Journal of Artificial Intelligence Research 69 (2020) 191-229 Submitted 12/2019; published 09/2020

The Parameterized Complexity of Motion Planning for Snake-Like Robots

Siddharth Gupta SIDDHART@POST.BGU.AC.IL
Guy Sa'ar SAAG@POST.BGU.AC.IL
Meirav Zehavi MEIRAVZE@BGU.AC.IL
Ben-Gurion University of the Negev, Israel

Abstract

We study the parameterized complexity of a variant of the classic video game Snake that models real-world problems of motion planning. Given a snake-like 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 from the initial position without intersecting itself. Naturally, this problem models a wide-variety of scenarios, ranging 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. Unfortunately, already on grid graphs, this problem is PSPACE-complete. Nevertheless, we prove that even on general graphs, the problem is solvable in FPT time with respect to the size of the snake. In particular, this shows that the problem is fixed-parameter tractable (FPT). Towards this, we show how to employ color-coding to sparsify the configuration graph of the problem to reduce its size significantly. We believe that our approach will find other applications in motion planning. Additionally, we show that the problem is unlikely to admit 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 planning problems (where the intermediate configurations of the motion are of importance) has so far been largely overlooked. 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 desired position or 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. Arguably, given that we handle physical objects, a basic requirement is that the agent must never *intersect* itself as well as other objects. When several agents are present (in a multi-agent movement problem), the coordination between them also plays a major role (see Demaine, Fekete, Keldenich, Scheffer, & Meijer, 2018, and the references within.)

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 points of views (Schwartz & Sharir, 1988; Galceran & Carreras, 2013; Yan, Jouandeau, & Cherif, 2013; Paden, Cáp, Yong, Yershov, & Frazzoli, 2016; Yang, Qi, Song, Xiao, Han, & Xia, 2016). For example, there are studies that exploit the topological structure of robot configuration spaces in order to design efficient algorithms for motion planning problems (see, e.g., Farber, 2017, and the references within.) Moreover, there are studies that explore the complexity of path planning for robot motion planning and prove NP-hardness and lower bound results for some motion planning problems (see, e.g., Canny & Reif, 1987, and the references within.) Unfortunately, the perspective of parameterized complexity—a central paradigm to design algorithms for computationally hard problem—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 the Snake Game problem, whose formulation is inspired by a 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 & Demaine, 2009; Kendall, Parkes, & Spoerer, 2008; Viglietta, 2014). Such 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, Grosof, & Lynch, 2017, and the 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 assert the NP-hardness or PSPACE-completeness of the puzzle at hand.

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 variant of Snake that was introduced by Biasi and Ophelders (2018), which models real-world problems of motion planning for agents of a "snake-like" shape. Given a snake-like 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 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 is 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—specifically, "obstacle-free" physical locations are represented by vertices, and edges indicate which locations are adjacent. Note that the graph might not be planar as in real-life scenarios the environment is often not merely a plane.

Nowadays, robots of a "snake-like" shape are of substantial interest–in particular, they are built and used in practice for medical operations (Degani, Choset, Wolf, & Zenati, 2006; Grifantini, 2008; Berthet-Rayne, Gras, Leibrandt, Wisanuvej, Schmitz, Seneci, & Yang, 2018) as well as various inspection and rescue missions on both land and water (Pfotzer, Klemm, Roennau, Zöllner, & Dillmann, 2017; Ye, Ma, Li, & Wang, 2004; Lu, Feng, Xie, Xu, Mao, Shan, Li, Bilik, Zidek, Martinek, & Rykala, 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.

Another motivation for exploring the Snake Game problem is that it is a very basic/abstract motion planning problem. We suggest to utilize our approach in the context of other motion planning problems where the environment is large compared to the total size of the objects that can move within it. Examples of such problems include the Push-pull block problem (Demaine et al., 2017). In this problem, we have a robot that moves in an *environment*, described by a grid graph G, a *start* vertex and a *target* vertex. In the environment, there might be some movable *blocks* that the robot can move by pushing or pulling them. The problem is to decide if the robot can get from the start vertex to the target vertex, by moving each time from a vertex to one of its unoccupied neighbors (vertices that do not contain any blocks), or by pushing or pulling a block to an unoccupied neighbor. Another example is the Coordinated Motion Planning problem (Halperin, Salzman, & Sharir, 2017) in which we have k objects, each one with an initial and a target position, and the goal is to find a collision-free path for the objects from the initial position to the target position. A similar problem is the Unlabeled Motion Planning problem (Halperin et al., 2017), where the input is the same as in the Coordinated Motion Planning, and the goal is also to find a collision-free path for the objects. Here we are not interested in a specific assignment between robots and final placements as long as each final placement is occupied by some robot at the end of the motion. Halperin et al. (2017) describe more examples, such as the Motion Planning for a Tethered Robot and the Time-Varying Environments problems, and more. A broader survey of the results is beyond the scope of this paper. To strengthen our claim that our approach might be used in other motion planning problems, we remark that we have evidence that our treewidth reduction may be implemented in the Push-pull block problem.

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.

1.1 Our Contribution and Methods

We present a comprehensive picture of the parameterized complexity of the Snake Game problem parameterized by the size of the snake, k.[1] Arguably, the choice of this parameter is the most natural and sensible one because in real-life scenarios as those mentioned above, the size of the snake-like robot is likely to be substantially smaller than the size of the

1. Definitions of basic notions in Parameterized Complexity can be found in Section 2.

environment. To some extent, our paper can be considered as pioneering work in the study of the parameterized complexity of motion planning problems where intermediate configurations are of importance (which has so far been largely neglected, see the next subsection), and may lay the foundations for further research of this topic.

FPT Algorithm. Our contribution is threefold. Our main result is the proof that the Snake Game problem 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. We remark that 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 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. Observe that 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 (Alon, Yuster, & Zwick, 1995), 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 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 ingoing from these new vertices to the vertex representing the final position.

Kernelization. Our second result is the proof that the Snake Game problem is unlikely to admit a polynomial kernel even on grid graphs. For this purpose, we present a non-trivial *cross-composition* (defined in Section 2) from Hamiltonian Cycle on grid graphs to the Snake Game problem. Our construction is inspired by the proof of NP-hardness of the version of Snake Game on grid graphs without holes in the presence of food, given by Biasi and Ophelders (2018). Roughly speaking, given t instances of Hamiltonian Cycle on grid graphs, our construction of an instance of Snake Game is 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 that "protrudes" from the beginning of L. Further, the size of the snake is set to be n, the number of vertices in each of the input grid graphs. Intuitively, we show that the snake can reach the final position from the initial position if and only if the snake can enter and exit one of the input grid graphs. In particular, to reach the final position, the snake has to find a place where it can "turn around", which can only be (potentially) done inside one of the input grid graphs. Specifically, entering and exiting one of the input grid graphs, say, G_i, would imply that at some point of time, the snake must be fully inside G_i and hence exhibit a Hamiltonian cycle within it.

Treewidth-Reduction. Our last result is a treewidth-reduction procedure for the Snake Game problem. More precisely, 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 & Seymour, 1995). First, we exploit the relatively recent breakthrough result by Chekuri and Chuzhoy (2016) that states that, for any positive integer $t \in \mathbb{N}$, any graph whose treewidth is at least $d \cdot t^c$ (for some fixed constants c and d)[2] has a $t \times t$-grid as a minor, and hence also a so called t-wall as a subgraph (see Section 5). 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 fixed 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 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 the Snake Game problem. 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.

1.2 Related Works in Parameterized Complexity

To the best of our knowledge, close to nothing was known on the parameterized complexity of problems of planning motion of agents where the motion plays an actual role.[3] By this comment, we mean that the problem does not only have an initial state and a final state, and intermediate states (with transitions between them) are not defined/immaterial. When only an initial state and a final state are present, the problem is actually a *static* problem and the term movement refers to distances rather than motion. For example, a static version in this spirit of the Snake Game problem (that neglects intermediate states and hence does not enforce the requirement that the snake should never intersect itself) would just ask whether the vertex on which lies the tail of the snake in the final position is reachable from the vertex on which lies the head of the snake in the initial position, which can be directly solved in linear time using BFS. We also remark that almost all problems studied from the

2. Currently, the best known bound on c is 10, given by Chuzhoy and Tan (2019).
3. Simultaneously to our work, Demaine *et al.* (2019) also studied reconfiguration of undirected paths and found that path reconfiguration is fixed-parameter tractable when parameterized by the length of the path.

viewpoint of Parameterized Complexity are NP-complete rather than PSPACE-hard (with several notable exceptions such as Pan & Vardi, 2006; de Haan & Szeider, 2017).

While there is a huge line of works that explore the classical complexity and approximability of problems of planning motion of agents (see the beginning of the introduction), the only work in Parameterized Complexity that we are aware of is by Cesati and Wereham (1995). Roughly speaking, the input of the problem studied by Cesati and Wereham (1995) consists of a robot and obstacles that are each represented by a set of polyhedrons. Specifically, the input includes a set O of polyhedrons (the obstacles), as well as 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 three-dimensional Euclidean space) is PSPACE-hard, and Cesati and Wereham (1995) adapted Reif's proof and showed that the problem is W[SAT]-hard when parameterized by k and hence unlikely to be FPT.

Lastly, when intermediate states are immaterial, it is less clear which problem is considered to be centered around motion rather than euclidean/graph distances and static constraints—for example, in such scenarios finding a shortest path or a path with specific restrictions in a given graph may be highly relevant to motion planning (see, e.g., Eiben & Kanj, 2018), but is not centered around the motion itself in the sense above. In this context, it is noteworthy to mention the comprehensive work in Parameterized Complexity by Demaine, Hajiaghayi and Marx (2014). Roughly speaking, the authors addressed multi-agent problems on graphs where the agents have types, there is an initial position (being a single vertex) for each agent, and a desired configuration that all agents should form (e.g., every agent of type "client" should have an agent of type "facility" nearby). Under some restrictions (e.g., some agents may move only a short distance), the objective is to make the minimum amount of movement so that afterwards the agents will form the desired configuration. Clearly, movement is a central component of this problem, but in a static sense—the intermediate configurations of the agents are immaterial (while moving to form the desired configuration, the agents can be in any configuration that they wish to, such as being placed all together on the same vertex, or being as far apart as possible). Demaine et al. (2014) presented, among other results, a dichotomy that concerns the fixed-parameter tractability of problems of this form parameterized by the number of agents (more precisely, of agents of a certain type) and another parameter related to the desired configuration.

2. Preliminaries

Let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. We denote the set $\mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$ by $\mathbb{N}_0^4$. For any $t \in \mathbb{N}$, we denote the set $\{1, 2, \ldots, t\}$ by $[t]$.

Graphs. Given a graph G, let $V(G)$ and $E(G)$ denote its vertex set and edge set, respectively. For a path P, let the *size* and the *length* of P denote the number of vertices and edges in P, respectively.

Let G be an undirected graph. For a vertex $v \in V(G)$, we denote the set of all the vertices adjacent to v in G by $N_G(v)$, i.e. $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$. The *edge contraction operation* for an edge $\{u, v\} \in E(G)$, is the addition of a new vertex w such that $N_G(w) = N_G(u) \cup N_G(v)$ and the deletion of the edge $\{u, v\}$ and the vertices u and v from G. For an edge $e \in E(G)$, we denote the graph obtained from G by contracting e by G/e. We denote the new vertex that was created by the contraction by v_e. The *edge subdivision operation* for an edge $\{u, v\} \in E(G)$, is the deletion of the edge $\{u, v\}$ from G and the addition of two new edges $\{u, w\}$ and $\{w, v\}$ for a new vertex w. For an undirected graph H, we say that *G can be obtained from H by subdividing edges*, if G can be derived from H by a sequence of edge subdivision operations. A *Hamiltonian cycle* in G is a cycle in G that visits every vertex of G exactly once. For other standard notations not explicitly defined here, we refer to the book by Diestel (2012).

For our negative result on kernelization, we make use of the Hamiltonian Cycle problem on grid graphs, defined as follows.

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

Given $k, \ell, r \in \mathbb{N}$, a *$(k \times \ell)$-$r$-grid* is a grid graph G with $V(G) = \{(x, y)|r \leq x \leq r + k - 1, r \leq y \leq r + \ell - 1\}$. Given a $(k \times \ell)$-r-grid G, the four *corner vertices* of G are $(r, r), (r, r + \ell - 1), (r + k - 1, r)$ and $(r + k - 1, r + \ell - 1)$. For any $k, \ell \in \mathbb{N}$ we denote $(k \times \ell)$-1-grid by $k \times \ell$-grid.

Given a graph G, the objective of the Hamiltonian Cycle problem is to decide whether there exists a Hamiltonian cycle in G. The Hamiltonian Cycle problem is known to be NP-complete even when restricted to grid graphs (Papadimitriou & Vazirani, 1984).

For our result on treewidth, we make use of the *treewidth* of a graph G, defined as follows.

Definition 2.2 (**Treewidth).** *A* tree decomposition *of a graph G is a tree T whose nodes, called* bags*, are labeled by subsets of vertices of G. For each vertex v the bags containing v must form a nonempty contiguous subtree of T, and for each edge uv at least one bag must contain both u and v. The* width *of the decomposition is one less than the maximum cardinality of any bag, and the* treewidth $\mathtt{tw}(G)$ *of G is the minimum width of any of its tree decompositions.*

Snake Game. Towards the definition of Snake Game, we begin by defining the notion of a configuration in this context.

Definition 2.3 (**(G, k)-Configuration).** *Let G be an undirected graph, and let $k \in \mathbb{N}$. A* (G, k)-configuration *is a tuple $(v_1, v_2, \ldots, v_k)$ where $v_i \in V(G)$ for every $1 \leq i \leq k$, which satisfies the following conditions.*

- *For every $1 \leq i \leq k - 1$, we have that $\{v_i, v_{i+1}\} \in E(G)$.*
- *For every $1 \leq i < j \leq k$, we have that $v_i \neq v_j$.*

Given a configuration conf, let $V(\mathsf{conf})$ denotes the set of its vertices. Intuitively, a (G, k)-configuration $(v_1, v_2, \ldots, v_k)$ is the sequence of vertices of a simple path on k vertices in G when traversed from one endpoint to the other; the path is termed a *snake*, and the vertices v_1 and v_k are termed the *head* and *tail* of the snake, respectively. For the sake of brevity, whenever G and k are clear from the context, we refer to a (G, k)-configuration simply as a *configuration.* We proceed to define how a snake is permitted to "move" from one position to another—that is, we define a *transition* (in one step) from one configuration to another.

Definition 2.4 (**1-Transition**). *Let G be an undirected graph, and let $k \in \mathbb{N}$. Let* $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$ *and* $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k)$ *be any two configurations. We say that the pair* $(\mathsf{conf}, \mathsf{conf}')$ *is a* 1-transition *if and only if the following conditions are satisfied.*

- *For every $1 \leq i \leq k-1$, $v'_1 \neq v_i$.*
- *For every $2 \leq i \leq k$, $v'_i = v_{i-1}$.*

We naturally extend the definition of a transition in one step to the definition of a transition in ℓ steps as follows.

Definition 2.5 (**ℓ-Transition**). *Let G be an undirected graph, and let $k, \ell \in \mathbb{N}$. Let* $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$ *and* $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k)$ *be any two configurations. We say that the pair* $(\mathsf{conf}, \mathsf{conf}')$ *is an* ℓ-transition *if there exists 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 $1 \leq i \leq \ell$, the pair* $(\mathsf{conf}_i, \mathsf{conf}_{i+1})$ *is a* 1-*transition. In that case, we also say that the tuple* $(\mathsf{conf}_1 = \mathsf{conf}, \mathsf{conf}_2, \ldots, \mathsf{conf}_{\ell+1} = \mathsf{conf}')$ *is an* ℓ-transition*.*

Before we define the Snake Game problem formally, we need one more definition concerning the reachability of one configuration from another, based on the definition of a transition.

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

We are now ready to give the formal definition of 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, we say that* $\mathbf{SG}$ *is a* No*-instance. By* solving *an instance of* Snake Game*, we mean that we correctly determine whether it is a* Yes*-instance or a* No*-instance.*

We now define a new auxiliary graph, the *ℓ-configuration graph*, which will be helpful throughout the paper.

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

We can also define reachability and feasibility of a "solution" (i.e. witness of a Yes-instance) for an instance of Snake Game via the 1-configuration graph by the following simple observation.

Observation 2.1. *Let G be an undirected graph, and let $k \in \mathbb{N}$. Let* conf *and* conf' *be any two configurations. Then* conf can reach conf' *if and only if there exists 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 exists a path from* init *to* fin *in the 1-configuration graph.*

Parameterized Complexity. A problem Π is a *parameterized* problem if each problem instance of Π is associated with a *parameter k*. For simplicity, we denote a problem instance of a parameterized problem Π as a pair (I, k) where the second argument is the parameter k associated with I. The main objective of the framework of Parameterized Complexity is to confine the combinatorial explosion in the running time of an algorithm for an NP-hard parameterized problem Π to depend only on k. Formally, we say that Π is *fixed-parameter tractable (FPT)* if any instance (I, k) of Π is solvable in time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is an arbitrary computable function of k. We remark that Parameterized Complexity also provides methods to show that a parameterized problem is unlikely to be FPT. The main technique is the one of parameterized reductions analogous to those employed in Classical Complexity. Here, the concept of W[1]-hardness replaces the one of NP-hardness.

A companion notion to that of fixed-parameter tractability is the one of a polynomial kernel. Formally, a parameterized problem Π is said to admit 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' (that is, (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 some polynomial that depends only on k. In case $\Pi' = \Pi$, we further say that Π admits a *polynomial kernel*. For more information on Parameterized Complexity, we refer the reader to books by Downey and Fellows (2013), Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk, and Saurabh (2015) and Fomin, Lokshtanov, Saurabh, and Zehavi (2018).

Non-Existence of a Polynomial Compression. Our proof of the "unlikely existence" of a polynomial compression (and hence also of a polynomial kernel) for Snake Game, even when restricted to grid graphs, relies on the well-known notion of cross-composition. We present this notion in a form sufficient for our proof.

Definition 2.9 (Cross-Composition). *A (not parameterized) problem* Π cross-composes *into a parameterized problem* Π' *if there exists a polynomial-time algorithm, called a* cross-composition*, that given instances $I_1, I_2, \ldots, I_t$ of Π for some $t \in \mathbb{N}$ that are of the same size s for some $s \in \mathbb{N}$, outputs an instance (I, k) of Π' such that the following conditions are satisfied.*

- *$k \leq p(s)$ for some polynomial p in s.*
- *(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 2.1. *(Bodlaender, Downey, Fellows, & Hermelin, 2009; Bodlaender, Jansen, & Kratsch, 2014) Let Π be an NP-hard (not parameterized) problem that cross-composes into a parameterized problem Π'. Then, Π' does not admit a polynomial compression, unless NP⊆coNP/poly.*

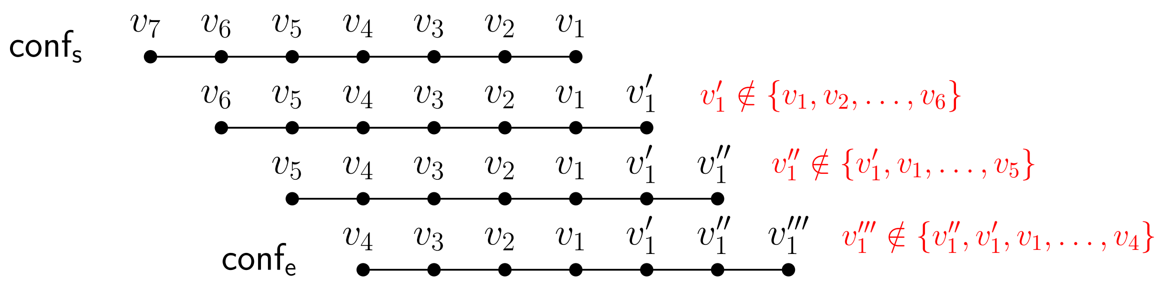

conf$_s$
v_7 v_6 v_5 v_4 v_3 v_2 v_1
v_6 v_5 v_4 v_3 v_2 v_1 v_1'
$v_1' \notin \{v_1, v_2, \ldots, v_6\}$
v_5 v_4 v_3 v_2 v_1 v_1' v_1''
$v_1'' \notin \{v_1', v_1, \ldots, v_5\}$
conf$_e$
v_4 v_3 v_2 v_1 v_1' v_1'' v_1'''
$v_1''' \notin \{v_1'', v_1', v_1, \ldots, v_4\}$

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

3. FPT Algorithm on General Graphs

In this section, we describe an FPT algorithm for the Snake Game problem. We first look at a naive algorithm that solves the Snake Game problem. The naive algorithm creates the 1-configuration graph, and looks for a path in this graph from the initial configuration to the final configuration. The problem with this approach is that the configuration graph might be huge, that is, possibly of size $\mathcal{O}(\binom{n}{k} \cdot k!)$. The idea of our algorithm presented in this section is to use color coding in order to sparsify the $(k-1)$-configuration graph. We create a family of coloring functions from the vertices of the graph G to the set of colors $[3k-2]$. Then, we build the sparse configuration graph by adding only those configurations that are colored in a specific way by any one of the coloring functions. We then prove that the sparse configuration graph is equivalent to the 1-configuration graph. That is, there exists a path from the initial configuration to the final configuration in the 1-configuration graph if and only if there exists a path from the initial configuration to the final configuration in the sparse configuration graph.

We begin by giving two conditions that identify when a pair of configurations is an ℓ-transition. These conditions will be useful throughout the section, and are stated in the following lemma. In this context, it may be helpful to refer to Figure 1, which shows two configurations $\mathsf{conf_s}$ and $\mathsf{conf_e}$ of length 6 where $(\mathsf{conf_s}, \mathsf{conf_e})$ is a 3-transition.

Lemma 3.1. *Let G be an undirected graph, and let $k, \ell \in \mathbb{N}$ such that $\ell \leq k$. Let $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$ and $\mathsf{conf}' = (v_1', v_2', \ldots, v_k')$ be any two configurations. Then, the pair $(\mathsf{conf}, \mathsf{conf}')$ is an ℓ-*transition *if and only if the following conditions are satisfied.*

- *For every $1 \leq i \leq \ell$, $v_i' \notin \{v_1, v_2, \ldots, v_{(k+i)-(\ell+1)}\}$.*
- *For every $\ell + 1 \leq i \leq k$, $v_i' = v_{i-\ell}$.*

Proof. We prove the statement by induction on ℓ.

Base case $(\ell = 1)$**.** When $\ell = 1$, the conditions are as follows:

- $v_1' \notin \{v_1, v_2, \ldots, v_{k-1}\}$.
- For every $2 \leq i \leq k$, $v_i' = v_{i-1}$.

By the definition of a 1-transition, two configurations form a 1-transition if and only if the above conditions are true, so the lemma is true for $\ell = 1$.

Inductive hypothesis. Suppose that the lemma is true for $\ell = t-1 \geq 1$.

Inductive step. We need to prove that the lemma is true for $\ell = t \geq 2$.
($\Rightarrow$) Let $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$ and $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k)$ be two configurations such that the pair $(\mathsf{conf}, \mathsf{conf}')$ is a t-transition. Then, by Definition 2.5, there exists a tuple of configurations $(\mathsf{conf}_1 = \mathsf{conf}, \mathsf{conf}_2, \ldots, \mathsf{conf}_t, \mathsf{conf}_{t+1} = \mathsf{conf}')$ such that for every $1 \leq i \leq t$, the pair $(\mathsf{conf}_i, \mathsf{conf}_{i+1})$ is a 1-transition. It follows that $(\mathsf{conf}, \mathsf{conf}_t)$ is a $(t-1)$-transition. Let $\mathsf{conf}_t = (w_1, w_2, \ldots, w_k)$. Then, by inductive hypothesis, the following conditions are true:

- For every $1 \leq i \leq t-1$, $w_i \notin \{v_1, v_2, \ldots, v_{(k+i)-t}\}$.
- For every $t \leq i \leq k$, $w_i = v_{i-t+1}$.

Also, $(\mathsf{conf}_t, \mathsf{conf}')$ is a 1-transition so by Definition 2.4, the following conditions are true:

- $v'_1 \notin \{w_1, w_2, \ldots, w_{k-1}\}$.
- For every $2 \leq i \leq k$, $v'_i = w_{i-1}$.

Combining the above conditions for $(\mathsf{conf}, \mathsf{conf}_t)$ and $(\mathsf{conf}_t, \mathsf{conf}')$, we get:

- For every $1 \leq i \leq t$, $v'_i \notin \{v_1, v_2, \ldots, v_{(k+i)-(t+1)}\}$.
- For every $t+1 \leq i \leq k$, $v'_i = v_{i-t}$.

($\Leftarrow$) Let $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$ and $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k)$ be two configurations such that the following conditions are satisfied:

- For every $1 \leq i \leq t$, $v'_i \notin \{v_1, v_2, \ldots, v_{(k+i)-(t+1)}\}$.
- For every $t+1 \leq i \leq k$, $v'_i = v_{i-t}$.

Let $\mathsf{conf}'' = (w_1, w_2, \ldots, w_k)$ be a configuration defined as follows:

- For every $1 \leq i \leq k-1$, $w_i = v'_{i+1}$.
- $w_k = v_{k-t+1}$.

From Definition 2.3, we know that for every $2 \leq i \leq k$, $v'_1 \neq v'_i$. So, from the definition of conf'', we get the following conditions for the pair $(\mathsf{conf}'', \mathsf{conf}')$:

- $v'_1 \notin \{w_1, w_2, \ldots, w_{k-1}\}$.
- For every $2 \leq i \leq k$, $v'_i = w_{i-1}$.

By Definition 2.4, $(\mathsf{conf}'', \mathsf{conf}')$ is a 1-transition. Also, by the conditions for $(\mathsf{conf}, \mathsf{conf}')$ and the definition of conf'', we get the following conditions for the pair $(\mathsf{conf}, \mathsf{conf}'')$:

- For every $1 \leq i \leq t-1$, $w_i \notin \{v_1, v_2, \ldots, v_{(k+i)-t}\}$.

- For every $t \leq i \leq k$, $w_i = v_{i-t+1}$.

By inductive hypothesis, $(\mathsf{conf}, \mathsf{conf}'')$ is a $(t-1)$-transition. Moreover, we already know that $(\mathsf{conf}'', \mathsf{conf}')$ is a 1-transition. So, by Definition 2.5, we get that $(\mathsf{conf}, \mathsf{conf}')$ is a t-transition. □

The following corollary about $(k-1)$-transitions follows directly from the above lemma.

Corollary 3.1. *Let G be an undirected graph, and $k \in \mathbb{N}$. Let $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$ and $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k)$ be any two configurations. Then, the pair $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-*transition *if and only if the following conditions are satisfied.*

- *For every $1 \leq i \leq k-1$, $v'_i \notin \{v_1, v_2, \ldots, v_i\}$.*
- *$v'_k = v_1$.*

In turn, the following observation about testing whether a pair of configurations is a $(k-1)$-transition follows directly from the above corollary.

Observation 3.1. *Let G be an undirected graph and $k \in \mathbb{N}$. Let conf and conf' be any two configurations. Then, we can test whether $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-transition in time $\mathcal{O}(k^2)$ by checking the conditions given in Corollary 3.1 for every vertex of conf and conf'.*

Observe that the size of the $(k-1)$-configuration graph is potentially huge. We overcome this difficulty by utilizing the technique of color-coding (Alon et al., 1995) to sparsify the configuration graph. We need the following definitions to describe the technique.

Definition 3.1 (Splitter). *Let $n, m, t \in \mathbb{N}$ such that $m \leq t$. An (n, m, t)-*splitter *$\mathcal{F}$ is a family of functions from $[n]$ to $[t]$ such that for every set $W \subseteq [n]$ of size at most m, there exists a function $f \in \mathcal{F}$ that is injective on W.*

Definition 3.2 (Permuter). *Let $n, t \in \mathbb{N}$ and S be a set of size n. An (S, t)-*permuter *$\mathcal{G}$ is a family of functions from S to $[t]$ such that for every ordered set $W \subseteq S$ of size at most t and for any $i, j \in \mathbb{N}$ such that $i \leq j \leq t$ and $j - i + 1 = |W|$, there exists a function $g \in \mathcal{G}$ which maps W to the ordered set $\{i, i+1, \ldots, j\}$.*

An efficient construction for an (S, t)-permuter easily follows from the well-known efficient construction of splitters (Naor, Schulman, & Srinivasan, 1995) as follows.

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

Proof. We can construct an (S, t)-permuter of size $t^{\mathcal{O}(t)} \log n$ as follows. Let $\mathcal{F}_1$ be an (n, t, t)-splitter. Let $\mathcal{F}_2$ be the family of all permutation functions from $[t]$ to $[t]$. Now consider the family of functions $\mathcal{F}^*$ obtained by composition of $\mathcal{F}_1$ with $\mathcal{F}_2$, i.e. $\mathcal{F}^* = \{f_2 \circ f_1 : f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2\}$. Here, $f_2 \circ f_1$ is the composition of the two functions obtained by performing f_1 first and then f_2, i.e. $(f_2 \circ f_1)(x) = f_2(f_1(x))$. Let $S = \{s_1, s_2, \ldots, s_n\}$ and $\mathcal{F}^* = \{f^*_1, f^*_2, \ldots, f^*_{|\mathcal{F}^*|}\}$. Now consider the family of functions $\mathcal{G} = \{g_1, g_2, \ldots, g_{|\mathcal{F}^*|}\}$ such that for every $1 \leq j \leq |\mathcal{F}^*|$ and $1 \leq i \leq n$, $g_j(s_i) = f^*_j(i)$. It is easy to see that $\mathcal{G}$ is our required (S, t)-permuter. We now argue about the size of $\mathcal{G}$ and the time taken by the construction. Naor *et al.* (1995) gave a construction of $\mathcal{F}_1$ of size $e^t t^{\mathcal{O}(\log t)} \log n$ in time $e^t t^{\mathcal{O}(\log t)} n \log n$, and the size of $\mathcal{F}_2$ is exactly $t!$. Thus the size of $\mathcal{G}$ is $(e^t t^{\mathcal{O}(\log t)} \log n) \cdot t! = t^{\mathcal{O}(t)} \log n$. As it is possible to list the image of a function in linear time in the size of the domain, we can compute $\mathcal{G}$ in time $t^{\mathcal{O}(t)} n \log n$. □

Given three configurations conf, conf' and conf'', we now define a special ordering, called *triplet order*, on the vertices of the triplet (conf, conf', conf''). Note that an example is given below the definition.

Definition 3.3 (**Triplet Order**). *Let G be an undirected graph, and let $k \in \mathbb{N}$. 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 any three configurations. Let $S = \{v_k, v_{k-1}, \ldots, v_1, v'_k, v'_{k-1}, \ldots, v'_1, v''_k, v''_{k-1}, \ldots, v''_1\}$ be the ordered multi-set defined on the vertices of triplet* (conf, $\mathsf{conf}', \mathsf{conf}''$)*. Then, the* triplet order *of* $(\mathsf{conf}, \mathsf{conf}', \mathsf{conf}'')$ *is the ordered set obtained from S by first removing the vertices of* conf *and* conf'' *that are common to* conf'*, and then removing the vertices of* conf *that are common to* conf''.

For example, let $\mathsf{conf} = (v_1, v_2, v_3, v_4, v_5, v_6, v_7)$, $\mathsf{conf}' = (v'_1, v'_2 = v_6, v'_3 = v_7, v'_4, v'_5 = v_1, v'_6 = v_2, v'_7 = v_3)$ and $\mathsf{conf}'' = (v''_1, v''_2, v''_3 = v_4, v''_4, v''_5, v''_6, v''_7 = v'_1)$ be three configurations of length 6. Then, the triplet order W of $(\mathsf{conf}, \mathsf{conf}', \mathsf{conf}'')$ is $\{v_5, v'_7, v'_6, v'_5, v'_4, v'_3, v'_2, v'_1, v''_6, v''_5, v''_4, v''_3, v''_2, v''_1\}$ (see Figure 2). Note that W is an ordered set that contains every distinct vertex of S exactly once.

The following lemma about triplet order will help us to reduce the size of the $(k-1)$-configuration graph.

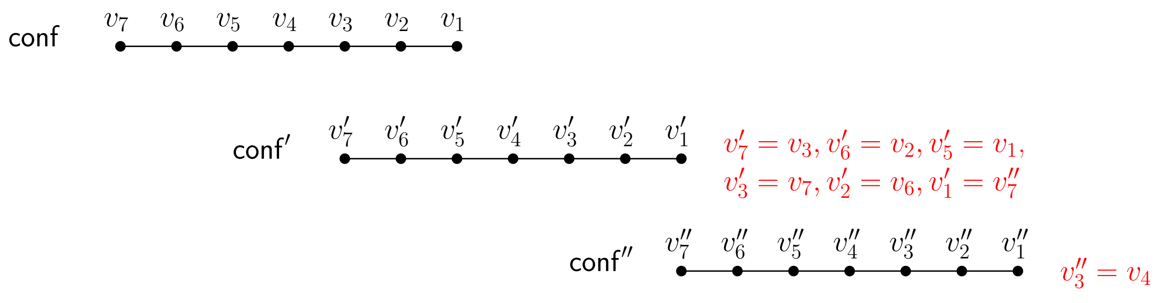
Lemma 3.3. *Let G be an undirected graph, and $k, \ell, r \in \mathbb{N}$ such that $\ell, r \leq k-1$. Let* $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$, $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k)$ *and* $\mathsf{conf}'' = (v''_1, v''_2, \ldots, v''_k)$ *be three configurations such that* $(\mathsf{conf}, \mathsf{conf}')$ *is an ℓ-transition and* $(\mathsf{conf}', \mathsf{conf}'')$ *is an r-transition. Let W be the triplet order of* $(\mathsf{conf}, \mathsf{conf}', \mathsf{conf}'')$ *and $\mathcal{G}$ be a $(V(G), 3k-2)$-permuter. Then, there exists a function $f \in \mathcal{G}$ that satisfies the following conditions (see Figure 2):*

(i) W is mapped to an ordered set $\{i, i+1, \ldots, k, k+1, \ldots, 2k-1, \ldots, j\}$ such that $1 \leq i \leq j \leq 3k-2$, $j - i + 1 = |W|$ and for every $1 \leq a \leq k$, $f(v'_a) = 2k - a$.

(ii) All the vertices in W have different images in f.

(iii) If there exists a configuration $\mathsf{conf}^* = (w_1 = v'_1, w_2, \ldots, w_k = v'_k)$ *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* $(\mathsf{conf}, \mathsf{conf}^*)$ *is an ℓ-transition and* $(\mathsf{conf}^*, \mathsf{conf}'')$ *is an r-transition.*

Proof. We first prove Conditions (i) and (ii). As $(\mathsf{conf}, \mathsf{conf}')$ is an ℓ-transition, by Lemma 3.1, conf and conf' can have at most ℓ vertices in common. Similarly, conf' and conf'' can have at most r vertices in common. As $\ell, r \leq k-1$, from the definition of triplet order, $|W| \leq \ell + k + r \leq 3k - 2$. Also $W \subseteq V(G)$, so by Definition 3.2, there exists a function in $\mathcal{G}$ that maps W to an ordered set $\{i, i+1, \ldots, j\}$ for any $i, j \in \mathbb{N}$ such that $i \leq j \leq 3k-2$ and $j - i + 1 = |W|$. As $|W| \geq k$, we can choose i and j such that there exists a function


conf
v_7 v_6 v_5 v_4 v_3 v_2 v_1
conf′
v'_7 v'_6 v'_5 v'_4 v'_3 v'_2 v'_1
$v'_7 = v_3, v'_6 = v_2, v'_5 = v_1,$
$v'_3 = v_7, v'_2 = v_6, v'_1 = v''_7$
conf″
v''_7 v''_6 v''_5 v''_4 v''_3 v''_2 v''_1
$v''_3 = v_4$

$$W = \{v_5, v'_7, v'_6, \dots, v'_1, v''_6, v''_5, \dots v''_1\},\ f(W) = \{6, 7, 8, \dots, 13, 14, 15, \dots, 19\}$$

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

$g \in \mathcal{G}$ that maps W to $\{i, i+1, \dots, k, k+1, \dots, 2k-1, \dots, j\}$ where for every $1 \le a \le k$, $g(v'_a) = 2k - a$. It is easy to see that all the vertices in W have different images in g. Thus by taking $f = g$, Conditions (i) and (ii) are satisfied.

We now prove Condition (iii). Suppose that there exists a configuration $\mathsf{conf}^* = (w_1 = v'_1, w_2, \dots, w'_k = v'_k)$ other than conf' such that **(1)** for every $1 \le a \le k$, $f(w_a) = 2k - a$, **(2)** for every $\ell + 1 \le a \le k$, $w_a = v_{a-\ell}$, and **(3)** for every $r + 1 \le a \le k$, $v''_a = w_{a-r}$. We prove that $(\mathsf{conf}, \mathsf{conf}^*)$ is an ℓ-transition. The proof of $(\mathsf{conf}^*, \mathsf{conf}'')$ being an r-transition is similar.

By the way of contradiction, assume that $(\mathsf{conf}, \mathsf{conf}^*)$ is not an ℓ-transition. Notice that for every $\ell + 1 \le a \le k$, $w_a = v_{a-\ell}$. So, from Lemma 3.1, there must exist a vertex w_b such that $w_b = v_c$ for some $1 \le b \le \ell$ and $1 \le c \le (k + b) - (\ell + 1)$. In particular, $f(v_c) = f(w_b) = 2k - b$. Moreover, $f(v'_b) = 2k - b$. As W contains every distinct vertex of $V(\mathsf{conf}) \cup V(\mathsf{conf}') \cup V(\mathsf{conf}'')$ exactly once, by Condition (ii), $v'_b = v_c$. As $(\mathsf{conf}, \mathsf{conf}')$ is an ℓ-transition, from Lemma 3.1, $v'_b \notin \{v_1, v_2, \dots, v_{(k+b)-(\ell+1)}\}$. In particular, $v'_b \ne v_c$ which is a contradiction. □

We now give the following lemma, about finding a particular labeled path in a vertex labeled graph, which will be helpful throughout the section.

Lemma 3.4. *Let $k, r, t \in \mathbb{N}$. Then, there exists an algorithm that, given an undirected graph G, a labeling function f from $V(G)$ to $[t]$ and any two vertices u and v in $V(G)$, runs in time $\mathcal{O}(|V(G)| + |E(G)|)$ and determines whether there exists a (simple) path $P = (w_1 = u, w_2, \dots, w_k = v)$ of length $k - 1$ between u and v in G such that for every $1 \le i \le k$, $f(w_i) = r + k - i$, and if the path exists, then it finds such a path.*

Proof. We first construct a new directed graph G' from G as follows. Initialize $V(G') = V(G)$ and $E(G') = \emptyset$. For every edge $\{x, y\} \in E(G)$ such that $|f(x) - f(y)| = 1$: if $f(x) > f(y)$, then add the arc (x, y) to $E(G')$; otherwise, add the arc (y, x) to $E(G')$. It is easy to see that, if there exists a (simple) path $P = (w_1 = u, w_2, \dots, w_k = v)$ of length $k - 1$ between u and v in G such that for every $1 \le i \le k$, $f(w_i) = r + k - i$, then P also exists in G' as a directed path from u to v. Note that in G', every arc is directed from a

Algorithm 1: Construction of a $(k-1)$-sparse configuration graph.

```
function sparseConfigurationGraph($\mathbf{SG}, k$);
let $\mathcal{C}'$ be an empty graph;
initialize $V(\mathcal{C}') = \{\mathsf{init}, \mathsf{fin}\}$;
construct a $(V(G), 3k-2)$-permuter $\mathcal{G}$ using Lemma 3.2;
for each $g \in \mathcal{G}$ do
    for each pair of vertices $(u,v) \in V(G) \times V(G)$ such that $u \neq v$ do
        if there is a (simple) path $P = (w_1 = u, w_2, \ldots, w_k = v)$ of length $k-1$
          between $u$ and $v$ in $G$ such that for every $1 \leq a \leq k$, $g(w_a) = 2k - a$ then
            add $P$ as a configuration with $u$ as its head to $V(\mathcal{C}')$;
        end
    end
end
for every triplet of distinct vertices $(u,v,w) \in V(G) \times V(G) \times V(G)$ do
    for every pair of vertices
      $(\mathsf{conf} = (x_1 = v, x_2, \ldots, x_k = u), \mathsf{conf}' = (x'_1 = w, x'_2, \ldots,$
      $x'_k = v)) \in V(\mathcal{C}') \times V(\mathcal{C}')$ do
        if $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-transition in $G$ then
            add the ordered pair $(\mathsf{conf}, \mathsf{conf}')$ as a directed edge in $E(\mathcal{C}')$;
        end
    end
end
return $(\mathcal{C}', \mathcal{G})$.
```

vertex labeled $i+1$ to a vertex labeled i for some $1 \leq i \leq t-1$ so if $f(u) = r+k-1$ and $f(v) = r$, then any path from u to v in G' will satisfy the required property. So we can perform breadth-first search (BFS) in G' starting from u and return the shortest path from u to v in G', if it exists. As we can construct G' in time $\mathcal{O}(|V(G)| + |E(G)|)$ and the BFS runs in $\mathcal{O}(|V(G')| + |E(G')|)$, the lemma follows. □

Let $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin} \rangle$ be an instance of Snake Game. We now give a procedure (Algorithm 1) to construct a new graph, called a $(k-1)$-*sparse configuration graph* of $\mathbf{SG}$, based on a $(V(G), 3k-2)$-permuter and Lemma 3.3. We first give an outline of the procedure.

We initialize $\mathcal{C}'$ to be a new configuration graph whose vertex set contains init and fin and whose edge set is empty. Then, we construct a $(V(G), 3k-2)$-permuter $\mathcal{G}$. For each function $g \in \mathcal{G}$, we take every pair of distinct vertices (u,v) of G and check whether there exists a path P between u and v of length $k-1$ such that if we traverse the path P from u to v, the ordered set of images of the vertices of P under g is $\{2k-1, 2k-2, \ldots, k\}$. If such a path P exists, which is checked using Lemma 3.4, then we add P as a configuration with u as its head to the vertex set of $\mathcal{C}'$. After finishing this for loop, we take every pair of distinct vertices $(\mathsf{conf}, \mathsf{conf}')$ of $\mathcal{C}'$ and add an edge from conf to conf' if $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-transition, which is checked using Observation 3.1. In the end, we return $\mathcal{C}'$ and $\mathcal{G}$.

We define the notion of a $(k-1)$-*sparse configuration graph* as follows.

Definition 3.4. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game*. Then, any graph returned by Algorithm 1 is called a* $(k-1)$-sparse configuration graph *of* $\mathbf{SG}$*.*

We first give the following lemma about the size of a $(k-1)$-sparse configuration graph and the running time of Algorithm 1.

Lemma 3.5. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game*. Then, Algorithm 1 runs in time* $k^{\mathcal{O}(k)} n^3 \log^2 n$ *and returns a* $(k-1)$*-sparse configuration graph of* $\mathbf{SG}$ *with* $k^{\mathcal{O}(k)} n^2 \log n$ *vertices and* $k^{\mathcal{O}(k)} n^3 \log^2 n$ *arcs.*

Proof. Let $|V(G)| = n$, $|E(G)| = m$, $\mathcal{C}'$ be a $(k-1)$-sparse configuration graph and $\mathcal{G}$ be a $(V(G), 3k-2)$-permuter constructed by Algorithm 1. From Corollary 5.1, we implicitly assume that $m = k^{\mathcal{O}(1)} n$. For each function $g \in \mathcal{G}$ and every pair of distinct vertices of G, the algorithm adds at most one vertex to $\mathcal{C}'$. Also for every triplet of distinct vertices of G, the algorithm adds an arc between every two vertices conf and conf' of $\mathcal{C}'$, such that the head of conf and tail of conf' is the same vertex and $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-transition. By Corollary 3.1, we know that if the pair $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-transition then the head of conf and tail of conf' should be the same vertex. As the algorithm performs this check for every triplet of distinct vertices of G, every pair $(\mathsf{conf}, \mathsf{conf}') \in V(\mathcal{C}') \times V(\mathcal{C}')$ that is a $(k-1)$-transition is added to $\mathcal{C}'$. For every pair of distinct vertices of G, the algorithm adds at most $|\mathcal{G}|$ configurations so the number of pairs of vertices of $\mathcal{C}'$ satisfying the condition given in line 13 is at most $|\mathcal{G}|^2$. By Lemma 3.2, one can construct a $(V(G), t)$-permuter of size $t^{\mathcal{O}(t)} \log n$ in time $t^{\mathcal{O}(t)} n \log n$. So, $|V(\mathcal{C}')| = k^{\mathcal{O}(k)} n^2 \log n$ and $|E(\mathcal{C}')| = k^{\mathcal{O}(k)} n^3 \log^2 n$. By Observation 3.1, for a given pair of configurations $(\mathsf{conf}, \mathsf{conf}')$, we can test whether $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-transition in time $\mathcal{O}(k^2)$. Also, by Lemma 3.4, we can find a path P between u and v satisfying the condition given in line 7 of Algorithm 1 in time $\mathcal{O}(n+m)$. So, the running time of Algorithm 1 is $(k^{\mathcal{O}(k)} n \log n + (k^{\mathcal{O}(k)} \log n) \cdot n^2 \cdot (n+m) + n^3 \cdot (k^{\mathcal{O}(k)} \log^2 n) \cdot k^2) = k^{\mathcal{O}(k)} n^3 \log^2 n$. ☐

Observe that, unlike the $(k-1)$-configuration graph, a $(k-1)$-sparse configuration graph may not be unique for a snake game $\mathbf{SG}$ even if the permuter is fixed. Indeed, given a function of the permuter and a pair of vertices of G, there may be more than one path that satisfies the condition given in line 7 of Algorithm 1 but we only add one of them (chosen arbitrarily) to our $(k-1)$-sparse configuration graph.

The following lemma concerns the relationship between the $(k-1)$-configuration graph and a $(k-1)$-sparse configuration graph.

Lemma 3.6. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game*. Let* $\mathcal{C}$ *be the* $(k-1)$*-configuration graph and* $\mathcal{C}'$ *be a* $(k-1)$*-sparse configuration graph of* $\mathbf{SG}$*. Let* $\mathsf{conf}, \mathsf{conf}', \mathsf{conf}'' \in V(\mathcal{C})$ *such that both* $(\mathsf{conf}, \mathsf{conf}')$ *and* $(\mathsf{conf}', \mathsf{conf}'')$ *are* $(k-1)$*-transitions. Then:*

(i) $\mathcal{C}'$ *is an induced subgraph of* $\mathcal{C}$*.*

(ii) There exists a configuration $\mathsf{conf}^* \in V(\mathcal{C}')$ *such that both* $(\mathsf{conf}, \mathsf{conf}^*)$ *and* $(\mathsf{conf}^*, \mathsf{conf}'')$ *are also* $(k-1)$*-transitions.*

Proof. We first prove that $V(\mathcal{C}') \subseteq V(\mathcal{C})$. As $V(\mathcal{C})$ is the set of all (G,k)-configurations, it suffices to show that every vertex in $V(\mathcal{C}')$ is a (G,k)-configuration. Clearly, init and fin are (G,k)-configurations. Let conf be a vertex in $V(\mathcal{C}')$ other than init and fin. Then, by construction, conf is a simple path of length $k-1$, with a head and a tail, in G. So, by Definition 2.3, conf is a (G,k)-configuration thus indeed, $V(\mathcal{C}') \subseteq V(\mathcal{C})$. We now prove that $\mathcal{C}'$ is an induced subgraph of $\mathcal{C}$. To this end, let conf and conf' be two configurations in $\mathcal{C}$. Then, there is an edge from conf to conf' in $\mathcal{C}$ if and only if $(\mathsf{conf}, \mathsf{conf}')$ is a $(k-1)$-transition. Moreover, if conf and conf' belong to $\mathcal{C}'$ then the same condition is true in $\mathcal{C}'$ by construction. In turn, this proves that $\mathcal{C}'$ is indeed an induced subgraph of $\mathcal{C}$.

We now prove Condition (ii). Let $\mathsf{conf} = (v_1, v_2, \ldots, v_k)$, $\mathsf{conf}' = (v'_1, v'_2, \ldots, v'_k = v_1)$ and $\mathsf{conf}'' = (v''_1, v''_2, \ldots, v''_k = v'_1)$. Let W be the triplet order of $(\mathsf{conf}, \mathsf{conf}', \mathsf{conf}'')$ and $\mathcal{G}$ be a $(V(G), 3k-2)$-permuter. As both $(\mathsf{conf}, \mathsf{conf}')$ and $(\mathsf{conf}', \mathsf{conf}'')$ are $(k-1)$-transitions, by Condition (i) of Lemma 3.3, there exists a function $f \in \mathcal{G}$ that maps W to an ordered set $\{i, i+1, \ldots, k, k+1, \ldots, 2k-1, \ldots, j\}$ such that $1 \le i \le j \le 3k-2$, $j - i + 1 = |W|$ and for every $1 \le a \le k$, $f(v'_a) = 2k - a$. Let prop be the property of the path P stated in line 7 of Algorithm 1. So, when Algorithm 1 considers $g = f, u = v'_1, v = v'_k$, conf' satisfies the property prop. As there can be more than one path between v'_1 and v'_k that satisfies the property prop, Algorithm 1 either adds conf' or any other path between v'_1 and v'_k satisfying the property prop. If the algorithm adds conf', then $\mathsf{conf}^* = \mathsf{conf}'$ and we are done.

Otherwise, let $\widehat{\mathsf{conf}} = (w_1 = v'_1, w_2, \ldots, w_k = v'_k)$ be the configuration added by the algorithm. As $\widehat{\mathsf{conf}}$ satisfies the property prop, for every $1 \le a \le k$, $g(w_a) = 2k - a$. Also, observe that $w_k = v'_k = v_1$ and $v''_k = v'_1 = w_1$, so by Condition (iii) of Lemma 3.3, both $(\mathsf{conf}, \widehat{\mathsf{conf}})$ and $(\widehat{\mathsf{conf}}, \mathsf{conf}'')$ are $(k-1)$-transitions. So, we can take $\mathsf{conf}^* = \widehat{\mathsf{conf}}$ and we are done. □

We now give the following lemma which relates paths in the $(k-1)$-configuration graph and a $(k-1)$-sparse configuration graph.

Lemma 3.7. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin} \rangle$ *be an instance of* Snake Game *and* $t \in \mathbb{N}$. *Let* $\mathcal{C}$ *be the* $(k-1)$*-configuration graph and* $\mathcal{C}'$ *be a* $(k-1)$*-sparse configuration graph of* $\mathbf{SG}$. *Let* $\mathsf{conf}_s, \mathsf{conf}_e \in V(\mathcal{C})$. *Then, there exists a path* $P = (\mathsf{conf}_1 = \mathsf{conf}_s, \mathsf{conf}_2, \ldots, \mathsf{conf}_{t+1} = \mathsf{conf}_e)$ *of length* t *from* conf_s *to* conf_e *in* $\mathcal{C}$ *if and only if there exists a path* $P' = (\mathsf{conf}'_1 = \mathsf{conf}_s, \mathsf{conf}'_2, \ldots, \mathsf{conf}'_t, \mathsf{conf}'_{t+1} = \mathsf{conf}_e)$ *of length* t *from* conf_s *to* conf_e *in* $\mathcal{C}$ *such that* $\mathsf{conf}'_i \in V(\mathcal{C}')$ *for every* $2 \le i \le t$.

Proof. We first prove the if part. The if part is straightforward: from Condition (i) of Lemma 3.6, $V(\mathcal{C}') \subseteq V(\mathcal{C})$ and $E(\mathcal{C}') \subseteq E(\mathcal{C})$, so take $P = P'$.

We now prove the only if part. Let $P = (\mathsf{conf}_1 = \mathsf{conf}_s, \mathsf{conf}_2, \ldots, \mathsf{conf}_{t+1} = \mathsf{conf}_e)$ be a path of length t from conf_s to conf_e in $\mathcal{C}$. As P is a path in $\mathcal{C}$ so, for every $1 \le i \le t$, $(\mathsf{conf}_i, \mathsf{conf}_{i+1})$ is a $(k-1)$-transition. Now consider the triplet $(\mathsf{conf}_1, \mathsf{conf}_2, \mathsf{conf}_3)$. From Condition (ii) of Lemma 3.6, there exists a configuration $\mathsf{conf}'_2 \in V(\mathcal{C}')$ such that both $(\mathsf{conf}_1, \mathsf{conf}'_2)$ and $(\mathsf{conf}'_2, \mathsf{conf}_3)$ are $(k-1)$-transitions. Now we construct a new path P^* from P by replacing conf_2 with conf'_2. It is easy to see that P^* is also a path in $\mathcal{C}$. So, by iteratively taking triplet $(\mathsf{conf}_i, \mathsf{conf}_{i+1}, \mathsf{conf}_{i+2})$ for every $1 \le i \le (t-1)$ and applying this transformation, we get a new path $P' = (\mathsf{conf}'_1 = \mathsf{conf}_s, \mathsf{conf}'_2, \ldots, \mathsf{conf}'_{t+1} = \mathsf{conf}_e)$ of length t in $\mathcal{C}$ such that $\mathsf{conf}'_i \in V(\mathcal{C}')$ for every $2 \le i \le t$. □

By putting $\mathsf{conf}_s = \mathsf{init}, \mathsf{conf}_e = \mathsf{fin}$ in Lemma 3.7, we have the following simple corollary.

Corollary 3.2. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game *and* $t \in \mathbb{N}$. *Let* $\mathcal{C}$ *be the* $(k-1)$*-configuration graph and* $\mathcal{C}'$ *be a* $(k-1)$*-sparse configuration graph of* $\mathbf{SG}$. *Then, there exists a path of length* t *from* init *to* fin *in* $\mathcal{C}$ *if and only if there exists a path of length* t *from* init *to* fin *in* $\mathcal{C}'$

Given an instance of Snake Game $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$, we now show how we use its $(k-1)$-configuration graph to solve it. By Observation 2.1, we know that $\mathbf{SG}$ is a Yes-instance if and only if there exists a path from init to fin in the 1-configuration graph. It is easy to see that every path of length $(k-1)$ between two configurations in the 1-configuration graph is compressed to a path of length 1 in the $(k-1)$-configuration graph. If the answer to an instance of Snake Game is Yes, then there may or may not exist a path from init to fin in the 1-configuration graph whose length is a multiple of $(k-1)$. We first consider the case where the length of the path is a multiple of $(k-1)$ (Section 3.1), and then extend the result for the case where the length of the path is not a multiple of $(k-1)$ (Section 3.2).

3.1 When the Length of the Path is a Multiple of $(k-1)$

In this subsection, we consider the case where the length of the path from init to fin in the 1-configuration graph is a multiple of $(k-1)$. We give the following lemma which relates paths in the 1-configuration graph and the $(k-1)$-configuration graph whose lengths are multiples of $(k-1)$.

Lemma 3.8. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game *and* $t \in \mathbb{N}$. *Let* $\mathcal{C}$ *be the* $(k-1)$*-configuration graph of* $\mathbf{SG}$. *Let* conf_s *and* conf_e *be any two configurations of* G. *Then, there exists a path of length* $t(k-1)$ *from* conf_s *to* conf_e *in the* 1*-configuration graph if and only if there exists a path of length* t *from* conf_s *to* conf_e *in* $\mathcal{C}$.

Proof. ($\Rightarrow$) Suppose that there exists a path $P = (\mathsf{conf}_1 = \mathsf{conf}_s, \ldots, \mathsf{conf}_k, \ldots, \mathsf{conf}_{2k-1}, \ldots, \mathsf{conf}_{t(k-1)+1} = \mathsf{conf}_e)$ of length $t(k-1)$ from conf_s to conf_e in the 1-configuration graph. For every $i = 1 + r(k-1)$ where $0 \leq r \leq (t-1)$, we replace the subpaths $(\mathsf{conf}_i, \mathsf{conf}_{i+1}, \ldots, \mathsf{conf}_{i+k-1})$ of length $(k-1)$ in P with $(\mathsf{conf}_i, \mathsf{conf}_{i+k-1})$ iteratively to obtain a new path $P' = (\mathsf{conf}'_1 = \mathsf{conf}_s, \mathsf{conf}'_2 = \mathsf{conf}_k, \mathsf{conf}'_3 = \mathsf{conf}_{2k-1}, \ldots, \mathsf{conf}'_{t+1} = \mathsf{conf}_e)$. Observe that every pair $(\mathsf{conf}'_i, \mathsf{conf}'_{i+1})$ in P' is a $(k-1)$-transition, so P' is a path of length t from conf_s to conf_e in $\mathcal{C}$.
($\Leftarrow$) Suppose that there exists a path $P = (\mathsf{conf}_1 = \mathsf{conf}_s, \mathsf{conf}_2, \ldots, \mathsf{conf}_{t+1} = \mathsf{conf}_e)$ of length t from conf_s to conf_e in $\mathcal{C}$. As every arc $(\mathsf{conf}_i, \mathsf{conf}_{i+1})$ in path P is a $(k-1)$-transition, we can replace every arc in P with the corresponding path of length $(k-1)$. By applying this transformation, we get a new path $P' = (\mathsf{conf}'_1 = \mathsf{conf}_s, \ldots, \mathsf{conf}'_k = \mathsf{conf}_2, \ldots, \mathsf{conf}'_{2k-1} = \mathsf{conf}_3, \ldots, \mathsf{conf}'_{t(k-1)+1} = \mathsf{conf}_e)$ of length $t(k-1)$ where every pair $(\mathsf{conf}'_i, \mathsf{conf}'_{i+1})$ in P' is a 1-transition, so P' is a path from conf_s to conf_e in the 1-configuration graph. ☐

By putting $\mathsf{conf}_s = \mathsf{init}$ and $\mathsf{conf}_e = \mathsf{fin}$ in Lemma 3.8, we have the following simple corollary.

Corollary 3.3. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game *and* $t \in \mathbb{N}$*. Let* $\mathcal{C}$ *be the* $(k-1)$*-configuration graph of* $\mathbf{SG}$*. Then, there exists a path of length* $t(k-1)$ *from* init *to* fin *in the* 1*-configuration graph if and only if there exists a path of length* t *from* init *to* fin *in* $\mathcal{C}$*.*

The combination of Corollaries 3.2 and 3.3 gives us the following observation.

Observation 3.2. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game *and* $t \in \mathbb{N}$*. Let* $\mathcal{C}'$ *be a* $(k-1)$*-sparse configuration graph of* $\mathbf{SG}$*. Then, there exists a path of length* $t(k-1)$ *from* init *to* fin *in the* 1*-configuration graph if and only if there exists a path of length* t *from* init *to* fin *in* $\mathcal{C}'$*.*

3.2 When the Length of the Path is not a Multiple of $(k-1)$

We now consider the case where the length of the path from init to fin in the 1-configuration graph is not a multiple of $(k-1)$. Let $r \in \mathbb{N}$ such that $r < k-1$. Let $\mathcal{C}$ be the $(k-1)$-configuration graph and $\mathcal{C}'$ be a $(k-1)$-sparse configuration graph of $\mathbf{SG}$. Let $P = (\mathsf{conf}_1 = \mathsf{init}, \ldots, \mathsf{conf}_k, \ldots, \mathsf{conf}_{2k-1}, \ldots, \mathsf{conf}_{t(k-1)+(r+1)} = \mathsf{fin})$ be a path from init to fin in the 1-configuration graph. Note that the length of the path P is $t(k-1)+r$, which is not a multiple of $k-1$. Consider the subpath $P' = (\mathsf{conf}_1 = \mathsf{init}, \ldots, \mathsf{conf}_k, \ldots, \mathsf{conf}_{2k-1}, \ldots, \mathsf{conf}_{t(k-1)+1})$ of P in the 1-configuration graph. By Lemmas 3.7 and 3.8, there exists a path $P'' = (\mathsf{conf}'_1 = \mathsf{init}, \mathsf{conf}'_2, \mathsf{conf}'_3, \ldots, \mathsf{conf}'_{t+1} = \mathsf{conf}_{t(k-1)+1})$ of length t in $\mathcal{C}$ such that $\mathsf{conf}'_i \in V(\mathcal{C}')$ for every $1 \leq i \leq t$.

Now, consider the path $P^* = (\mathsf{conf}'_1 = \mathsf{init}, \mathsf{conf}'_2, \ldots, \mathsf{conf}'_{t+1}, \mathsf{conf}'_{t+2} = \mathsf{fin})$ defined on the vertex set $V = V(P'') \cup \{\mathsf{fin}\}$. Observe that, in P^*, all the vertices except possibly conf'_{t+1} belong to $V(\mathcal{C}')$. For every $1 \leq i \leq t$, $(\mathsf{conf}'_i, \mathsf{conf}'_{i+1})$ is a $(k-1)$-transition, but possibly $\mathsf{conf}'_{t+1} \notin V(\mathcal{C}')$, we only know that $(\mathsf{conf}'_i, \mathsf{conf}'_{i+1}) \in E(\mathcal{C}')$ for every $1 \leq i \leq t-1$. Moreover, $(\mathsf{conf}'_{t+1}, \mathsf{conf}'_{t+2} = \mathsf{fin})$ is an r-transition so $(\mathsf{conf}'_{t+1}, \mathsf{fin}) \notin E(\mathcal{C}')$. To handle the vertex conf'_{t+1} and edges $(\mathsf{conf}'_t, \mathsf{conf}'_{t+1})$ and $(\mathsf{conf}'_{t+1}, \mathsf{conf}'_{t+2} = \mathsf{fin})$, we now give a procedure (Algorithm 2) to construct a new graph called a *generalized* $(k-1)$*-sparse configuration graph* of $\mathbf{SG}$ based on a $(k-1)$-sparse configuration graph. We first give an outline of the procedure.

We first initialize $\mathcal{C}'$ to be a $(k-1)$-sparse configuration graph of $\mathbf{SG}$ and $\mathcal{G}$ to be a corresponding $(V(G), 3k-2)$-permuter, returned by Algorithm 1. Then, we initialize a new graph $\mathcal{C}''$ as $\mathcal{C}'$, i.e. all the vertices and edges of $\mathcal{C}'$ are present in $\mathcal{C}''$. In addition to these vertices and edges, for every function $g \in \mathcal{G}$ and every pair of vertices (u, v) of G, such that $v \neq u$ and $u = f_{s+1}$ for some $1 \leq s \leq k-2$, we add at most a vertex and an edge as follows. If there exists a path $\widehat{P}$ of length $k-1$ between u and v such that $(\widehat{P}, \mathsf{fin})$ is a s-transition, considering $\widehat{P}$ as a configuration with u as its head and while traversing the path $\widehat{P}$ from u to v, the ordered set of images of the vertices of $\widehat{P}$ under g is $\{2k-1, 2k-1, \ldots, k\}$, then we add $\widehat{P}$ as a configuration with u as its head to $V(\mathcal{C}'')$ and the ordered pair $(\widehat{P}, \mathsf{fin})$ as a directed edge to $E(\mathcal{C}'')$. In the end, we return $\mathcal{C}''$.

Similarly to the notion of a $(k-1)$-sparse configuration graph, we now define the notion of a *generalized* $(k-1)$*-sparse configuration graph* as follows.

Definition 3.5. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game*. Then, any graph returned by Algorithm 2 is called a* generalized $(k-1)$-sparse configuration graph*.*

Algorithm 2: Construction of a generalized $(k-1)$-sparse configuration graph.

```
function generalizedSparseConfigurationGraph($\mathbf{SG}, k$);
let $\mathsf{fin} = (f_1, f_2, \dots, f_k)$;
initialize $(\mathcal{C}', \mathcal{G})$ = sparseConfigurationGraph($\mathbf{SG}, k$);
initialize $\mathcal{C}'' = \mathcal{C}'$;
for each $g \in \mathcal{G}$ do
    for every vertex $u \in V(G)$ such that $u = f_{s+1}$ for some $1 \leq s \leq k-2$ do
        for every vertex $v \in V(G)$ such that $v \neq u$ do
            if there is a (simple) path $\widehat{P} = (w_1 = u, w_2, \dots, w_k = v)$ of length $k-1$
              between $u$ and $v$ in $G$ such that, for every $1 \leq a \leq k$, $g(w_a) = 2k-a$
              and for every $1 \leq a \leq k-s$, $w_a = f_{a+s}$ then
                add $\widehat{P}$ as a configuration with $u$ as its head to $V(\mathcal{C}'')$;
                add ordered pair $(P, \mathsf{fin})$ as a directed edge in $E(\mathcal{C}'')$;
            end
        end
    end
end
return $\mathcal{C}''$.
```

We give the following lemma which relates paths in the 1-configuration graph and a generalized $(k-1)$-sparse configuration graph whose lengths are not multiples of $(k-1)$.

Lemma 3.9. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin} \rangle$ *be an instance of* SNAKE GAME *and* $t, r \in \mathbb{N}$ *such that* $r < k-1$. *Let* $\mathcal{C}''$ *be a generalized* $(k-1)$*-sparse configuration graph of* $\mathbf{SG}$. *Then, there exists a path of length* $t(k-1)+r$ *from* init *to* fin *in the* 1*-configuration graph if and only if there exists a path of length* $t+1$ *from* init *to* fin *in* $\mathcal{C}''$.

Proof. Let $\mathcal{C}$ be the $(k-1)$-configuration graph of $\mathbf{SG}$. Let $P = (\mathsf{conf}_1 = \mathsf{init}, \dots, \mathsf{conf}_k, \dots, \mathsf{conf}_{2k-1}, \dots, \mathsf{conf}_{t(k-1)+(r+1)} = \mathsf{fin})$ be a path from init to fin in the 1-configuration graph. From the above discussion, we can get a path $P^* = (\mathsf{conf}'_1 = \mathsf{init}, \mathsf{conf}'_2, \dots, \mathsf{conf}'_{t+1}, \mathsf{conf}'_{t+2} = \mathsf{fin})$ defined on the vertex set $V = V(P'') \cup \{\mathsf{fin}\}$ such that all the vertices except possibly conf'_{t+1} belong to $V(\mathcal{C}')$ and all the edges except possibly $(\mathsf{conf}'_t, \mathsf{conf}'_{t+1})$ and $(\mathsf{conf}'_{t+1}, \mathsf{conf}'_{t+2} = \mathsf{fin})$ belong to $E(\mathcal{C}')$.

Observe that $\mathcal{C}'$ is a subgraph of $\mathcal{C}''$. So, if we can find a configuration $\mathsf{conf}^* \in V(\mathcal{C}'')$ such that $(\mathsf{conf}'_t, \mathsf{conf}^*)$ is a $(k-1)$-transition, $(\mathsf{conf}^*, \mathsf{conf}'_{t+2})$ is an r-transition, and $(\mathsf{conf}'_t, \mathsf{conf}^*)$ and $(\mathsf{conf}^*, \mathsf{conf}'_{t+2})$ belong to $E(\mathcal{C}'')$, then we can replace conf'_{t+1} with conf^* in P^* to get a path from init to fin in $\mathcal{C}''$.

Consider the triplet $(\mathsf{conf}'_t, \mathsf{conf}'_{t+1}, \mathsf{conf}'_{t+2} = \mathsf{fin})$. Let $\mathsf{conf}'_{t+1} = (v'_1, v'_2, \dots, v'_k)$ and $\mathsf{fin} = (f_1, f_2, \dots, f_k)$. Let W be the triplet order of $(\mathsf{conf}'_t, \mathsf{conf}'_{t+1}, \mathsf{fin})$ and $\mathcal{G}$ be a $(V(G), 3k-2)$-permuter. As $(\mathsf{conf}'_t, \mathsf{conf}'_{t+1})$ is a $(k-1)$-transition and $(\mathsf{conf}'_{t+1}, \mathsf{fin})$ is an r-transition $(r < k-1)$, by Condition (i) of Lemma 3.3, there exists a function $h \in \mathcal{G}$ that maps W to an ordered set $\{i, i+1, \dots, k, k+1, \dots, 2k-1, \dots, j\}$ such that $1 \leq i \leq j \leq 3k-2$, $j-i+1 = |W|$ and for every $1 \leq a \leq k$, $h(v'_a) = 2k-a$. Also, as $(\mathsf{conf}'_{t+1}, \mathsf{fin})$ is an r-transition $(r < k-1)$, by Lemma 3.1, $v'_1 = f_{r+1}$. Let prop be the property of the path $\widehat{P}$

stated in line 8 of Algorithm 2. So, when Algorithm 2 considers $g = h, u = v'_1$ and $v = v'_k$, conf'_{t+1} satisfies the property prop for $s = r$. But there can be more than one path between v'_1 and v'_k that satisfy the property prop, so Algorithm 2 either adds conf'_{t+1} or any other path between v'_1 and v'_k satisfying the property prop. If the algorithm adds conf'_{t+1}, then $\mathsf{conf}^* = \mathsf{conf}'_{t+1}$ and we are done.

Otherwise, let $\widehat{\mathsf{conf}} = (w_1 = v'_1, w_2, \ldots, w_k = v'_k)$ be the configuration added by the algorithm. As $\widehat{\mathsf{conf}}$ satisfies the property prop for $s = r$ so for every $1 \leq a \leq k$, $g(w_a) = 2k-a$ and for every $1 \leq a \leq k-r$, $w_a = f_{a+r}$. Additionally, observe that $w_k = v'_k = v_1$, by Condition (iii) of Lemma 3.3, $(\mathsf{conf}, \widehat{\mathsf{conf}})$ is a $(k-1)$-transition and $(\widehat{\mathsf{conf}}, \mathsf{conf}'')$ is an r-transition. So, we can take $(\mathsf{conf}^* = \widehat{\mathsf{conf}})$ and we are done. $\square$

As $\mathcal{C}'$ is a subgraph of $\mathcal{C}''$, combining Observation 3.2 and Lemma 3.9 results in the following theorem.

Theorem 3.1. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game *and* $t, r \in \mathbb{N}$ *such that* $r \leq k-1$. *Then, there exists a path of length* $t(k-1)+r$ *from* init *to* fin *in the* 1*-configuration graph if and only if there exists a path of length* $t+1$ *from* init *to* fin *in any generalized* $(k-1)$*-sparse configuration graph.*

We now give the following lemma about the size of a generalized $(k-1)$-sparse configuration graph and the running time of Algorithm 2.

Lemma 3.10. *Let* $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game. *Then, Algorithm 2 runs in time* $k^{\mathcal{O}(k)}n^3\log^2 n$ *and returns a generalized* $(k-1)$*-sparse configuration graph of* $\mathbf{SG}$ *with* $k^{\mathcal{O}(k)}n^2\log n$ *vertices and* $k^{\mathcal{O}(k)}n^3\log^2 n$ *arcs.*

Proof. Let $|V(G)| = n$, $|E(G)| = m$, $\mathcal{C}'$ be a $(k-1)$-sparse configuration graph and $\mathcal{G}$ be a $(V(G), 3k-2)$-permuter constructed by Algorithm 1. From Corollary 5.1, we implicitly assume that $m = k^{\mathcal{O}(1)}n$. Let $\mathcal{C}''$ be a generalized $(k-1)$-sparse configuration graph. In addition to the vertices of $\mathcal{C}'$, Algorithm 2 adds at most a vertex and an edge to $\mathcal{C}''$ for each function $g \in \mathcal{G}$ and every pair of distinct vertices in G. From Lemma 3.5, we know that $|V(\mathcal{C}')| = k^{\mathcal{O}(k)}n^2\log n$ and $|E(\mathcal{C}')| = k^{\mathcal{O}(k)}n^3\log^2 n$ and Algorithm 1 runs in time $k^{\mathcal{O}(k)}n^3\log^2 n$. So, $|V(\mathcal{C}'')| = k^{\mathcal{O}(k)}n^2\log n$ and $|E(\mathcal{C}'')| = k^{\mathcal{O}(k)}n^3\log^2 n$. Also, again by Lemma 3.4, we can find a path $\widehat{P}$ between u and v satisfying the the condition given in line 8 of Algorithm 2 in time $\mathcal{O}(n+m)$. So, the running time of Algorithm 2 is $k^{\mathcal{O}(k)}n^3\log^2 n + (k^{\mathcal{O}(k)}\log n)\cdot n^2\cdot(n+m)) = k^{\mathcal{O}(k)}n^3\log^2 n$. $\square$

Thus, from Theorem 3.1 and Lemma 3.10, we derive the following theorem.

Theorem 3.2. *There exists an algorithm that, given an instance of* Snake Game $\mathbf{SG} = \langle G, k, \mathsf{init}, \mathsf{fin}\rangle$*, solves* $\mathbf{SG}$ *in time* $k^{\mathcal{O}(k)}|V(G)|^3\log^2|V(G)|$. *Moreover, it finds the shortest path from* init *to* fin *if one exists.*

In Table 1, we compare the running time of our algorithm to a naive algorithm that creates the 1-configuration graph and then looks for a path from the initial configuration to the final configuration. This naive algorithm does not use our method in order to sparsify the configuration graph. Observe that the size of an arbitrary configuration graph is $\mathcal{O}(\binom{n}{k}\cdot k!)$, where $n = |V(G)|$. Therefore, the running time of such an algorithm is $|V(G)|^{\mathcal{O}(k)}\cdot k^{\mathcal{O}(k)}$.

	Our algorithm	Naive algorithm
$k = 5, n = 30$	$8.4 \cdot 10^7$	$7.5 \cdot 10^{10}$
$k = 10, n = 100$	10^{16}	10^{30}
$k = 10, n = 500$	10^{18}	10^{37}
$k = 20, n = 10,000$	10^{38}	10^{107}
$k = 20, n = 100,000$	10^{41}	10^{127}

Table 1: Running time comparison.

Notice that a huge difference is obtained already for small instances. Moreover, in the real life, a snake of size 20 is already very big, therefore we expect that the differences in the running time are even more significant in practice.

4. Non-Existence of a Polynomial Kernel on Grid Graphs

In this section, we prove that the Snake Game problem is unlikely to admit a polynomial kernel even on grid graphs. Our proof is based on the exhibition of a cross-composition (see Definition 2.9) and on Proposition 2.1. We remind that our cross-composition is inspired by the proof of NP-hardness of Snake Game on grid graphs by Biasi and Ophelders (2018). As the source problem Π for the cross-composition, we select the Hamiltonian Cycle problem on grid graphs, and the target problem is Snake Game on grid graphs. In what follows, we first present the reduction that serves as the cross-composition (Section 4.1), and then prove its correctness (Section 4.2).

4.1 Construction of the Reduction Function

We refer to our reduction function as HamToSna. Its input consists of t instances $I_1, I_2, \ldots,$ I_t of the Hamiltonian Cycle problem on grid graphs where $t \in \mathbb{N}$, and its output is a single instance of the Snake Game problem on grid graphs. Each instance I_i is a grid graph G_i with n vertices, which we can assume without loss of generality to be connected else the instance is trivially a No-instance. We need the following definition for describing the reduction function.

For the construction of HamToSna, we would like to "separate" the t input grid graphs in the sense that we would like to identify specific "squares" in the output grid graph where the input grid graphs will be embedded. The notion of a *boundary square of a grid graph* (defined below) will serve this purpose. As intuition for its definition and the next observation, note that any connected grid graph G can be thought of as an induced subgraph of a $(|V(G)| \times |V(G)|)$-0- grid. We will denote the four corner vertices of this $(|V(G)| \times |V(G)|)$-0-grid by $(r_{\min}, c_{\min}), (r_{\min}, c_{\max}), (r_{\max}, c_{\min})$ and $(r_{\max}, c_{\max})$. Accordingly, we will say that G is "bounded" by $(r_{\min}, r_{\max}, c_{\min}, c_{\max})$. We would like this boundary to be unique, to which end we have Conditions 1, 2 and 3 as follows.

Definition 4.1 (A $(r_{\min}, r_{\max}, c_{\min}, c_{\max})$-Boundary Square of a Connected Grid Graph). *A* boundary square *of a connected grid graph G is a quadruple $(r_{\min}, r_{\max}, c_{\min},$ $c_{\max})$ where $r_{\min}, r_{\max}, c_{\min}, c_{\max} \in \mathbb{N}_0$, such that the following conditions are satisfied.*

1. $r_{\min} = \min\{r \in \mathbb{N}_0 \mid$ *there exists* $c \in \mathbb{N}_0$ *such that* $(r, c) \in V(G)\}$.

2. $c_{\min} = \min\{c \in \mathbb{N}_0 \mid$ *there 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. *Let* G *be a connected grid graph. Then, there exists a quadruple* $(r_{\min}, r_{\max}, c_{\min}, c_{\max}) \in \mathbb{N}_0^4$ *such that* $(r_{\min}, r_{\max}, c_{\min}, c_{\max})$ *is a boundary square of* G*, and it is unique.*

The HamToSna **Reduction Function: Preprocessing.** Let $G_1, G_2, \ldots, G_t$ be the t instances of Hamiltonian Cycle on grid graphs. From Observation 4.1, we know that there exists a boundary square of G_i for every $1 \le i \le t$. Let $(\widehat{r}^i_{\min}, \widehat{r}^i_{\max}, \widehat{c}^i_{\min}, \widehat{c}^i_{\max}) \in \mathbb{N}_0^4$ be the boundary square of G_i. Then, we modify G_i as follows. For every vertex $(a, b) \in V(G_i)$, we replace it by the vertex $(a - \widehat{r}^i_{\min} + (n+1), b - \widehat{c}^i_{\min} + i(n+1))$. Intuitively, this operation simply means that we "shift" the entire grid graph G_i in parallel to the axes. It is easy to see that the new boundary square $(r^i_{\min}, r^i_{\max}, c^i_{\min}, c^i_{\max}) \in \mathbb{N}_0^4$ of G_i satisfies the following conditions.

- $r^i_{\min} = n + 1$.
- $c^i_{\min} = i(n + 1)$.

In addition, because $(r^i_{\min}, r^i_{\max}, c^i_{\min}, c^i_{\max})$ is the boundary square of G_i, Condition 1 in Definition 4.1 implies that there exists a vertex $(a, b) \in V(G_i)$ such that $a = r^i_{\min}$. As there can be more than one such vertex, for every $1 \le i \le t$, we arbitrarily choose one such vertex in $V(G_i)$ and denote it by $(r^i_{\min}, c_i)$.

The HamToSna **Reduction Function: Construction** After executing the above preprocessing step, we define $\mathsf{HamToSna}(G_1, G_2, \ldots, G_t) = \langle G, n, \mathsf{init}, \mathsf{fin}\rangle$ as follows. (See Figure 3.

1. $V(G) = V^{\mathsf{InitFinConfs}} \cup V^{\mathsf{Connectors}} \cup V^{\mathsf{Instances}}$, where
   - $V^{\mathsf{InitFinConfs}} = \{(n-1, j) \mid 0 \le j \le n-1\} \cup \{(i, n-2) \mid 0 \le i \le n-1\}$,
   - $V^{\mathsf{Connectors}} = \{(n-1, j) \mid n \le j \le c_t\} \cup \{(n, c_i) \mid 1 \le i \le t\}$, and
   - $V^{\mathsf{Instances}} = \bigcup_{i=1}^{t} V(G_i)$.
2. $E(G) = \{((a, b), (c, d)) \mid (a, b), (c, d) \in V(G), |a - c| + |b - d| = 1\}$.
3. $\mathsf{init} = ((n-1, n-1), (n-1, n-2), (n-1, n-3), \ldots, (n-1, 0))$.
4. $\mathsf{fin} = ((0, n-2), (1, n-2), (2, n-2), \ldots, (n-1, n-2))$.

The following observation about the reduction function HamToSna follows directly from its definition.

Observation 4.2. *The reduction function* HamToSna *returns a valid instance of the* Snake Game *problem on grid graphs. Moreover, it can be computed in polynomial time.*

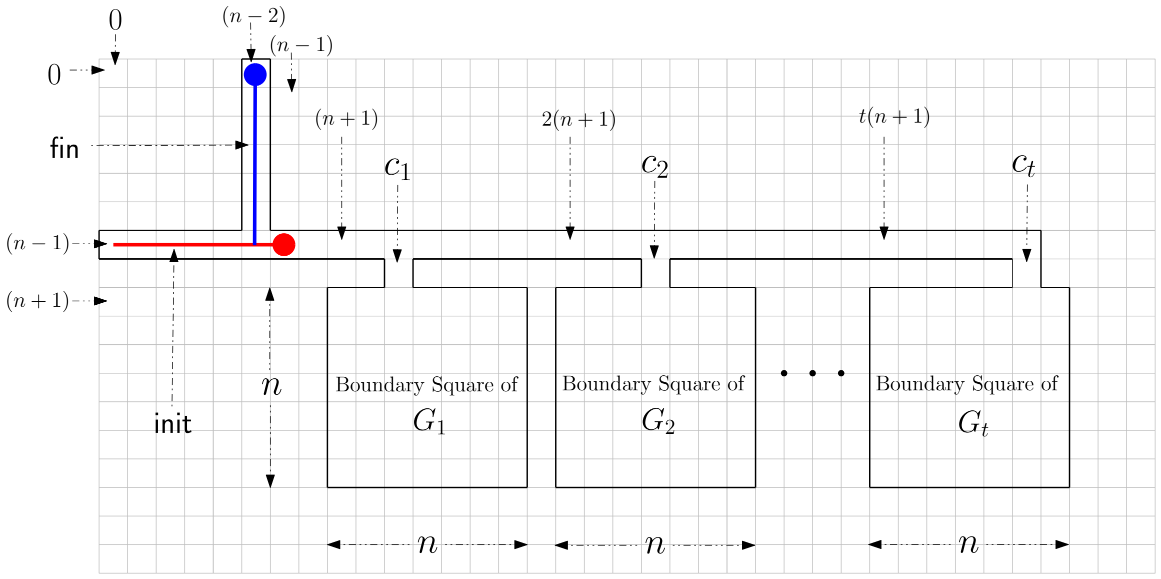


Figure 3: Graph constructed by the reduction function HamToSna.

4.2 The Correctness of Construction

To prove the correctness of our reduction, we need to show that the configuration init can reach the configuration fin if and only if there exists a Yes-instance among the t instances of Hamiltonian Cycle. Towards this, we begin by showing that in order to reach the configuration fin, the configuration init must first reach a configuration where the head of the snake is at one of the vertices of one of the t instances of Hamiltonian Cycle, that is, at $\bigcup_{i=1}^{t} V(G_i)$. Clearly, the snake must repeatedly make a move (i.e. a 1-transition) until it has not reached fin, and as long as it has also not reached a vertex in $\bigcup_{i=1}^{t} V(G_i)$, the choice of this move is very limited. Indeed, to see this, note that the graph we have constructed has no cycle that involves vertices that do not belong to $\bigcup_{i=1}^{t} V(G_i)$. Thus, starting at init and making one move at a time, the snake will never be able to "turn around" to reach fin unless it traverses a cycle in a graph G_i.

To formalize this, we define the notion of the *head's neighbor set*. Roughly speaking, this set consists of the vertices to which the snake (in a given configuration) can move its head, thereby describing all possible 1-transitions of the configuration.

Definition 4.2 (Head's Neighbor Set). *Let $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ be an instance of* Snake Game. *Let* conf $= (v_1, \ldots, v_k)$ *be a configuration. The* head's neighbor set *of* conf*, denote by* $N^h(\mathsf{conf})$*, is defined as* $\{u \in V(G) | \{u, v_1\} \in E(G), u \neq v_i$ *for every* $1 \leq i \leq k-1\}$.

The following observation follows directly from Definition 4.2.

Observation 4.3. *Let $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ be an instance of* Snake Game. *Let* conf $= (v_1, \ldots, v_k)$ *be a configuration. For every $u \in N^h(\mathsf{conf})$, let* conf^u *denotes the tuple* $(u, v_1, \ldots, v_{k-1})$. *Then, for every $u \in N^h(\mathsf{conf})$,* conf^u *is a configuration. Moreover,* $\bigcup_{u \in N^h(\mathsf{conf})}\{\mathsf{conf}^u\}$ *is the set of all the configurations* conf′ *such that* (conf, conf′) *is a* 1*-transition.*

Now, we further observe that if the initial and final configurations are not equal, the final configuration is reachable from the initial configuration if and only if the final configuration is reachable from at least one of the configurations where the head of the snake moves to one of the vertices in the head's neighbor set.

Observation 4.4. *Let $\langle G,k,\mathsf{init},\mathsf{fin}\rangle$ be an instance of* Snake Game *where* $\mathsf{init}\neq\mathsf{fin}$. *Then, $\langle G,k,\mathsf{init},\mathsf{fin}\rangle$ is a* Yes*-instance if and only if there exists a* Yes*-instance in $\{\langle G,k,\mathsf{init}^u,\mathsf{fin}\rangle \mid u\in N^h(\mathsf{init})\}$.*

Proof. ($\Rightarrow$) Suppose that $\langle G,k,\mathsf{init},\mathsf{fin}\rangle$ is a Yes-instance. Then, by Definitions 2.6 and 2.7, the pair $(\mathsf{init},\mathsf{fin})$ is an ℓ-transition for some $\ell\in\mathbb{N}$. Moreover, by Definition 2.5, there exists a tuple $(\mathsf{conf}_1=\mathsf{init},\mathsf{conf}_2,\ldots,\mathsf{conf}_\ell=\mathsf{fin})$ such that, for every $1\leq i\leq\ell$, the pair $(\mathsf{conf}_i,\mathsf{conf}_{i+1})$ is a 1-transition. As $\mathsf{init}\neq\mathsf{fin}$, we know that $\ell>1$. Moreover, as $(\mathsf{init},\mathsf{conf}_2)$ is a 1-transition, by Observation 4.3, there exists $u\in N^h(\mathsf{init})$ such that $\mathsf{init}^u=\mathsf{conf}_2$. So, we get that $(\mathsf{conf}_2=\mathsf{init}^u,\ldots,\mathsf{conf}_\ell=\mathsf{fin})$ is an $\ell-1$-transition. Thus, by Definitions 2.6 and 2.7, $\langle G,k,\mathsf{init}^u,\mathsf{fin}\rangle$ is a Yes-instance.

($\Leftarrow$) Suppose that $\langle G,k,\mathsf{init}^u,\mathsf{fin}\rangle$ is a Yes-instance for some $u\in N^h(\mathsf{init})$. Then, by Definitions 2.6 and 2.7, the pair $(\mathsf{init}^u,\mathsf{fin})$ is an ℓ-transition for some $\ell\in\mathbb{N}$. Moreover, by Definition 2.5, there exists a tuple $(\mathsf{conf}_1=\mathsf{init}^u,\mathsf{conf}_2,\ldots,\mathsf{conf}_\ell=\mathsf{fin})$ such that, for every $1\leq i\leq\ell$, the pair $(\mathsf{conf}_i,\mathsf{conf}_{i+1})$ is a 1-transition. As $(\mathsf{init},\mathsf{init}^u)$ is a 1-transition, we get that $(\mathsf{init},\mathsf{conf}_1,\mathsf{conf}_2,\ldots,\mathsf{conf}_\ell)$ is an $\ell+1$-transition. Thus, by Definitions 2.6 and 2.7, $\langle G,k,\mathsf{init},\mathsf{fin}\rangle$ is a Yes-instance. $\square$

We proceed as follows. For every $1\leq i\leq t$, let $\mathsf{init_i}$ denote the configuration whose head is $(r^i_{\min},c_i)$ and the rest of its vertices does not belong to $V(G_i)$. (See Figure 4). Then, we show that $\langle G,k,\mathsf{init},\mathsf{fin}\rangle$ is a Yes-instance if and only if at least one of the instances $\langle G,k,\mathsf{init_i},\mathsf{fin}\rangle$ is a Yes-instance.

Lemma 4.1. *Let $G_1,G_2,\ldots,G_t$ be t instances of* Hamiltonian Cycle *on grid graphs. Let $\mathsf{HamToSna}(G_1,G_2,\ldots,G_t)=\langle G,n,\mathsf{init},\mathsf{fin}\rangle$, and $\mathsf{init}_i:=((n+1,c_i),(n,c_i),(n-1,c_i),(n-1,c_i-1)\ldots(n-1,c_i-(n-3)))$ for every $1\leq i\leq t$ (see Figure 4). Then, $\langle G,n,\mathsf{init},\mathsf{fin}\rangle$ is a* Yes*-instance if and only if there exists a* Yes*-instance in $\{\langle G,n,\mathsf{init}_i,\mathsf{fin}\rangle \mid 1\leq i\leq t\}$.*

Proof. For every $1\leq i\leq t$, denote $\mathsf{init}'_i=((n-1,c_i),(n-1,c_i-1),\ldots,(n-1,c_i-(n-1)))$. Refer to Figure 4. Recall that $\mathsf{init}=((n-1,n-1),(n-1,n-2),(n-1,n-3),\ldots,(n-1,0))$, and notice that $N^h(\mathsf{init})=\{(n-1,n)\}$. So, by Observation 4.4, we know that $\langle G,n,\mathsf{init},\mathsf{fin}\rangle$ is a Yes-instance if and only if $\langle G,n,((n-1,n),(n-1,n-1),(n-1,n-2),\ldots,(n-1,1)),\mathsf{fin}\rangle$ is a Yes-instance. By iteratively repeating this step c_1-n additional times, we derive that $\langle G,n,\mathsf{init},\mathsf{fin}\rangle$ is a Yes-instance if and only if $\langle G,n,\mathsf{init}'_1,\mathsf{fin}\rangle$ is a Yes-instance.

Now, for any $1\leq i\leq t-1$, note that $N^h(\mathsf{init}'_i)=\{(n,c_i),(n-1,c_i+1)\}$ and $N^h(((n,c_i),(n-1,c_i),\ldots,(n-1,c_i-(n-2))))=\{(n+1,c_i)\}$. So, by Observation 4.4, we know that $\langle G,n,\mathsf{init}'_i,\mathsf{fin}\rangle$ is a Yes-instance if and only if at least one of the instances $\langle G,n,\mathsf{init}_i,\mathsf{fin}\rangle$ and $\langle G,n,((n-1,c_i+1),(n-1,c_i),\ldots,(n-1,c_i-(n-2))),\mathsf{fin}\rangle\}$ is a Yes-instance. Furthermore, repeating the argument of the first paragraph for $((n-1,c_i+1),(n-1,c_i),\ldots,(n-1,c_i-(n-2)))$, we obtain that $\langle G,n,((n-1,c_i+1),(n-1,c_i),\ldots,(n-1,c_i-(n-2))),\mathsf{fin}\rangle$ is a Yes-instance if and only if $\langle G,n,\mathsf{init}'_{i+1},\mathsf{fin}\rangle$ is a Yes-instance.

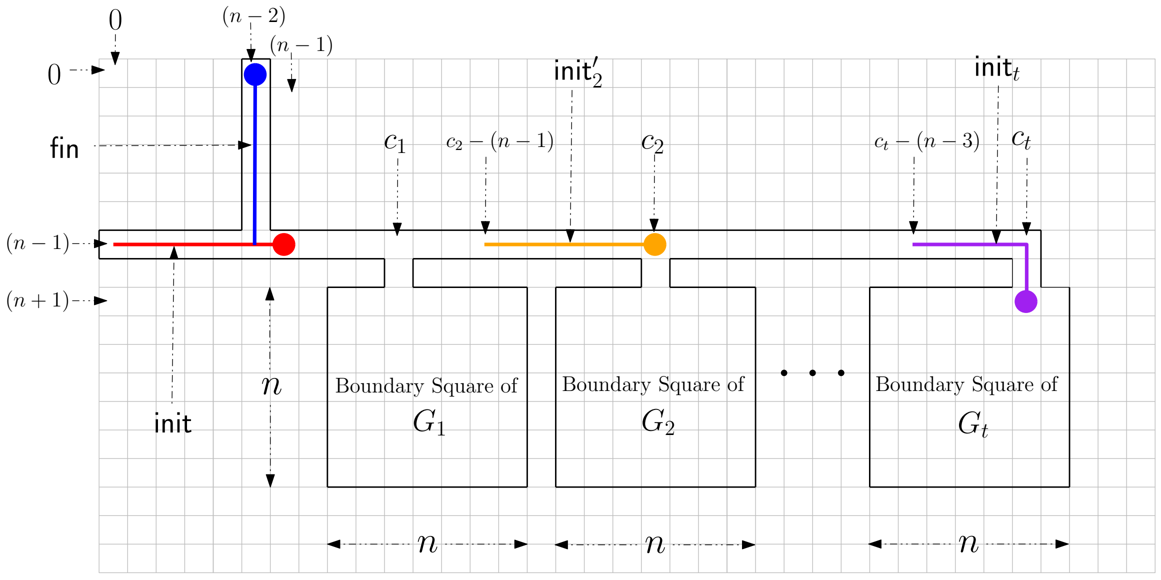


Figure 4: The configurations init, fin, init_i and init'_i for $1 \leq i \leq t$ (specifically, init'_2 and init_t are shown).

Putting the two claims above together, we derive that $\langle G, n, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance if and only if there exists a Yes-instance in $\{\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle \mid 1 \leq i \leq t-1\} \cup \{\langle G, n, \mathsf{init}'_t, \mathsf{fin}\rangle\}$. As $N^h(\mathsf{init}'_t) = \{(n, c_t)\}$ and $N^h(((n, c_t), (n-1, c_t), \dots, (n-1, c_t-(n-2)))) = \{(n+1, c_t)\}$, by Observation 4.4, we know that $\langle G, n, \mathsf{init}'_t, \mathsf{fin}\rangle$ is a Yes-instance if and only if $\langle G, n, \mathsf{init}_t, \mathsf{fin}\rangle$ is a Yes-instance. In turn, this completes the proof. □

Up until now, we have shown that the snake must reach one of the graphs corresponding to the t instances of Hamiltonian Cycle in order to reach fin. Next, we show that once the snake reaches one of these instances, it can reach fin if and only if that instance is a Yes-instance of Hamiltonian Cycle. For the sake of clarity, we split the proof into two lemmas as follows.

Lemma 4.2. *If G_i is a Yes-instance of the* Hamiltonian Cycle *problem for some $1 \leq i \leq t$, then $\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle$ is a Yes-instance of* Snake Game.

Proof. Assume that G_i is a Yes-instance of the Hamiltonian Cycle problem for some $1 \leq i \leq t$. Let $V(G_i) = \{a_1, a_2, \dots, a_n\}$ where $a_1 = (n+1, c_i)$. So, there exists a simple cycle $C = a_1 - \dots - a_n - a_1$ in G_i. Note that $V(\mathsf{init}_i) \cap V(G_i) = \{a_1\}$, $\{a_1, a_2\} \in E(G_i)$ and $a_1 \neq a_2$. Therefore, $(\mathsf{init}_i = (a_1, (n, c_i), (n-1, c_i), (n-1, c_i-1), \dots (n-1, c_i-(n-3))), (a_2, a_1, (n, c_i), (n-1, c_i), (n-1, c_i-1), \dots (n-1, c_i-(n-4))))$ is a 1-transition.

By iteratively repeating the above transformation $n-1$ additional times, we get that the pair $(\mathsf{init}_i = (a_1, (n, c_i), \dots (n-1, c_i-(n-3))), (a_1, a_n, \dots a_2))$ is an n-transition. So, by Definition 2.6, we know that init_i can reach $(a_1, a_n, \dots a_2)$. Moreover, by Observation 4.4 (as in the proof of Lemma 4.1), it is easily seen that $(a_1, a_n, \dots a_2)$ can reach fin. Combining these two statements together, we get that init_i can reach fin. By Definition 2.7, we thus conclude that $\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle$ is a Yes-instance of the Snake Game problem. □

Now, we show that the opposite direction of Lemma 4.2 is also true. Intuitively, if the snake is located at init_i, then it must traverse a Hamiltonian cycle in G_i in order to "exit" G_i and reach fin—in particular, the Hamiltonicity condition has to be satisfied otherwise the snake will intersect itself while trying to "exit" G_i.

Lemma 4.3. *If $\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle$ is a Yes-instance of* Snake Game *for some $1 \leq i \leq t$, then G_i is a Yes-instance of the* Hamiltonian Cycle *problem.*

Proof. Assume that $\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle$ is a Yes-instance of the Snake Game problem for some $1 \leq i \leq t$. By Definitions 2.6 and 2.7, the pair $(\mathsf{init}_i, \mathsf{fin})$ is an ℓ-transition for some $\ell \in \mathbb{N}$. Moreover, by Definition 2.5, there exists a tuple $T = (\mathsf{conf}_1 = \mathsf{init}_i, \mathsf{conf}_2, \ldots, \mathsf{conf}_{\ell+1} = \mathsf{fin})$ such that, for every $1 \leq i \leq \ell$, the pair $(\mathsf{conf}_i, \mathsf{conf}_{i+1})$ is a 1-transition. For every $1 \leq j \leq \ell + 1$, let $\mathsf{conf}_j = (v^j_1, \ldots, v^j_n)$.

First, we show that $\ell + 1 > n$. By way of contradiction, suppose that $\ell + 1 \leq n$. Note that $|V(\mathsf{init}_i)| = |V(\mathsf{fin})| = n$. As $(\mathsf{init}_i, \mathsf{fin})$ is an ℓ-transition for some $\ell \leq n - 1$, by Lemma 3.1, $|V(\mathsf{init}_i) \cap V(\mathsf{fin})| \geq n - \ell \geq 1$. By construction, $|V(\mathsf{init}_i) \cap V(\mathsf{fin})| \leq 1$, so we get that $|V(\mathsf{init}_i) \cap V(\mathsf{fin})| = 1$, which in turn gives us that $\ell = n - 1$. So, by Corollary 3.1, head of the configuration init_i should be same as the tail of the configuration fin, which is a contradiction. Thus, we get that $\ell + 1 > n$, hence it is well defined to consider the first $n + 1$ configurations of the tuple T.

Let $a = (n+1, c_i)$. Observe that $v^1_1 = a \in V(G_i)$. Furthermore, observe that $\mathsf{HamToSna}$ constructs G in a way that a is the only vertex in $V(G_i)$ that is connected to vertices not in $V(G_i)$. As $v^1_2 = (n, c_i)$, by the definition of 1-transition, $v^2_1 \neq (n, c_i)$, $v^2_1 \neq v^1_1$ and $\{v^1_1, v^2_1\} \in E(G)$. Moreover, (n, c_i) is the only neighbor of v^1_1 which is not in $V(G_i)$, so $v^2_1 \in V(G_i)$ and $(v^1_1, v^2_1) \in E(G_i)$. By repeating the same argument for the first n configurations of the tuple T, we get that for every $1 \leq k < j \leq n$, $v^j_1 \in V(G_i)$, $v^k_1 \neq v^j_1$ and for every $1 \leq j < n$, $\{v^j_1, v^{j+1}_1\} \in E(G_i)$. In turn, we get that for every $1 \leq k < j \leq n$, $v^n_j \in V(G_i)$, $v^n_k \neq v^n_j$ and for every $1 \leq j < n$, $(v^n_j, v^n_{j+1}) \in E(G_i)$. Therefore, $v^n_1 - \cdots - v^n_n$ is a Hamiltonian *path* in G_i.

Towards the proof that v^n_1 is adjacent to v^n_n in G_i, notice that $v^n_n = a \neq v^n_1$. Moreover, from tuple T, we know that fin is reachable from conf_n. As a is the only vertex in $V(G_i)$ that is connected to vertices not in $V(G_i)$ and by the definition of 1-transition, we get that $v^{n+1}_1 = a$. This implies that $\{a, v^1_n\} \in E(G_i)$. Therefore, we get that $a = v^n_1 - \ldots, v^n_n - a$ is a Hamiltonian cycle in G_i. We thus conclude that G_i is a Yes-instance of the Hamiltonian Cycle problem. ☐

We now combine Lemmas 4.2 and 4.3 to conclude the correctness of the construction.

Lemma 4.4. *Let $G_1, G_2, \ldots, G_t$ be t instances of* Hamiltonian Cycle *on grid graphs. Then, at least one of the instances $G_1, G_2, \ldots, G_t$ is a Yes-instance of* Hamiltonian Cycle *if and only if $\mathsf{HamToSna}(G_1, G_2, \ldots, G_t)$ is a Yes-instance of* Snake Game.

Proof. ($\Rightarrow$) Assume that at least one of $G_1, G_2, \ldots, G_t$ is a Yes-instance of Hamiltonian Cycle. Let i be such that G_i is a Yes-instance of Hamiltonian Cycle. From Lemma 4.2, we get that $\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle$ is a Yes-instance of Snake Game. Therefore, at least one instance in $\{\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle \mid 1 \leq i \leq t\}$ is a Yes-instance. So, from Lemma 4.1, we get that $\mathsf{HamToSna}$ $(G_1, G_2, \ldots, G_t) = \langle G, n, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance of Snake Game.

($\Leftarrow$) Assume that $\mathsf{HamToSna}(G_1, G_2, \ldots, G_t) = \langle G, n, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance of Snake Game. Then, from Lemma 4.1, we get that at least one instance in $\{\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle \mid 1 \leq i \leq t\}$ is a Yes-instance. Let i be such that $\langle G, n, \mathsf{init}_i, \mathsf{fin}\rangle$ is a Yes-instance. In turn, from Lemma 4.3, we get that G_i is a Yes-instance of Hamiltonian Cycle. So, at least one of $G_1, G_2, \ldots, G_t$ is a Yes-instance of Hamiltonian Cycle. □

4.3 Conclusion of the Proof

Up until now, we have proved the correctness of our reduction. Now, we are ready to prove the result of the (unlikely) existence of a polynomial kernel (or even compression) for Snake Game on grid graphs.

Lemma 4.5. *The* Hamiltonian Cycle *problem on grid graphs cross-composes into the* Snake Game *problem on grid graphs.*

Proof. By Observation 4.2, HamToSna is computable in polynomial-time. By Lemma 4.4, given any t instances of the Hamiltonian Cycle problem on grid graphs, HamToSna evaluates to a Yes-instance of the Snake Game problem on grid graphs if and only if at least one of the t instances of Hamiltonian Cycle is a Yes-instance. Thus, by Definition 2.9, Hamiltonian Cycle on grid graphs cross-composes into Snake Game on grid graphs. □

Thus, from Lemma 4.5 and Proposition 2.1, we get the following theorem.

Theorem 4.1. *The* Snake Game *problem on grid graphs does not admit a polynomial compression unless NP⊆coNP/poly.*

Proof. By Lemma 4.5, Hamiltonian Cycle on grid graphs cross-composes into Snake Game on grid graphs. As Hamiltonian Cycle on grid graphs is an NP-hard problem (Papadimitriou & Vazirani, 1984), Proposition 2.1 implies that Snake Game on grid graphs does not admit a polynomial compression unless NP⊆coNP/poly. □

5. Treewidth Reduction on General Graphs

A treewidth reduction of a problem Π is an algorithm that, given an instance I of Π, works in time $|I|^{\mathcal{O}(1)}$ and outputs an equivalent instance I' where the treewidth of the output graph is bounded by some computable function of some measure of the instance. In this section, we give a treewidth reduction of Snake Game, where the treewidth of the output graph is bounded by a polynomial in the size of the snake. This result may be interesting on its own due to the next observation. In practice, the graph may or may not look like a tree (e.g., when many obstacles are present and create cycles, or when elevated paths create alternatives), then treewidth reduction can reduce and simplify the instance substantially. From this reduction, we get that the "hard" instances are the ones with a structure similar to a tree.

A natural question that arises is whether Snake Game is FPT with respect to the graph treewidth. We have not resolved this, and hence it is left for further research. Nonetheless, due to our result of the unlikeliness of the Snake Game to admit a polynomial kernel with respect to the size of the snake in Section 4, combined with the result of this section, we can conclude that it is unlikely that Snake Game admits a polynomial kernel with respect to the graph treewidth. On the other hand, if Snake Game is FPT with respect to the graph treewidth, then the algorithm we present in this section can serve as a prepossessing routine in order to get an FPT algorithm with respect to $\min(k, t)$ where t is the treewidth of the graph. Moreover, as the number of edges of a graph with treewidth t is bounded by $t \cdot n$, the treewidth reduction we present can also serve as a prepossessing routine for reducing the number of edges. As it is shown in Section 4 that Snake Game is unlikely to admit a polynomial kernel, the treewidth reduction is a good alternative for a useful prepossessing. In addition, as we state in Section 1, we believe that the same treewidth reduction can be implemented in various other movement problems, and we have evidence that it can be implemented in the Push-pull block problem (defined in Section 1).

Our proof is based on a theorem that, for a given undirected graph G and $k \in \mathbb{N}$, asserts the following statement. We can, in time $\mathcal{O}(|V(G)| + |E(G)|)$, either find a specific "pattern" in the graph called a wall, or determine that G has treewidth $k^{\mathcal{O}(1)}$. Given an instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of Snake Game, we show how we use that pattern in G in order to decrease the number of vertices in G, yet retain an equivalent instance of Snake Game. Specifically, we utilize rerouting arguments to find a so called *irrelevant edge* (with respect to contraction), inspired by the classic work of Robertson and Seymour (1995). Because either way we obtain an exploitable structure (a wall or small treewidth), this method is known as "win/win approach" (Cygan et al., 2015). By applying this method repeatedly, we derive in polynomial time an equivalent instance of Snake Game, $\langle G', k, \mathsf{init}, \mathsf{fin}\rangle$, such that G' has treewidth $k^{\mathcal{O}(1)}$.

First, we present the formal definition of the specific pattern we are looking for (see Figure 5).

Definition 5.1 (Elementary r-Wall). *Let $r \in \mathbb{N}$. Let G_r be the $r \times 2r$-grid, i.e. $V(G_r) = \{(i, j) \mid i \in \{1, \dots, r\}, j \in \{1, \dots, 2r\}\}$ and $E(G_r) = \{\{(i, j), (i', j')\} \mid |i - i'| + |j - j'| = 1\}$. The* elementary r-wall *is the graph obtained from G_r by deleting all edges $\{(2i - 1, 2j - 1), (2i, 2j - 1)\}$ for $i \in \{1, 2, \dots, \lfloor r/2 \rfloor\}$ and $j \in \{1, 2, \dots, r\}$ and all edges $\{(2i, 2j), (2i + 1, 2j)\}$ for $i \in \{1, 2, \dots, \lfloor (r - 1)/2 \rfloor\}$ and $j \in \{1, 2, \dots, r\}$, and then deleting the two resulting vertices of degree* 1.

Definition 5.2 (r-Wall). *Let $r \in \mathbb{N}$. A graph G is an* r-wall *if it can be obtained from an elementary r-wall by subdividing edges.*

We would like to exploit the notion of an r-wall to present our reduction. To this end, we say that a graph G *contains an r-wall* if there exists a subgraph H of G such that H is an r-wall. Now, we aim to show that, given an instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of Snake Game where G contains a $3\sqrt{k}$-wall, we can efficiently decrease the number of vertices in G and still retain an equivalent instance of Snake Game. In particular, we consider a $3\sqrt{k}$-wall because of its following special property. For any two vertices s and t in a $3\sqrt{k}$-wall H, there exists a simple path (or simple cycle if $s = t$) in H that starts with s and ends with t whose size is at least k. Moreover, as it follows from the next observation, this property is preserved even if we contract any edge in H.

Observation 5.1. *Let H be a $3\sqrt{k}$-wall. Let $e = \{u, v\} \in E(H)$. Then, for every pair of vertices $s, t \in V(H/e)$, there exists a (simple) path P (or cycle if $s = t$) in H/e between s and t such that the size of P is at least k.*

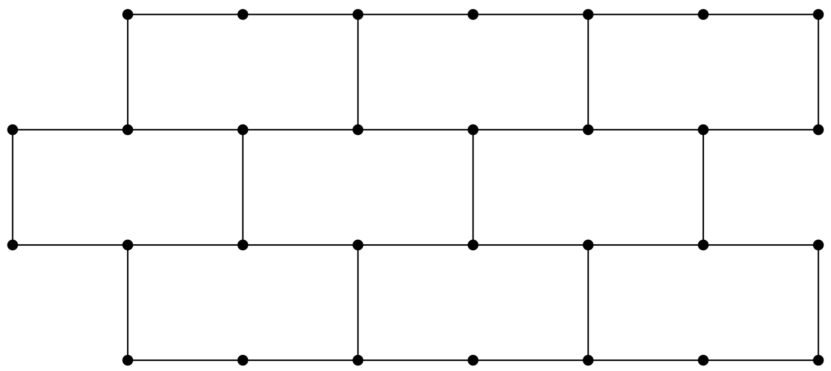


Figure 5: An elementary 4-wall.

Next, we consider the case where we have two configurations conf_1 and conf_2, and a (simple) path of size at least k between the head of conf_1 and the tail of conf_2. In addition, we suppose that the path does not "intersect" conf_1 and conf_2 besides at these two vertices. We prove (in Lemma 5.1) that, in this case, conf_2 must be reachable from conf_1. Intuitively, the snake can move from conf_1 through the path and thereby it can reach conf_2. To this end, we start with a simple observation. In this observation, we consider the case where we have a configuration conf_1 and a (simple) path P of size at least k that starts at the head of conf_1, ends at a vertex t, and the remainder of P and conf_1 do not intersect. Informally, our observation states that in this case, by making the snake traverse P, we get that conf_1 can reach some configuration conf_2 where t is the head, and the rest of the vertices of the snake is located on P. Moreover, if P is of size exactly k, then it corresponds to a configuration that is reachable from conf_1. Notice that in this case, the path is reachable from conf_1 in a "reversed" order of vertices, as the first vertex of the path "becomes" the tail of the snake when the snake traverses the path.

Observation 5.2. *Let $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ be an instance of* Snake Game. *Let $\mathsf{conf}_1 = (u_1, \dots, u_k)$ be a configuration. Let $P = (v_1, \dots, v_\ell)$ be a simple path or cycle in G of size at least k, such that $u_1 = v_1$, and for all $2 \leq i \leq k$ and $2 \leq j \leq \ell$, $u_i \neq v_j$. Then, there exists a configuration $\mathsf{conf}_2 = (u'_1, \dots, u'_k)$ such that $u'_1 = v_\ell$, for all $1 \leq i \leq k$, $u'_i \in V(P)$, and conf_1 can reach conf_2.*

In addition, if P is a path of size k, then $\mathsf{conf} = (v_\ell, \dots, v_1)$ is a configuration reachable from conf_1.

Now, we use Observation 5.2 to prove the following lemma.

Lemma 5.1. *Let $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ be an instance of* Snake Game. *Let $\mathsf{conf}_1 = (u_1, \dots, u_k)$ and $\mathsf{conf}_2 = (a_1, \dots, a_k)$ be two configurations. Let $P = (v_1, \dots, v_\ell)$ be a simple path or cycle in G of size at least k such that the following conditions hold (see Figure 6).*

- *$v_1 = u_1$ and $v_\ell = a_k$.*
- *For every $2 \leq i \leq k$ and $2 \leq j \leq \ell$, $u_i \neq v_j$.*

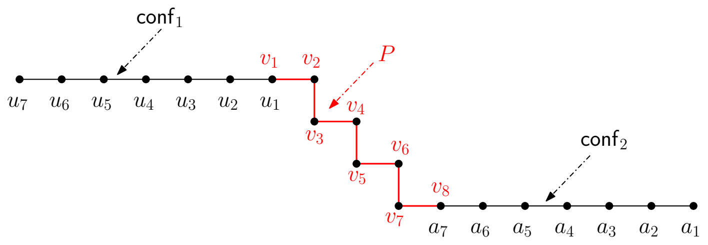


Figure 6: An example of the pre-conditions of Lemma 5.1 for $k = 7$ and $\ell = 8$.

- *For every* $1 \leq i \leq k-1$ *and* $1 \leq j \leq \ell - 1$*,* $a_i \neq v_j$*.*

Then, conf_1 *can reach* conf_2*.*

Proof. Notice that $\mathsf{conf}_1 = (u_1, \ldots, u_k)$ and $P = (v_1, \ldots, v_\ell)$ satisfy the conditions of Observation 5.2. Therefore, by this observation, there exists a configuration $\mathsf{conf}'_2 = (u'_1, \ldots, u'_k)$ such that the following conditions are satisfied:

- $u'_1 = v_\ell$.
- For every $1 \leq i \leq k$, $u'_i \in V(P)$.
- conf_1 can reach conf'_2.

Now, note that $\mathsf{conf}_2 = (a_1, \ldots, a_k)$ corresponds to a simple path in G, which can be denoted by $P' = (a'_1 = a_k, \ldots, a'_k = a_1)$. In particular, the following conditions are satisfied:

- $a'_1 = u'_1$.
- For every $2 \leq i \leq k$ and $2 \leq j \leq k$, $u'_i \neq a'_j$.
- P' is a simple path in G of size k.

Therefore, by Observation 5.2, conf'_2 can reach conf_2. In turn, we conclude that conf_1 can reach conf_2. □

Now, given an instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of Snake Game, we aim to show that if G contains a $3\sqrt{k}$-wall H that intersects neither init nor fin, then for any $e \in E(H)$, $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is an instance equivalent to $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$. The next lemma states the "harder" direction of this claim. Intuitively, if $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance, then there exists an ℓ-transition from init to fin for some $\ell \in \mathbb{N}$. So, we can consider the first and last configurations that intersect $V(H)$ (if such configurations exist), say conf_1 and conf_2. Because conf_1 is the first configuration that intersects $V(H)$, we show (in the proof of the next lemma) that the head of the snake in conf_1 is a vertex from $V(H)$, and the rest of the vertices of the snake do not intersect $V(H)$. Similarly, we show that the tail of the snake in conf_2 is a vertex from $V(H)$, and the rest of the vertices of the snake do not intersect $V(H)$. By Observation 5.1, we get that there is a path in H of size at least k between the head of conf_1 and the tail of conf_2. By traversing this path from conf_1, we show that snake can reach conf_2. Therefore, we get that (in $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$) init can reach conf_1, conf_1 can reach conf_2, and conf_2 can reach fin. So, in turn we conclude that $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance.

Lemma 5.2. *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 $e = \{u, v\} \in E(H)$. If $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ is a* Yes*-instance, then $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is a* Yes*-instance.*

Proof. Suppose that $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance. Then, there exists an ℓ-transition $T = (\mathsf{conf}_1, \ldots, \mathsf{conf}_{\ell+1})$ such that $\mathsf{conf}_1 = \mathsf{init}$ and $\mathsf{conf}_{\ell+1} = \mathsf{fin}$.

First, assume that for every $1 \le i \le \ell+1$, conf_i does not contain any vertices from $V(H)$. Then, $(\mathsf{conf}_1, \ldots, \mathsf{conf}_{\ell+1})$ is also an ℓ-transition in $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$, so it is a Yes-instance. Now, assume that there exists $1 \le i \le \ell + 1$ such that conf_i contains at least one vertex from $V(H)$. Let i and j be such that conf_i is the first configuration and conf_j is the last configuration in T that contains a vertex from $V(H)$. Notice that $1 < i \le j < \ell+1$ because we assumed that H contains vertices from neither init nor fin. Denote $\mathsf{conf}_i = (a_1, \ldots, a_k)$, and observe that $a_1 \in V(H)$ and for every $2 \le m \le k$, $a_m \notin V(H)$ (otherwise conf_i would not be the first configuration to contain a vertex from H, a contradiction). Similarly, denote $\mathsf{conf}_j = (b_1, \ldots, b_k)$, and observe that $b_k \in V(H)$ and for every $1 \le m \le k-1$, $b_m \notin V(H)$.

Recall that $v_e \in V(G/e)$ denotes the new vertex that was created by the contraction of e. Now, let conf'_i be a configuration and a'_1 a vertex defined as follows. If $a_1 = u$ or $a_1 = v$ then $\mathsf{conf}'_i = (v_e, a_2, \ldots, a_k)$ and $a'_1 = v_e$, else $\mathsf{conf}'_i = \mathsf{conf}_i$ and $a'_1 = a_1$. Similarly, if $b_k = u$ or $b_k = v$ then let $\mathsf{conf}'_j = (b_1, \ldots, b_{k-1}, v_e)$ and $b'_k = v_e$, otherwise $\mathsf{conf}'_j = \mathsf{conf}_j$ and $b'_k = b_k$. Now, notice that $(\mathsf{conf}_1, \ldots, \mathsf{conf}_{i-1}, \mathsf{conf}'_i)$ is an $(i-1)$-transition and $(\mathsf{conf}'_j, \mathsf{conf}_{j+1}, \ldots, \mathsf{conf}_{\ell+1})$ is an $(\ell + 1 - j)$-transition in $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$. Therefore, we know that both conf_1 can reach conf'_i and conf'_j can reach $\mathsf{conf}_{\ell+1}$ in $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$.

By Observation 5.1, we know that there exists a simple path $P = (u_1, \ldots, u_\ell)$ (or cycle if $a'_1 = b'_k$) of size at least k such that $u_1 = a'_1$, $u_\ell = b'_k$, and for every $1 \le i \le \ell$, $u_i \in V(H/e)$. Therefore, by Lemma 5.1 we get that conf'_i can reach conf'_j in $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$. Overall, as $\mathsf{conf}_1 = \mathsf{init}$ can reach conf'_i, conf'_i can reach conf'_j and conf'_j can reach $\mathsf{conf}_{\ell+1} = \mathsf{fin}$, we conclude that init can reach fin in $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$. Therefore $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ Yes-instance. □

Now, we consider the opposite direction of Lemma 5.2. Roughly speaking, the idea of the proof is as follows. If $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance, then we have an ℓ-transition $(\mathsf{conf}_1, \ldots, \mathsf{conf}_{\ell+1})$ such that $\mathsf{conf}_1 = \mathsf{init}$ and $\mathsf{conf}_{\ell+1} = \mathsf{fin}$. We aim to "fix" $(\mathsf{conf}_1, \ldots,$ $\mathsf{conf}_{\ell+1})$ to "fit" $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ by substituting all appearances of v_e. Intuitively, if the snake traverse a path that passes through v_e in G/e, it can visit at least one among u and v instead in G.

Lemma 5.3. *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 $e = \{u, v\} \in E(H)$. If $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is a* Yes*-instance, then $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ is a* Yes*-instance.*

Proof. Suppose that $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance. Then, there exists an ℓ-transition $(\mathsf{conf}_1\ , \ldots, \mathsf{conf}_{\ell+1})$ such that $\mathsf{conf}_1 = \mathsf{init}$ and $\mathsf{conf}_{\ell+1} = \mathsf{fin}$. If for every $1 \le i \le \ell + 1$, $v_e \notin \mathsf{conf}_i$, then $(\mathsf{conf}_1, \ldots, \mathsf{conf}_{\ell+1})$ is also an ℓ-transition in $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$, so it is a Yes-instance.

Now, assume that there exists $1 \le i \le \ell + 1$ such that v_e appears in conf_i. Because H does not have any vertices from init or fin, we get that v_e does not appear in init or fin. For each $1 \le i \le \ell + 1$, denote $\mathsf{conf}_i = (v^i_1, \ldots, v^i_k)$. Then, for each $1 \le i \le \ell + 1$ and $1 \le j \le k$

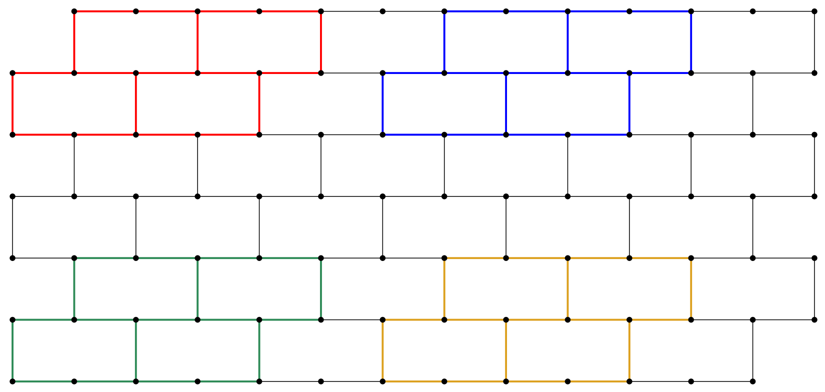


Figure 7: Four distinct elementary 3-walls in an elementary 7-wall.

such that $v^i_j = v_e$, it follows that $v^{i-j+1}_1 = v_e$. So, to cover all configurations in the given ℓ-transition $(\mathsf{conf}_1, \ldots, \mathsf{conf}_{\ell+1})$ that contain v_e, it is enough to consider only configurations $\mathsf{conf}_i = (v^i_1, \ldots, v^i_k)$ such that $v^i_1 = v_e$ and the $k-1$ configurations that come after them.

Now, let $1 \leq i \leq \ell + 1$ such that $\mathsf{conf}_i = (v^i_1, \ldots, v^i_k)$ and $v^i_1 = v_e$. By definition, $v^{i-1}_1 \in N_{G/e}(v_e)$. Recall that $e = \{u, v\}$ and G/e is the graph obtained from G by contracting $\{u, v\}$. Therefore, $N_{G/e}(v_e) = N_G(u) \cup N_G(v)$. Assume without loss of generality that $v^{i-1}_1 \in N_G(u)$. Now, if $v^{i+1}_1 \in N_G(u)$, then for every $0 \leq s \leq k-1$, we can replace v^{i+s}_{1+s} by u and thereby define valid configurations in $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$. Notice that we derive a valid transition in $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$. Now, assume $v^{i+1}_1 \in N_G(v)$. Then we can replace $\mathsf{conf}_i, \ldots, \mathsf{conf}_{i+k-1}$ by $\mathsf{conf}'_i, \ldots, \mathsf{conf}'_{i+k}$, where $\mathsf{conf}'_i = (u, v^i_2, \ldots, v^i_k)$, $\mathsf{conf}'_{i+1} = (v, u, v^i_2, \ldots, v)$, $\mathsf{conf}'_{i+2} = (v^{i+1}_1, v, u, v^i_2, \ldots, v^i_{k-2})$,..., $\mathsf{conf}'_{i+k} = (v^{i+k-1}_1, \ldots, v^{i+k-1}_{k-1}, v)$. Notice that, again, we derive a valid transition in $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$.

Overall, notice that by modifying the given ℓ-transition in this manner, we get (for some $t \geq \ell$) a t-transition $(\mathsf{conf}'_1, \ldots, \mathsf{conf}'_{t+1})$ in $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$, such that $\mathsf{conf}'_1 = \mathsf{init}$ and $\mathsf{conf}'_{t+1} = \mathsf{fin}$. Therefore, $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance. $\square$

By combining the last two lemmas, we immediately derive the following result.

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.*

In the last three lemmas, we assumed that the $3\sqrt{k}$-wall H contains vertices from neither init nor fin. In order to ensure that we have such a $3\sqrt{k}$-wall, we need to search for an r-wall, for a larger r, such that whenever G contains an r-wall, it also contains such H. From the next observation, a $7k$-wall is large enough.

Observation 5.3. *Let $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ be an instance of* Snake Game*. If G contains a $7k$-wall, then it contains a $3\sqrt{k}$-wall H such that H does not have any vertices from* init *or* fin*. Moreover, given a $7k$-wall, such a $3\sqrt{k}$-wall H can be found in time polynomial in k.*

Algorithm 3: TrRedAlgIter

```
1 function TrRedAlgIter($\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$);
2 Use the algorithm from Proposition 5.1 to try to find a $7k$-wall $H$ in $G$.;
3 if failed to find such $H$ then
4     Report that $G$ has treewidth $k^{\mathcal{O}(1)}$ and return;
5 end
6 Use Observation 5.3 to find a $3\sqrt{k}$-wall $H'$ in $H$ having vertices from neither init
     nor fin;
7 Choose an arbitrary edge $e \in E(H')$;
8 return $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$;
```

Proof. Let H' be a $7k$-wall in G. Notice that H' contains more than $2k$ $3\sqrt{k}$-walls that have no vertices in common (see Figure 7), and $2k+1$ such $3\sqrt{k}$-walls can be found in polynomial time. Since $|V(\mathsf{init}) \cup V(\mathsf{fin})| \leq 2k$, by the pigeonhole principle, we get that H', and so does G, contains a $3\sqrt{k}$-wall H such that H does not have any vertices from init or fin. In order to find such a $3\sqrt{k}$-wall, we simply examine iterate over our $2k+1$ different $3\sqrt{k}$-walls until the desired one is found. Therefore, a $3\sqrt{k}$-wall H such that H does not have any vertices from init or fin exists and can be found in time polynomial in k. □

5.1 The Algorithm for Reduction of Treewidth

Now, we are ready to present our algorithm for reduction of treewidth, which we call TrRedAlg. The input of TrRedAlg is an instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of Snake Game and its output should be an equivalent instance $\langle G', k, \mathsf{init}, \mathsf{fin}\rangle$ where the treewidth of G' is $k^{\mathcal{O}(1)}$. Towards that, TrRedAlg performs at most $|V(G)|$ iterations that can each be executed in time polynomial in $|V(G)|$. At each iteration, the algorithm either returns an instance $\langle G', k, \mathsf{init}, \mathsf{fin}\rangle$ equivalent to $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ such that $|V(G')| = |V(G)| - 1$, or reports that G has treewidth $k^{\mathcal{O}(1)}$. First, we present the procedure executed for each iteration, called TrRedAlgIter (see Algorithm 3). For the execution of TrRedAlgIter, we use as "black box" a known algorithm (Chekuri & Chuzhoy, 2016) that (in polynomial time) either finds an r-wall in a graph, or reports that its treewidth is $k^{\mathcal{O}(1)}$ (see Proposition 5.1).

Proposition 5.1. *There exists an algorithm that, given a graph G and a nonnegative integer t, either finds a t-wall H in G, or reports that G has treewidth $t^{\mathcal{O}(1)}$. Moreover, that algorithm runs in time $\mathcal{O}(|V(G)| + |E(G)|)$.*

Now, we aim to show that TrRedAlgIter is correct. That is, given an instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of Snake Game, if TrRedAlgIter reports that G has treewidth $k^{\mathcal{O}(1)}$, then we need to show that this is the case; otherwise, TrRedAlgIter returns an instance of Snake Game, and we need to show that this instance is equivalent to $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$.

Lemma 5.5. *Let $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ be an instance of* Snake Game. *If* TrRedAlgIter($\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$) *reports that G has treewidth $k^{\mathcal{O}(1)}$, then G has treewidth $k^{\mathcal{O}(1)}$. Otherwise,* TrRedAlgIter *returns an equivalent instance of $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$.*

Algorithm 4: TrRedAlg

```
1 function TrRedAlgIter($\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$);
2 $G^I \leftarrow G$;
3 while TrRedAlgIter($I = \langle G^I, k, \mathsf{init}, \mathsf{fin}\rangle$) does not report $G^I$ has treewidth $k^{\mathcal{O}(1)}$ do
4     $I \leftarrow$ TrRedAlgIter($I$);
5 end
6 return $I$ ;
```

Proof. First, assume that $\mathsf{TrRedAlgIter}(\langle G, k, \mathsf{init}, \mathsf{fin}\rangle)$ reports that G has treewidth $k^{\mathcal{O}(1)}$. Then, by the definition of TrRedAlgIter, the algorithm from Proposition 5.1 reported that G has treewidth $k^{\mathcal{O}(1)}$. Therefore, by the correctness of Proposition 5.1, it follows that G indeed has treewidth $k^{\mathcal{O}(1)}$.

Now, observe that if TrRedAlgIter does not report that G has treewidth $k^{\mathcal{O}(1)}$, then it returns the instance $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$. Then, by the definition of TrRedAlgIter and the correctness of Proposition 5.1 and Observation 5.3, H' is indeed a $3\sqrt{k}$-wall in G having vertices from neither init nor fin. Therefore, by Lemma 5.4, $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance if and only if $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle$ is a Yes-instance. Thus, $\langle G/e, k, \mathsf{init}, \mathsf{fin}\rangle)$ is equivalent to $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$. ☐

Now, we argue that TrRedAlgIter run in time polynomial in $|V(G)|$.

Observation 5.4. *The algorithm* TrRedAlgIter *runs in time* $\mathcal{O}(|V(G)|^2)$.

Proof. The algorithm from Proposition 5.1 runs in time $\mathcal{O}(|V(G)| + |E(G)|)$. Given a $7k$-wall, by Observation 5.3, a $3\sqrt{k}$-wall H' having vertices from neither init nor fin can be found in time polynomial in $k \leq |V(G)|$. The other steps of TrRedAlgIter can be implemented in $\mathcal{O}(1)$ time. As $|E(G)| = \mathcal{O}(|V(G)|^2)$, the observation follows. ☐

Now, we present our reduction algorithm TrRedAlg (see Algorithm 4). We denote the outputs of the algorithms TrRedAlgIter and TrRedAlg for an instance I by TrRedAlgIter(I) and TrRedAlg(I) respectively.

Now, we show that TrRedAlg runs in time polynomial in $|V(G)|$.

Lemma 5.6. *The algorithm* TrRedAlg *runs in time* $\mathcal{O}(|V(G)|^3)$.

Proof. Consider an input instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$. At each iteration $t \geq 2$, TrRedAlg calls TrRedAlgIter with an instance $\langle G^t, k, \mathsf{init}, \mathsf{fin}\rangle$ such that $|V(G^t)| = |V(G^{t-1})| - 1$. By Lemma 5.4, TrRedAlgIter works in time polynomial in $|V(G^t)|$. Therefore, as $G^1 = G$, there are at most $|V(G)|$ iterations. Thus, we conclude that TrRedAlg runs in time $\mathcal{O}(|V(G)|^3)$. ☐

Now, we show that for a given instance of Snake Game $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$, TrRedAlg $(\langle G, k, \mathsf{init}, \mathsf{fin}\rangle) = \langle G', k, \mathsf{init}, \mathsf{fin}\rangle$ is an equivalent to $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$, and also G' has treewidth $k^{\mathcal{O}(1)}$.

Lemma 5.7. *Let* $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ *be an instance of* Snake Game. *Then* TrRedAlg$(\langle G, k, \mathsf{init}, \mathsf{fin}\rangle) = \langle G', k, \mathsf{init}, \mathsf{fin}\rangle$ *is an equivalent instance of* $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$*, and* G' *has treewidth* $k^{\mathcal{O}(1)}$.

Proof. By Lemma 5.5, at each iteration of $\mathsf{TrRedAlg}$, $\mathsf{TrRedAlgIter}$ returns an instance of Snake Game, that is equivalent to the previous one. Therefore, as the first instance is $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$, the output of $\mathsf{TrRedAlg}$ denoted by $\langle G', k, \mathsf{init}, \mathsf{fin}\rangle$, is equivalent to $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$. By the definition of $\mathsf{TrRedAlg}$ and Lemma 5.5, it directly follows that G' has treewidth $k^{\mathcal{O}(1)}$. □

Finally, we combine the last two lemmas to conclude the correctness of our algorithm of the reduction.

Theorem 5.1. *Given any instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of* Snake Game*,* $\mathsf{TrRedAlg}$ *returns an equivalent instance $\langle G', k, \mathsf{init}, \mathsf{fin}\rangle$ such that G' has treewidth $k^{\mathcal{O}(1)}$. Moreover,* $\mathsf{TrRedAlg}$ *runs in time $\mathcal{O}(|V(G)|^3)$.*

Observe that, a graph G on n vertices and with treewidth $\mathtt{tw}$ has $\mathcal{O}(\mathtt{tw}^2 n)$ edges as the tree decomposition of G can have at most n bags and every bag contributes at most $\mathtt{tw}^2$ edges. Thus, the following corollary follows directly from the Theorem 5.1.

Corollary 5.1. *Given any instance $\langle G, k, \mathsf{init}, \mathsf{fin}\rangle$ of* Snake Game*,* $\mathsf{TrRedAlg}$ *returns an equivalent instance $\langle G', k, \mathsf{init}, \mathsf{fin}\rangle$ such that $|E(G')| = k^{\mathcal{O}(1)}|V(G)'|$.*

6. Conclusion and Directions for Future Research

In this paper, we presented a coherent picture of the parameterized complexity of the Snake Game problem. As our main contribution, we proved that Snake Game is FPT. This result was complemented by the proofs that Snake Game is unlikely to admit a polynomial kernel even on grid graphs, but admits a treewidth-reduction procedure. For our main contribution, we presented a novel application of the method of color-coding by utilizing it to sparsify the configuration graph of the problem. To some extent, our paper can be considered as pioneering work in the study of the parameterized complexity of motion planning problems where intermediate configurations are of importance (which has so far been largely neglected), and may lay the foundations for further research of this topic. In this regard, the choice of Snake Game was a natural starting point given that it involves only one mobile object as well as captures only the most basic property of a physical object—it cannot intersect itself and other obstacles.

We conclude the paper with a few directions for further research. As a first step to broaden the scope of this kind of research, we suggest to consider the parameterized complexity of motion planning problems that involve several mobile agents. Possibly, not all mobile agents can be controlled—that is, the patterns of motion of some of them may be predefined. The controllable mobile agents themselves can have either a joint objective or independent tasks. Arguably, the most basic requirement of the mobile agents is to avoid collision with one another while they perform these tasks. We remark that a natural parameter in this regard is the number of mobile agents. Second, apart (or in addition to) the study of multiple agents, we also find it interesting to consider other types of motion as well as other shapes of robots. Moreover, one can study objectives that are more complicated than merely reaching one position from another, particularly objectives where the mobile agent can effect the environment rather than only move within it. Lastly, we also suggest to consider the study of the parameterized complexity of motion planning problems in geometric settings in 3D—while graphs provide a useful abstraction, some problems may require a more accurate representation of the robots and physical environment.

Acknowledgments

A preliminary version of this paper appeared in the proceedings of IJCAI 2019. The research of the first author is supported in part by the Zuckerman STEM Leadership Program. The research of the third author is supported in part by the ISF grant no. 1176/18 and BSF grant no. 2018302.

References

Alon, N., Yuster, R., & Zwick, U. (1995). Color-coding. *J. ACM*, *42*(4), 844–856.

Berthet-Rayne, P., Gras, G., Leibrandt, K., Wisanuvej, P., Schmitz, A., Seneci, C. A., & Yang, G.-Z. (2018). The i2snake robotic platform for endoscopic surgery. *Annals of Biomedical Engineering*, *46*(10), 1663–1675.

Biasi, M. D., & Ophelders, T. (2018). The complexity of snake and undirected NCL variants. *Theor. Comput. Sci.*, *748*, 55–65.

Bodlaender, H. L., Downey, R. G., Fellows, M. R., & Hermelin, D. (2009). On problems without polynomial kernels. *J. Comput. Syst. Sci.*, *75*(8), 423–434.

Bodlaender, H. L., Jansen, B. M. P., & Kratsch, S. (2014). Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, *28*(1), 277–305.

Canny, J. F., & Reif, J. H. (1987). New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pp. 49–60. IEEE Computer Society.

Cesati, M., & Wereham, H. T. (1995). Parameterized complexity analysis in robot motion planning. In *Proceedings of the 25th IEEE International Conference on Systems, Man and Cybernetics, Volume 1, October, 23-25, 1995, Vancouver, Canada*, pp. 1–6.

Chekuri, C., & Chuzhoy, J. (2016). Polynomial bounds for the grid-minor theorem. *J. ACM*, *63*(5), 40:1–40:65.

Chuzhoy, J., & Tan, Z. (2019). Towards tight(er) bounds for the excluded grid theorem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 1445–1464.

Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., & Saurabh, S. (2015). *Parameterized Algorithms*. Springer.

de Haan, R., & Szeider, S. (2017). Parameterized complexity classes beyond para-np. *J. Comput. Syst. Sci.*, *87*, 16–57.

Degani, A., Choset, H., Wolf, A., & Zenati, M. A. (2006). Highly articulated robotic probe for minimally invasive surgery. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4167–4172.

Demaine, E. D., Eppstein, D., Hesterberg, A., Jain, K., Lubiw, A., Uehara, R., & Uno, Y. (2019). Reconfiguring undirected paths. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, pp. 353–365.

Demaine, E. D., Fekete, S. P., Keldenich, P., Scheffer, C., & Meijer, H. (2018). Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch. In *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, pp. 29:1–29:15.

Demaine, E. D., Grosof, I., & Lynch, J. (2017). Push-pull block puzzles are hard. In *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pp. 177–195.

Demaine, E. D., Hajiaghayi, M. T., & Marx, D. (2014). Minimizing movement: Fixed-parameter tractability. *ACM Trans. Algorithms*, *11*(2), 14:1–14:29.

Diestel, R. (2012). *Graph Theory, 4th Edition*, Vol. 173 of *Graduate texts in mathematics*. Springer.

Downey, R. G., & Fellows, M. R. (2013). *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Eiben, E., & Kanj, I. A. (2018). How to navigate through obstacles?. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pp. 48:1–48:13.

Farber, M. (2017). Configuration spaces and robot motion planning algorithms. *Lecture Notes Series, Institute for Mathematical Sciences*, *35*.

Fomin, F., Lokshtanov, D., Saurabh, S., & Zehavi, M. (2018). *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press.

Galceran, E., & Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, *61*(12), 1258–1276.

Grifantini, K. (2008). Snakelike robots for heart surgery. *MIT Technology Review*.

Halperin, D., Salzman, O., & Sharir, M. (2017). Algorithmic motion planning. In *Handbook of Discrete and Computational Geometry*, p. 1311–1342.

Hearn, R. A., & Demaine, E. D. (2009). *Games, puzzles and computation*. A K Peters.

Kendall, G., Parkes, A. J., & Spoerer, K. (2008). A survey of np-complete puzzles. *ICGA Journal*, *31*(1), 13–34.

Lu, Z., Feng, D., Xie, Y., Xu, H., Mao, L., Shan, C., Li, B., Bilik, P., Zidek, J., Martinek, R., & Rykala, Z. (2016). Study on the motion control of snake-like robots on land and in water. *Perspectives in Science*, *7*, 101 – 108. 1st Czech-China Scientific Conference 2015.

Naor, M., Schulman, L. J., & Srinivasan, A. (1995). Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pp. 182–191.

Paden, B., Cáp, M., Yong, S. Z., Yershov, D. S., & Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. Intelligent Vehicles*, *1*(1), 33–55.

Pan, G., & Vardi, M. Y. (2006). Fixed-parameter hierarchies inside PSPACE. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pp. 27–36.

Papadimitriou, C. H., & Vazirani, U. V. (1984). On two geometric problems related to the traveling salesman problem. *J. Algorithms*, *5*(2), 231–246.

Pfotzer, L., Klemm, S., Roennau, A., Zöllner, J., & Dillmann, R. (2017). Autonomous navigation for reconfigurable snake-like robots in challenging, unknown environments. *Robotics and Autonomous Systems*, *89*, 123 – 135.

Reif, J. H. (1979). Complexity of the mover's problem and generalizations (extended abstract). In *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*, pp. 421–427.

Robertson, N., & Seymour, P. D. (1995). Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, *63*(1), 65–110.

Robertson N., Seymour, P. (1995). Graph minors. xiii. the disjoint path problem. *J. Combinatorial Theory Ser.*, 65–110.

Schwartz, J. T., & Sharir, M. (1988). A survey of motion planning and related geometric algorithms. *Artif. Intell.*, *37*(1-3), 157–169.

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

Yan, Z., Jouandeau, N., & Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, *10*(12), 399.

Yang, L., Qi, J., Song, D., Xiao, J., Han, J., & Xia, Y. (2016). Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, *2016*(7426913), 22.

Ye, C., Ma, S., Li, B., & Wang, Y. (2004). Turning and side motion of snake-like robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5075–5080.