; Paden *et al.*, 2016; Yang *et al.*, 2016]. Unfortunately, the perspective of parameterized complexity—a central paradigm to design algorithms for computationally hard problems—has been largely overlooked in this context. In this paper, we present a comprehensive picture of the parameterized complexity of a single-agent movement problem called SNAKE GAME, whose formulation is inspired by the classic video game of the same name.

In the past decade, the study of the theory behind the computational complexity of puzzles (such as video games) has become very popular [Hearn and Demaine, 2009; Kendall *et al.*, 2008; Viglietta, 2014]. These puzzles are often based on motion planning problems that can model tasks to be performed by agents in real-life scenarios. Moreover, their formulations are frequently simple enough to provide a clean abstraction of basic issues in this regard, therefore making them attractive for laying foundations for general analysis. For example, a very long line of work analyzed the complexity of various push-block puzzles (see [Demaine *et al.*, 2017] and references within), where a box-shaped agent with the ability to push/pull other boxes should utilize its ability in order to reach one position from another. We remark that more often than not, studies of the theory behind the computational complexity of puzzles only provide negative results that assert NP-hardness or PSPACE-completeness.

The classic game Snake is among the most well-known video games that involve the motion of a single agent. The game dates back to 1978, and has enjoyed implementation across a wide range of platforms since then. Unlike most other video games, the popularity of Snake has hardly decreased despite its age—indeed, new versions of Snake still appear to this day. We study the parameterized complexity of a problem inspired by Snake that was introduced by [Biasi and Ophelders, 2018], which we call SNAKE GAME. This problem is of relevance to real-world motion planning problems for agents of a "snake-like" shape (as discussed below). Given such a robot with an initial position and a final position in an environment (modeled by a graph), our objective is to determine whether the robot can reach the final position

[*]Supported in part by the Zuckerman STEM Leadership Program
[†]Supported in part by the Frankel Foundation
[‡]Supported by Israel Science Foundation (ISF) grant no. 1176/18

from the initial position without intersecting itself. Roughly speaking, the position of the robot is modeled by a simple ordered path in the graph, and one position P is reachable (in one step) from another position P' if the path P can be obtained from P' by adding one vertex to the beginning of P' and removing one vertex from its end. The (immobile) obstacles in the environment are implicitly encoded in the input graph—"obstacle-free" physical locations are represented by vertices, and edges indicate which locations are adjacent.

Nowadays, robots of a "snake-like" shape are of substantial interest–in particular, they are built and used in practice for medical operations [Degani *et al.*, 2006; Grifantini, 2008; Berthet-Rayne *et al.*, 2018] as well as various inspection and rescue missions on both land and water [Pfotzer *et al.*, 2017; Ye *et al.*, 2004; Lu *et al.*, 2016]. A snake-like shape and serpentine locomotion offer immediate advantages for such purposes; the restricted area of mobility also makes the requirement of the robot to avoid intersecting itself and other obstacles a highly non-trivial issue that is mandatory to take into account. Moreover, the SNAKE GAME problem is a natural abstraction to model a wide-variety of other scenarios, which range from the transportation of linked wagons towed by a locomotor at an airport or a supermarket to the movement of a group of agents that travel in an "ant-like" fashion and the construction of trains in amusement parks.

Biasi and Ophelders [Biasi and Ophelders, 2018] proved that the SNAKE GAME problem is PSPACE-complete even on *grid graphs* (see Section 2). Additionally, they considered the version aligned with the video game, where "food" items are located on vertices. Here, the task is not to reach a pre-specified position, but to collect all food items by visiting their vertices—when a food item is collected, the size of the snake increases by a fixed integer $g \geq 1$. They showed that this version is NP-hard even on rectangular grid graphs without "holes", and PSPACE-complete even when there are only two food items, or the initial size of the snake is 1.

Related Works in Parameterized Complexity. To the best of our knowledge, the only work in Parameterized Complexity on single/multi-agent movement problems is by [Cesati and Wereham, 1995][1]. The input of the problem in [Cesati and Wereham, 1995] is a set O of polyhedrons (obstacles), and a set P of polyhedrons (the robot) that are freely linked together at a set of linkage vertices V such that P has k degrees of freedom of movement. The objective is to decide whether a given final position of the robot is reachable from a given initial position of the robot where the robot is allowed to intersect neither itself nor the obstacles. [Reif, 1979] proved that this problem (in 3-dimensional space) is PSPACE-hard, and [Cesati and Wereham, 1995] adapted Reif's proof and showed that the problem is W[SAT]-hard with respect to k.

Clearly, problems where intermediate states are undefined/immaterial (such as SHORTEST PATH) can still concern issues relevant to motion planning. Here, we point out a comprehensive work by [Demaine *et al.*, 2014] on the parameterized complexity of a wide-class of motion planning problems for multiple agents in a "static sense" (similar to SHORTEST PATH): given a set of agents with types, an initial position (a single vertex) for each agent and a target configuration (e.g., each agent of type "client" should have an agent of type "facility" nearby), the goal is to make the agents traverse minimum distance towards the formation of the target configuration. Here, intermediate states are undefined/immaterial; thus, on the way to form the target configuration, the agents can be in arbitrary configurations such as being placed all together on the same vertex, or being as far apart as possible.

Our Contribution. We present a comprehensive picture of the parameterized complexity of SNAKE GAME parameterized by the size of the snake, k. The choice of this parameter is the most natural and sensible one—in the real-life scenarios mentioned earlier, the size of the snake-like robot is likely to be substantially smaller than that of the entire environment. To some extent, our paper is a pioneering work in the study of the parameterized complexity of single-agent movement problems, and may lay the foundations for further research of this topic. Specifically, our contribution is threefold.

I. FPT Algorithm. Our main result asserts that SNAKE GAME is *fixed-parameter tractable (FPT)* with respect to k. Specifically, we develop an algorithm that solves SNAKE GAME in time $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ where n is the number of vertices in the input graph. Our algorithm can also output the length of the shortest "route" from the initial position to the final position (if any such route exists) within the same time complexity. The design of our algorithm involves a novel application of the method of color-coding [Alon *et al.*, 1995] to sparsify the ($(k-1)$-th power of the) *configuration graph* of the problem. Roughly speaking, the configuration graph is the directed graph whose vertices represent the positions of the snake in the environment, and where there is an arc from one vertex u to another vertex v if the position represented by v is reachable in one step from the position represented by u. The number of vertices of the configuration graph equals the number of (simple) ordered paths on k vertices in the input graph, which can potentially be *huge*—for example, if the input graph is a clique, then there are $\binom{n}{k}k!$ configurations.

We first present a handy characterization of the reachability of one configuration from another in $t \geq 1$ steps, based on which we elucidate the structure of certain triplets of configurations. Then, we perform several iterations where we color the vertices of the input graph based on the method of color-coding, but where order between some of the colors is of importance. Within each coloring iteration, we test for every pair of vertices in the input graph whether there exists a particular path on k vertices between them—in which case we pick one such path. The collection of paths found throughout these iterations form the vertex set of our new configuration graph (in addition to the initial and final positions), while our characterization of reachability determines the arcs. In particular, this new configuration graph, unlike the original configuration graph, has only $k^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ vertices. We prove that this new configuration graph is a valid "sparsification" of the $(k-1)$-th power of the original configuration graph—specifically, for any $t \geq 1$, the initial position can reach the final position in the original configuration graph within $(k-1)t$ steps if and only if the initial position

[1] Simultaneously to our work, a paper on reconfiguration of undirected paths has appeared [Demaine *et al.*, 2019].

can reach the final position in the new configuration graph within t steps. Clearly, this is insufficient because the initial position may reach the final position in the original configuration graph within a number of steps that is not a multiple of $(k-1)$—however, we find that this technicality can be overcome by adding, to the new configuration graph, a small number of new vertices as well as arcs outgoing from these new vertices to the vertex representing the final position.

II. Kernelization. Our second result shows that SNAKE GAME is unlikely to admit a polynomial kernel even on grid graphs. For this purpose, we present a non-trivial *cross-composition* from HAMILTONIAN CYCLE on grid graphs to SNAKE GAME. Our construction is inspired by the proof in [Biasi and Ophelders, 2018] of NP-hardness of a version of SNAKE GAME in the presence of food. Roughly speaking, given t instances of HAMILTONIAN CYCLE on grid graphs, we construct an instance of SNAKE GAME as follows. We position the t input grid graphs so that they are aligned to appear one after the other, and connect them as "pendants" from a long path L placed just above them. The initial position of the snake is at the beginning of L, and its final position is at a short path "protruding" from the beginning of L. Further, the size of the snake is set to n, the number of vertices in each input grid graph. Intuitively, we show that the snake can reach the final position from the initial one if and only if it can enter and exit one of the input grid graphs. In particular, to reach the final position, the snake has to find a place to "turn around", which can only be (potentially) done inside one of the input grid graphs. Entering and exiting one of the input grid graphs would imply that at some state, the snake must be fully inside that graph and hence exhibit a Hamiltonian cycle.

III. Treewidth-Reduction. Our last result is a treewidth-reduction procedure: we develop a polynomial-time algorithm that given an instance of SNAKE GAME, outputs an equivalent instance of SNAKE GAME where the treewidth of the graph is bounded by a polynomial in k. Our procedure is based on the irrelevant vertex technique [Robertson and Seymour, 1995]. First, we exploit the breakthrough result by [Chekuri and Chuzhoy, 2016] that states that for any $t \in \mathbb{N}$, any graph whose treewidth is at least $d \cdot t^c$ (for some constants c and d) has a $t \times t$-grid as a minor, and hence also a t-*wall* as a subgraph. (Currently, the best bound on c is 10 [Chuzhoy and Tan, 2019].) We utilize this result to argue that if the treewidth of our input graph is too large, then it has a ck-wall as a subgraph (for some constant c) such that no vertex of this ck-wall belongs to the initial or final positions of the snake. The main part of our proof is a non-trivial *re-routing argument* that shows that in such a wall, we can arbitrarily choose any pair of adjacent vertices, contract the edge between them and thereby obtain an equivalent instance of SNAKE GAME. Thus, as long as we do not yet have a graph of small treewidth at hand, we can efficiently find an edge to contract, and eventually obtain a graph of small treewidth. From this procedure, we also derive that $m = k^{\mathcal{O}(1)}n$.

2 Preliminaries

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$, and $\mathbb{N}_0^4 = \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$. For a graph G, $V(G)$ and $E(G)$ denote its vertex and edge sets, respectively. For $v \in V(G)$, $N_G(v) = \{u \in V(G) \mid \{u,v\} \in E(G)\}$. The *contraction* of an edge $e = \{u,v\} \in E(G)$ is the replacement of u and v by a new vertex w where $N_G(w) = N_G(u) \cup N_G(v)$, to obtain a new graph denoted by G/e. The *subdivision* of an edge $\{u,v\} \in E(G)$ is the deletion of $\{u,v\}$ and the addition of the edges $\{u,w\}$ and $\{w,v\}$ for a new vertex w. A *Hamiltonian cycle* is a (simple) cycle that visits all the vertices of G. For a path P, the *size* and the *length* of P denote the number of vertices and edges in P, respectively.

Definition 2.1 (**Grid Graph**). *A* grid graph *is a finite undirected graph G with $V(G) \subseteq \{(i,j) \mid i,j \in \mathbb{N}_0\}$, and $\{(i,j),(i',j')\} \in E(G)$ if and only if $|i-i'| + |j-j'| = 1$.*

The *treewidth* of a graph G is a standard measure for its distance to a tree, defined below. Any tree has treewidth 1, and an n-vertex clique has treewidth $n-1$.

Definition 2.2 (**Treewidth**). *A* tree decomposition *of a graph G is a tree T whose nodes, called* bags*, are subsets of $V(G)$. For each $v \in V(G)$, the bags containing v form a nonempty subtree of T, and for each $\{u,v\} \in E(G)$, at least one bag contains both u and v. The* width *of the decomposition is one less than the maximum size of any bag, and the* treewidth *of G is the minimum width of any of its tree decompositions.*

Snake Game. We first define the notion of a configuration.

Definition 2.3 (**Configuration**). *For an undirected graph G and $k \in \mathbb{N}$, a (G,k)-*configuration *(for short,* configuration*) is a tuple $(v_1, v_2, \ldots, v_k)$, where $v_i \in V(G)$ for all $1 \leq i \leq k$, that satisfies two conditions:* **(i)** *for all $1 \leq i \leq k-1$, $\{v_i, v_{i+1}\} \in E(G)$, and* **(ii)** *for all $1 \leq i < j \leq k$, $v_i \neq v_j$.*

For a configuration conf, let $V(\mathsf{conf})$ be its set of vertices. Intuitively, a configuration $(v_1, v_2, \ldots, v_k)$ is the sequence of vertices of a simple path on k vertices in G traversed from one endpoint to another; the path is termed a *snake*, and the vertices v_1 and v_k are its *head* and *tail*, respectively. Now, we define how a snake "moves" from one position to another.

Definition 2.4 (**1-Transition**). *For an undirected graph G and $k \in \mathbb{N}$, a pair $(\mathsf{conf} = (v_1, v_2, \ldots, v_k), \mathsf{conf}' = (v_1', v_2', \ldots, v_k'))$ of configurations is a* 1-transition *if* **(i)** *for all $1 \leq i \leq k-1$, $v_1' \neq v_i$, and* **(ii)** *for all $2 \leq i \leq k$, $v_i' = v_{i-1}$.*

We extend a transition in one step to ℓ steps as follows.

Definition 2.5 (**ℓ-Transition**). *Let G be an undirected graph, and $k, \ell \in \mathbb{N}$. A pair $(\mathsf{conf}, \mathsf{conf}')$ of configurations is an* ℓ-transition *if there is a tuple $(\mathsf{conf}_1 = \mathsf{conf}, \mathsf{conf}_2, \ldots, \mathsf{conf}_{\ell+1} = \mathsf{conf}')$ of $\ell+1$ configurations such that, for every $1 \leq i \leq \ell$, $(\mathsf{conf}_i, \mathsf{conf}_{i+1})$ is a 1-transition.*

Based on the definition of a transition, we define the reachability of one configuration from another.

Definition 2.6 (**Reachability**). *Let G be an undirected graph, and $k \in \mathbb{N}$. Let conf and conf' be two configurations. We say that* conf can reach conf' *(alternatively,* conf' is reachable from conf*) if $(\mathsf{conf}, \mathsf{conf}')$ is an ℓ-transition for some $\ell \in \mathbb{N}$.*

We are now ready to define the SNAKE GAME problem.

Definition 2.7 (**Snake Game**). *An instance of* SNAKE GAME *is quadruple $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin} \rangle$ where G is an undirected graph, $k \in \mathbb{N}$, and* init *and* fin *are two configurations. We say that $\mathbf{SG}$ is a Yes-instance if* init *can reach* fin*; otherwise,*

SG *is a* No*-instance. To* solve **SG**, *it is required to determine whether it is a* Yes*-instance or a* No*-instance.*

We also define an auxiliary graph on configurations.

Definition 2.8 (**ℓ-Configuration Graph**). *Let G be an undirected graph, and $k, \ell \in \mathbb{N}$. The* ℓ-configuration graph *of (G, k) is a directed graph with a vertex for every (G, k)-configuration and an arc from a vertex* conf *to another vertex* conf′ *if* (conf, conf′) *is an ℓ-transition.*

The notion of reachability can be analyzed via the 1-configuration graph as implied by the following observation.

Observation 2.1. *Let G be an undirected graph, and $k \in \mathbb{N}$. A configuration* conf can reach *a configuration* conf′ *if and only if there is a path from* conf *to* conf′ *in the* 1*-configuration graph. In particular, an instance* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin} \rangle$ *of* Snake Game *is a* Yes*-instance if and only if there is a path from* init *to* fin *in the* 1*-configuration graph.*

Parameterized Complexity. A problem Π is *parameterized* if each instance of Π is associated with a *parameter* k. A problem Π is *FPT* if any instance (I, k) of Π is solvable in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is any computable function of k. Furthermore, Π admits a *polynomial compression* if there exists a (not necessarily parameterized) problem Π' and a polynomial-time algorithm that given an instance (I, k) of Π, outputs an equivalent instance I' of Π' (i.e. (I, k) is a Yes-instance of Π if and only if I' is a Yes-instance of Π') such that $|I'| \leq p(k)$ where p is any polynomial that depends only on k. In case $\Pi' = \Pi$, Π admits a *polynomial kernel*. For more information, refer to [Cygan *et al.*, 2015].

3 FPT Algorithm on General Graphs

In this section, we present our FPT algorithm for Snake Game. To begin our analysis, we give two conditions that characterize when a pair of configurations is an ℓ-transition. (For an example, see Fig. 1.) The correctness can be proved by induction on ℓ. For lack of space, full proofs are omitted.

Lemma 3.1. *For any $\ell \leq k$, a pair* $(\mathsf{conf} = (v_1, v_2, \ldots, v_k), \mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k))$ *of configurations is an ℓ-*transition *if and only if* **(i)** *for all $1 \leq i \leq \ell$, $v'_i \notin \{v_1, v_2, \ldots, v_{(k+i)-(\ell+1)}\}$, and* **(ii)** *for all $\ell + 1 \leq i \leq k$, $v'_i = v_{i-\ell}$.*

As a corollary, we obtain the following result.

Corollary 3.1. *A pair* $(\mathsf{conf} = (v_1, v_2, \ldots, v_k), \mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k))$ *of configurations is a* $(k-1)$-transition *if and only if* **(i)** *for all $1 \leq i \leq k-1$, $v'_i \notin \{v_1, \ldots, v_i\}$, and* **(ii)** *$v'_k = v_1$.*

In turn, this corollary gives rise to a simple test of whether a pair of configurations is a $(k-1)$-transition as follows.

Observation 3.1. *It can be tested whether a pair* (conf, conf′) *of configurations is a $(k-1)$-transition in time $\mathcal{O}(k^2)$.*

To sparsify the $(k-1)$-configuration graph, we utilize the method of color-coding [Alon *et al.*, 1995]. While standard applications are coupled with a derandomization object called a *splitter*, we require a related object called a *permuter*.

Definition 3.1 (**Permuter**). *Let $t \in \mathbb{N}$, and S be a set. An* (S, t)-permuter *$\mathcal{G}$ is a family of functions from S to $\{1, \ldots, t\}$ such that for any ordered set $W \subseteq S$ and for any $i, j \in \mathbb{N}$ such that $i \leq j \leq t$ and $j - i + 1 = |W| \leq t$, there is a function $g \in \mathcal{G}$ mapping W to the ordered set $\{i, i+1, \ldots, j\}$.*

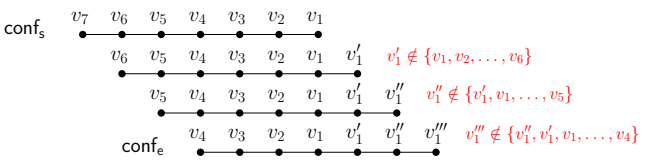

Figure 1: An example of a 3-transition $(\mathsf{conf}_s, \mathsf{conf}_e)$ where $k = 7$.

An efficient construction for an (S, t)-permuter follows from the efficient construction of a splitter [Naor *et al.*, 1995].

Lemma 3.2. *Given $t \in \mathbb{N}$ and a set S of size n, an (S, t)-permuter of size $t^{\mathcal{O}(t)} \log n$ can be found in time $t^{\mathcal{O}(t)} n \log n$.*

Given three configurations conf, conf′ and conf″, we define a special ordering, called *triplet order*, on the vertices of the triplet (conf, conf′, conf″). (For an example, see Fig. 2.)

Definition 3.2 (**Triplet Order**). *Let* $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$, $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k)$ *and* $\mathsf{conf}'' = (v''_1, v''_2, \ldots, v''_k)$ *be three configurations. The* triplet order *of* (conf, conf′, conf″) *is the ordered set W obtained from the ordered multiset $S = \{v_k, v_{k-1}, \ldots, v_1, v'_k, v'_{k-1}, \ldots, v'_1, v''_k, v''_{k-1}, \ldots, v''_1\}$ by removing first the vertices of* conf *and* conf″ *that are common to* conf′*, and then the other vertices of* conf *common to* conf″*.*

Note that W is an ordered set where each distinct vertex in S occurs once. The relation between a triplet order and a permuter is summarized as follows.

Lemma 3.3. *For any $\ell, r < k$, let* $\mathsf{conf} = (v_1, \ldots, v_k)$, $\mathsf{conf}' = (v'_1, \ldots, , v'_k)$ *and* $\mathsf{conf}'' = (v''_1, \ldots, , v''_k)$ *where* (conf, conf′) *is an ℓ-transition, and* (conf′, conf″) *is an r-transition. Let $\mathcal{G}$ be a $(V(G), 3k-2)$-permuter. Then, there is $f \in \mathcal{G}$ that assigns distinct integers to the vertices in the triplet order W of* (conf, conf′, conf″)*, and satisfies the following conditions.*

- *(i) f maps W to an ordered set $\{i, i+1, \ldots, k, k+1, \ldots, 2k-1, \ldots, j\}$ where $1 \leq i \leq j \leq 3k-2$ and $j-i+1 = |W|$. Moreover, for each $1 \leq a \leq k$, $f(v'_a) = 2k-a$.*
- *(ii) If there is* $\mathsf{conf}^* = (w_1 = v'_1, w_2, \ldots, w_k = v'_k)$ *such that* **(1)** *for every $1 \leq a \leq k$, $f(w_a) = 2k-a$,* **(2)** *for every $\ell + 1 \leq a \leq k$, $w_a = v_{a-\ell}$, and* **(3)** *for every $r + 1 \leq a \leq k$, $v''_a = w_{a-r}$, then* (conf, conf*) *is an ℓ-transition and* (conf*, conf″) *is an r-transition.*

The sparsification also requires an efficient computation of paths that will determine the vertices of the sparse configuration graph. The proof is based on a BFS computation.

Lemma 3.4. *There is a linear-time algorithm that, given an undirected graph G, $k, t, r \in \mathbb{N}$, a function $f : V(G) \to \{1, \ldots, t\}$, and vertices $u, v \in V(G)$, decides if G has a (simple) path $P = (w_1 = u, w_2, \ldots, w_k = v)$ (of size k between u and v) such that for every $1 \leq i \leq k$, $f(w_i) = r + k - i$; if such a path exists, then the algorithm outputs one such path.*

Given an instance $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin} \rangle$ of Snake Game, we now give a procedure (Algorithm 1) to construct a new configuration graph of **SG**, which invokes Observation 3.1 and Lemmata 3.2 and 3.4. We refer to the outputted graph as a $(k-1)$-*sparse configuration graph*. Unlike the $(k-1)$-configuration graph, a $(k-1)$-sparse configuration graph may not be unique for **SG** even if the permuter is fixed. Indeed,

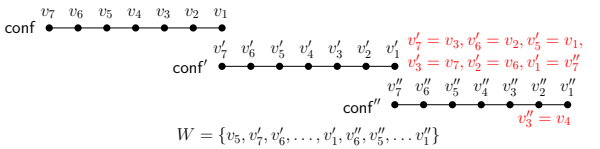

Figure 2: An illustration for Lemma 3.3 where $\ell = 4$ and $r = 6$.

there may be exponentially many paths satisfying the condition in line 5 of Algorithm 1, but we only add one of them (chosen arbitrarily) to our $(k-1)$-sparse configuration graph.

We bound the size of the output and the running time of Algorithm 1 as follows. (Here, $n = |V(G)|$.)

Lemma 3.5. *Algorithm 1 runs in time $k^{\mathcal{O}(k)}n^3 \log^2 n$, and returns a $(k-1)$-sparse configuration graph with $k^{\mathcal{O}(k)}n^2 \log n$ vertices and $k^{\mathcal{O}(k)}n^3 \log^2 n$ arcs.*

Having Lemma 3.3, we derive a relation between the $(k$–$1)$ -configuration graph and a $(k-1)$-sparse configuration graph.

Lemma 3.6. *Let $\mathcal{C}$ be the $(k-1)$-configuration graph, and $\mathcal{C}'$ be a $(k-1)$-sparse configuration graph. Then,* **(i)** *$\mathcal{C}'$ is an induced subgraph of $\mathcal{C}$. Moreover,* **(ii)** *for any $\mathsf{conf}, \mathsf{conf}', \mathsf{conf}'' \in V(\mathcal{C})$ where $(\mathsf{conf}, \mathsf{conf}')$ and $(\mathsf{conf}', \mathsf{conf}'')$ are $(k-1)$-transitions, there is $\mathsf{conf}^* \in V(\mathcal{C}')$where $(\mathsf{conf}, \mathsf{conf}^*)$ and $(\mathsf{conf}^*, \mathsf{conf}'')$ are also $(k-1)$-transitions.*

We now relate paths in the $(k-1)$-configuration graph and a $(k-1)$-sparse configuration graph. The "if" direction is a consequence of item **(i)** in Lemma 3.6, and the "only if" direction can be proved by a repeated application of item **(ii)**.

Lemma 3.7. *Let $\mathcal{C}$ be the $(k-1)$-configuration graph, $\mathcal{C}'$ be a $(k-1)$-sparse configuration graph, and $\mathsf{conf}_s, \mathsf{conf}_e \in V(\mathcal{C})$. For any $t \in \mathbb{N}$, there is a path $P = (\mathsf{conf}_1 = \mathsf{conf}_s, \mathsf{conf}_2, \ldots, \mathsf{conf}_{t+1} = \mathsf{conf}_e)$ from conf_s to conf_e in $\mathcal{C}$ if and only if there is a path $P' = (\mathsf{conf}'_1 = \mathsf{conf}_s, \mathsf{conf}'_2, \ldots, \mathsf{conf}'_t, \mathsf{conf}'_{t+1} = \mathsf{conf}_e)$ from conf_s to conf_e in $\mathcal{C}$ where $\mathsf{conf}'_i \in V(\mathcal{C}')$ for each $2 \le i \le t$.*

By substituting $\mathsf{conf}_s = \mathsf{init}$ and $\mathsf{conf}_e = \mathsf{fin}$, we know that there is a path from init to fin in $\mathcal{C}$ if and only if there is a path of the same length from init to fin in $\mathcal{C}'$. From this corollary, we easily derive the following result.

Lemma 3.8. *Let $\mathcal{C}'$ be a $(k-1)$-sparse configuration graph. For any $t \in \mathbb{N}$, there is a path of length $t(k-1)$ from* init *to* fin *in the 1-configuration graph if and only if there is a path of length t from* init *to* fin *in $\mathcal{C}'$.*

Thus, the sparsification of the $(k-1)$-configuration graph alone is insufficient—there may exist a path from init to fin in the 1-configuration graph whose length is not a multiple of $k-1$. This technicality is not difficult to overcome by adding $k^{\mathcal{O}(k)}n \log n$ new vertices and arcs (incident to fin) to $\mathcal{C}'$ and thus deriving an *enriched $(k-1)$-sparse configuration graph*.

Lemma 3.9. *There is an $k^{\mathcal{O}(k)}n^3 \log^2 n$-time algorithm that returns an enriched $(k-1)$-sparse configuration graph $\mathcal{C}''$ with $k^{\mathcal{O}(k)}n^2 \log n$ vertices and $k^{\mathcal{O}(k)}n^3 \log^2 n$ arcs. For any $t, r \in \mathbb{N}$ where $r < k-1$, there is a path of length $t(k-1)+r$ from* init *to* fin *in the 1-configuration graph if and only if there is a path of length t from* init *to* fin *in $\mathcal{C}'$.*

Algorithm 1: sparseConfigurationGraph($\mathbf{SG}, k$)

```
1  let C' with V(C') = {init, fin} and E(C') = ∅;
2  construct a (V(G), 3k − 2)-permuter G;
3  for each g ∈ G do
4      for each pair (u, v) ∈ V(G) × V(G) do
5          if G has a path P = (w_1 = u, w_2, ..., w_k = v) s.t.
             for each 1 ≤ a ≤ k, g(w_a) = 2k − a then
6              add conf = (w_1, ..., w_k) to V(C');
7          end
8      end
9  end
10 for each pair (conf, conf') ∈ V(C') × V(C') that is a
     (k − 1)-transition do
11     add the arc (conf, conf') toE(C');
12 end
13 return C'.
```

Our theorem follows from this lemma and Observation 2.1.

Theorem 3.1. SNAKE GAME *is solvable in time $k^{\mathcal{O}(k)}n^3 \log^2 n$, and a shortest path from* init *to* fin *(if one exists) can be found within the same time complexity.*

4 No Polynomial Kernel on Grid Graphs

In this section, we prove that SNAKE GAME is unlikely to admit a polynomial kernel even on grid graphs, based on a theorem on cross-compositions by [Bodlaender *et al.*, 2009].

Definition 4.1 (Cross-Composition). *A problem Π* cross-composes *into a parameterized problem Π' if there is a polynomial-time algorithm, called a* cross-composition*, that given t instances $I_1, I_2, \ldots, I_t$ of Π of the same size s, outputs an instance (I, k) of Π' where $k \le p(s)$ for some polynomial p in s, and (I, k) is a Yes-instance of Π' if and only if at least one of the instances $I_1, I_2, \ldots, I_t$ is a Yes-instance of Π.*

Proposition 4.1. *Let Π be an NP-hard problem that cross-composes into a parameterized problem Π'. Unless NP⊆coNP/poly, Π' admits no polynomial compression.*

In HAMILTONIAN CYCLE, the goal is to decide if a given graph has a Hamiltonian cycle; on grid graphs, this problem is NP-hard [Papadimitriou and Vazirani, 1984]. We present a cross-composition, HamToSna, whose input consists of t instances $G_1, \ldots, G_t$ of HAMILTONIAN CYCLE on grid graphs on n vertices where $t, n \in \mathbb{N}$, and its output is an instance **SG** of SNAKE GAME. Without loss of generality, let each G_i be connected. To construct **SG**, we need the following notion.

Definition 4.2 (Boundary Square). *A* boundary square *of a grid graph G is a tuple $(r_{\min}, r_{\max}, c_{\min}, c_{\max}) \in \mathbb{N}_0^4$ where:*

1. $r_{\min} = \min\{r \in \mathbb{N}_0 \mid \exists c \in \mathbb{N}_0$ *such that* $(r, c) \in V(G)\}$.
2. $c_{\min} = \min\{c \in \mathbb{N}_0 \mid \exists r \in \mathbb{N}_0$ *such that* $(r, c) \in V(G)\}$.
3. $r_{\max} - r_{\min} = c_{\max} - c_{\min} = |V(G)| - 1$.

Observation 4.1. *Any connected grid graph has a unique boundary square.*

We preprocess each G_i as follows: by Observation 4.1, G_i has a unique boundary square $(\widehat{r}^i_{\min}, \widehat{r}^i_{\max}, \widehat{c}^i_{\min}, \widehat{c}^i_{\max})$; we replace each vertex $(a, b) \in V(G_i)$ by the vertex $(a - \widehat{r}^i_{\min} +$

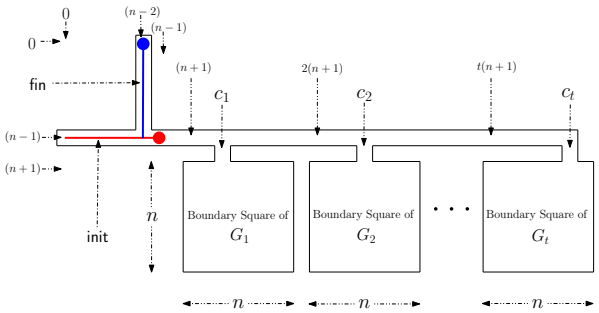

Figure 3: The graph constructed by the reduction HamToSna.

$(n+1), b-\widehat{c}^i_{\min}+i(n+1))$. Intuitively, this operation "pushes" G_i in parallel to the axes. It is easy to see that the new boundary square $(r^i_{\min}, r^i_{\max}, c^i_{\min}, c^i_{\max})$ of G_i has the following properties: $r^i_{\min} = n+1$ and $c^i_{\min} = i(n+1)$. Moreover, there exists $(a,b) \in V(G_i)$ where $a = r^i_{\min}$; we choose (arbitrarily) such a vertex in $V(G_i)$, and denote it by $(r^i_{\min}, c_i)$.

Now, $\mathsf{HamToSna}(G_1, \ldots, G_t) = \langle G, n, \mathsf{init}, \mathsf{fin}\rangle$ is defined as follows (see Fig. 3).

1. G is the grid graph on $V(G) = \widehat{V} \cup V^{\mathsf{conn}} \cup V^{\mathsf{inst}}$, where: $\widehat{V} = \{(n-1, j) \mid 0 \le j \le n-1\} \cup \{(i, n-2) \mid 0 \le i \le n-1\}$; $V^{\mathsf{conn}} = \{(n-1, j) \mid n \le j \le c_t\} \cup \{(n, c_i) \mid 1 \le i \le t\}$; $V^{\mathsf{inst}} = \bigcup_{i=1}^{t} V(G_i)$.
2. $\mathsf{init} = ((n-1, n-1), (n-1, n-2), \ldots, (n-1, 0))$.
3. $\mathsf{fin} = ((0, n-2), (1, n-2), \ldots, (n-1, n-2))$.

The intuition underlying this construction is given in Section 1. For lack of space, we directly summarize the result.

Lemma 4.1. HAMILTONIAN CYCLE *on grid graphs cross-composes into* SNAKE GAME *on grid graphs.*

Thus, by Proposition 4.1, we derive the following theorem.

Theorem 4.1. SNAKE GAME *on grid graphs does not admit a polynomial compression unless* NP ⊆ coNP/poly.

5 Treewidth-Reduction on General Graphs

In this section, we present a treewidth-reduction procedure for SNAKE GAME. Our algorithm is based on the analysis of walls, defined as follows (see Fig. 4).

Definition 5.1 (r**-Wall).** *For $r \in \mathbb{N}$, the* elementary r-wall *is the graph obtained from the $r \times 2r$-grid by deleting all edges $\{(2i-1, 2j-1), (2i, 2j-1)\}$ for $i \in \{1, \ldots, \lfloor r/2 \rfloor\}$ and $j \in \{1, \ldots, r\}$ and all edges $\{(2i, 2j), (2i+1, 2j)\}$ for $i \in \{1, \ldots, \lfloor (r-1)/2 \rfloor\}$ and $j \in \{1, \ldots, r\}$, and then deleting the two resulting degree-1 vertices. An* r-wall *is any graph obtained from the elementary r-wall by subdividing edges.*

From the work of [Chekuri and Chuzhoy, 2016], we deduce that to develop our treewidth-reduction procedure, we can focus on the case where G contains an $7k$-wall.

Proposition 5.1. *There is a polynomial-time algorithm that, given a graph G and $t \in \mathbb{N}$, either finds a t-wall H in G, or reports that G has treewidth $t^{\mathcal{O}(1)}$.*

We need our wall not only to be "large", but also to intersect neither init nor fin. To this end, we have the following lemma, proved by the pigeonhole principle (see Fig. 4).

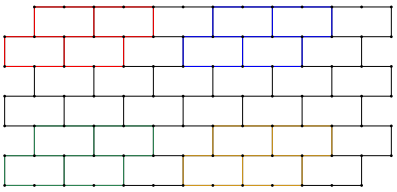

Figure 4: Four distinct elementary 3-walls in an elementary 7-wall.

Lemma 5.1. *Given an instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of* SNAKE GAME *where G contains a $7k$-wall, a $3\sqrt{k}$-wall that has vertices from neither* init *nor* fin *can be found in time $k^{\mathcal{O}(1)}$.*

Now, we aim to show that if G contains a $3\sqrt{k}$-wall, then we can efficiently decrease the number of vertices in G and still retain an equivalent instance of SNAKE GAME. Having a $3\sqrt{k}$-wall will be useful for us due to its following property.

Lemma 5.2. *Let H be a $3\sqrt{k}$-wall. Let $e = \{u, v\} \in E(H)$. For any $s, t \in V(H/e)$, there is a (simple) path P (or cycle if $s = t$) in H/e between s and t whose size is at least k.*

A routing argument proves the lemma below, see Fig. 5.

Lemma 5.3. *Let $\mathsf{conf}_1 = (u_1, \ldots, u_k)$ and $\mathsf{conf}_2 = (a_1, \ldots, a_k)$ be two configurations, and $P = (v_1, \ldots, v_\ell)$ be a simple path/cycle in G of size at least k such that* **(i)** *$v_1 = u_1$ and $v_\ell = a_k$,* **(ii)** *for all $2 \le i \le k$ and $2 \le j \le \ell$, $u_i \ne v_j$, and* **(iii)** *for all $1 \le i \le k-1$ and $1 \le j \le \ell - 1$, $a_i \ne v_j$. Then, conf_1 can reach conf_2.*

The proof of the main lemma of this section, stated below, is based on Lemmata 5.2 and 5.3. In particular, to prove the "only if" direction, we consider the first and last configurations in a "solution" transition that intersect H, and modify the sequence of configurations that occur between them.

Lemma 5.4. *Let $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ be an instance of* SNAKE GAME *where G contains a $3\sqrt{k}$-wall H having vertices from neither* init *nor* fin. *Let $e = \{u, v\} \in E(H)$. Then, $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ is a* Yes*-instance if and only if $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is a* Yes*-instance.*

Our treewidth-reduction procedure repeatedly invokes Proposition 5.1 and Lemma 5.1 to find a $3\sqrt{k}$-wall H having vertices from neither init nor fin (in which case it contracts an edge of H), or conclude that the treewidth of G is $k^{\mathcal{O}(1)}$. Correctness follows from Lemma 5.4.

Theorem 5.1. *There is a polynomial-time algorithm that, given an instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of* SNAKE GAME, *returns an equivalent instance $\langle G', k, \mathsf{init}, \mathsf{fin}\rangle$ of* SNAKE GAME *where G' has treewidth $k^{\mathcal{O}(1)}$.*

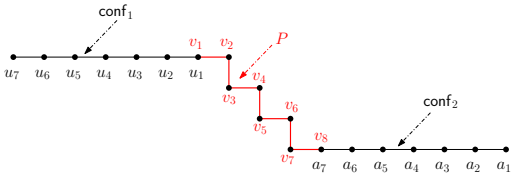

Figure 5: The conditions of Lemma 5.3 for $k = 7$ and $\ell = 8$.

References

[Alon *et al.*, 1995] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

[Berthet-Rayne *et al.*, 2018] Pierre Berthet-Rayne, Gauthier Gras, Konrad Leibrandt, Piyamate Wisanuvej, Andreas Schmitz, Carlo A. Seneci, and Guang-Zhong Yang. The i2snake robotic platform for endoscopic surgery. *Annals of Biomedical Engineering*, 46(10):1663–1675, Oct 2018.

[Biasi and Ophelders, 2018] Marzio De Biasi and Tim Ophelders. The complexity of snake and undirected NCL variants. *Theor. Comput. Sci.*, 748:55–65, 2018.

[Bodlaender *et al.*, 2009] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

[Cesati and Wereham, 1995] Marco Cesati and H. Todd Wereham. Parameterized complexity analysis in robot motion planning. In *The 25th IEEE International Conference on Systems, Man and Cybernetics*, pages 1–6, 1995.

[Chekuri and Chuzhoy, 2016] Chandra Chekuri and Julia Chuzhoy. Polynomial bounds for the grid-minor theorem. *J. ACM*, 63(5):40:1–40:65, 2016.

[Chuzhoy and Tan, 2019] Julia Chuzhoy and Zihan Tan. Towards tight(er) bounds for the excluded grid theorem. In *The Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1445–1464, 2019.

[Cygan *et al.*, 2015] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[Degani *et al.*, 2006] Amir Degani, Howie Choset, Alon Wolf, and Marco A Zenati. Highly articulated robotic probe for minimally invasive surgery. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4167–4172, 2006.

[Demaine *et al.*, 2014] Erik Demaine, Mohammad Taghi Hajiaghayi, and Dániel Marx. Minimizing movement: Fixed-parameter tractability. *ACM Trans. Algorithms*, 11(2):14:1–14:29, 2014.

[Demaine *et al.*, 2017] Erik Demaine, Isaac Grosof, and Jayson Lynch. Push-pull block puzzles are hard. In *Algorithms and Complexity - 10th International Conference, CIAC*, pages 177–195, 2017.

[Demaine *et al.*, 2018] Erik Demaine, Sándor P. Fekete, Phillip Keldenich, Christian Scheffer, and Henk Meijer. Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. In *34th International Symposium on Computational Geometry (SoCG)*, pages 29:1–29:15, 2018.

[Demaine *et al.*, 2019] Erik D. Demaine, David Eppstein, Adam Hesterberg, Kshitij Jain, Anna Lubiw, Ryuhei Uehara, and Yushi Uno. Reconfiguring Undirected Paths. *arXiv e-prints*, page arXiv:1905.00518, May 2019.

[Galceran and Carreras, 2013] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61:1258–1276, 2013.

[Grifantini, 2008] Kristina Grifantini. Snakelike robots for heart surgery. *MIT Technology Review*, 2008.

[Hearn and Demaine, 2009] Robert Hearn and Erik Demaine. *Games, puzzles and computation*. Peters, 2009.

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

[Lu *et al.*, 2016] Zhenli Lu, Dayu Feng, Yafei Xie, Huigang Xu, Limin Mao, Changkao Shan, Bin Li, Petr Bilik, Jan Zidek, Radek Martinek, and Zdenek Rykala. Study on the motion control of snake-like robots on land and in water. *Perspectives in Science*, 7:101 – 108, 2016. 1st Czech-China Scientific Conference 2015.

[Naor *et al.*, 1995] Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–191, 1995.

[Paden *et al.*, 2016] Brian Paden, Michal Cáp, Sze Zheng Yong, Dmitry S. Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. Intelligent Vehicles*, 1(1):33–55, 2016.

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

[Pfotzer *et al.*, 2017] L. Pfotzer, S. Klemm, A. Roennau, J.M. Zöllner, and R. Dillmann. Autonomous navigation for reconfigurable snake-like robots in challenging, unknown environments. *Robotics and Autonomous Systems*, 89:123 – 135, 2017.

[Reif, 1979] John H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 421–427, 1979.

[Robertson and Seymour, 1995] Neil Robertson and Paul D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[Schwartz and Sharir, 1988] Jacob T. Schwartz and Micha Sharir. A survey of motion planning and related geometric algorithms. *Artif. Intell.*, 37(1-3):157–169, 1988.

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

[Yang *et al.*, 2016] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, , and Yong Xia. Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, 2016(7426913):22, 2016.

[Ye *et al.*, 2004] Changlong Ye, Shugen Ma, Bin Li, and Yuechao Wang. Turning and side motion of snake-like robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5075–5080, 2004.