The Propagation Depth of Local Consistency

Christoph Berkholz

RWTH Aachen University, Aachen, Germany

Abstract. We establish optimal bounds on the number of nested propagation steps in k-consistency tests. It is known that local consistency algorithms such as arc-, path- and k-consistency are not efficiently parallelizable. Their inherent sequential nature is caused by long chains of nested propagation steps, which cannot be executed in parallel. This motivates the question "What is the minimum number of nested propagation steps that have to be performed by k-consistency algorithms on (binary) constraint networks with n variables and domain size d?"

It was known before that 2-consistency requires $\Omega(nd)$ and 3-consistency requires $\Omega(n^2)$ sequential propagation steps. We answer the question exhaustively for every $k \geq 2$: there are binary constraint networks where any k-consistency procedure has to perform $\Omega(n^{k-1}d^{k-1})$ nested propagation steps before local inconsistencies were detected. This bound is tight, because the overall number of propagation steps performed by k-consistency is at most $n^{k-1}d^{k-1}$.

1 Introduction

A constraint network (X, D, C) consists of a set X of n variables over a domain D of size d and a set of constraints C that restrict possible assignments of the variables. The *constraint satisfaction problem* (CSP) is to find an assignment of the variables with values from D such that all constraints are satisfied. The constraint satisfaction problem can be solved in exponential time by exhaustive search over all possible assignments. *Constraint propagation* is a technique to speed up the exhaustive search by restricting the search space in advance. This is done by iteratively propagating new constraints that follow from previous ones. Most notably, in local consistency algorithms the overall goal is to propagate new constraints to achieve some kind of consistency on small parts of the constraint network. Additionally, if local inconsistencies were detected, it follows that the constraint network is also globally inconsistent and hence unsatisfiable.

The k-consistency test [8] is a well-known local consistency technique, which enforces that every satisfying $(k-1)$-partial assignment can be extended to a satisfying k-partial assignment. At the beginning, all partial assignments that violate a constraint were marked as inconsistent. Then the following inference rule is applied iteratively:

> If h is a consistent ℓ-partial assignment ($\ell < k$) for which there exists a variable $x \in X$ such that $h \cup \{x \mapsto a\}$ is inconsistent for all $a \in D$, then mark h and all its extensions as inconsistent.

B. O'Sullivan (Ed.): CP 2014, LNCS 8656, pp. 158–173, 2014.

After at most $n^{k-1}d^{k-1}$ propagation steps this procedure stops. If the empty assignment becomes inconsistent, we say that (strong) k-consistency *cannot be established*. In this case we know that the constraint network is globally inconsistent. Otherwise, if k-consistency *can be established*, we can use the propagated constraints to restrict the search space for a subsequent exhaustive search. There are several different k-consistency algorithms in the literature, especially for $k = 2$ (arc consistency) and $k = 3$ (path consistency), which all follow this propagation scheme. The main difference between these algorithms are the underlying data structure and the order in which they apply the propagation rule. It seems plausible to apply the propagation rule in parallel in order to detect local inconsistencies in different parts of the constraint network at the same time. Indeed, this intuition has been used to design parallel arc and path consistency algorithms [15,16]. On the other hand, the k-consistency test is known to be PTIME-complete [10,11] and hence not efficiently parallelizable (unless NC=PTIME). The main bottleneck for parallel approaches are the sequential dependencies in the propagation rule: some assignments will be marked as inconsistent after some other assignments became inconsistent.

For 2-consistency the occurrence of long chains of sequential dependencies has been observed very early [6] and was recently studied in depth in [4]. There are simple constraint networks for which 2-consistency requires $\Omega(nd)$ nested propagation steps. Ladkin and Maddux [14] used algebraic techniques to show that 3-consistency requires $\Omega(n^2)$ nested propagation steps on binary constraint networks with constant domain. We extend these previous results and obtain a complete picture of the propagation depth of k-consistency. Our main result (Theorem 1) states that for every constant $k \geq 2$ and given integers n, d there is a constraint network with n variables and domain size d such that every k-consistency algorithm has to perform $\Omega(n^{k-1}d^{k-1})$ nested propagation steps. This lower bound is optimal as it is matched by the trivial upper bound $n^{k-1}d^{k-1}$ on the overall number of propagation steps. It follows that every parallel propagation algorithm for k-consistency has a worst case time complexity of $\Omega(n^{k-1}d^{k-1})$. Since the best-known running time of a sequential algorithm for k-consistency is $O(n^k d^k)$ [5] it follows that no significant improvement over the sequential algorithm is possible.

2 Preliminaries

As first pointed out by Feder and Vardi [7] the CSP is equivalent to the structure homomorphism problem where two finite relational structures A and B are given as input. The universe $V(\mathsf{A})$ of structure A corresponds to the set of variables X and the universe $V(\mathsf{B})$ of structure B corresponds to the domain D. The constraints are encoded into relations such that every homomorphism from A to B corresponds to a solution of the CSP. For the rest of this paper we mainly stick to this definition as it is more convenient to us. In fact, our main result benefits to a large extend from the fruitful connection between these two viewpoints.

In the introduction we have presented k-consistency as a propagation procedure on constraint networks. Below we restate the definition in terms of a formal

inference system (which is inspired by the proof system in [1] and is a generalization of [4]). This view allows us to gain insight into the structure of the propagation process and to formally state our main theorem afterwards. At the end of this section we provide a third characterization of k-consistency in terms of the existential pebble game, which is the tool of our choice in the proof of the main theorem.

2.1 CSP-Refutations

Given two σ-structures A and B, every line of our derivation system is a partial mapping from $V(\mathsf{A})$ to $V(\mathsf{B})$. The axioms are all partial mappings $p\colon V(\mathsf{A}) \to V(\mathsf{B})$ that are not partial homomorphisms. We have the following derivation rule to derive a new inconsistent assignment p. For all partial mappings $p'_i \subseteq p$, $x \in V(\mathsf{A})$ and $V(\mathsf{B}) = \{a_1, \dots, a_n\}$:

$$\frac{p'_1 \cup \{x \mapsto a_1\} \quad \cdots \quad p'_n \cup \{x \mapsto a_n\}}{p} \tag{1}$$

A *CSP-derivation* of p is a sequence $(p_1, \dots, p_\ell = p)$ such that every p_i is either an axiom or derived from lines p_j, $j < i$, via the derivation rule (1). A *CSP-refutation* is a CSP-derivation of $\emptyset$. Every derivation of p can naturally be seen as a directed acyclic graph (dag) where the nodes are labeled with lines from the derivation, one node of in-degree 0 is labeled with p and all nodes of out-degree 0 are labeled with axioms. If p_i is derived from $p_{j_1}, \dots, p_{j_n}$ using (1), then there is an arc from p_i to each $p_{j_1}, \dots, p_{j_n}$.

Given a CSP-derivation P, we let $\mathrm{Prop}(P)$ be the set of propagated mappings $p \in P$, i. e. all lines in the derivation that are not axioms. We define the *width* of a derivation P to be $\mathrm{width}(P) = \max_{p \in \mathrm{Prop}(P)} |p|$.[1] Furthermore, $\mathrm{depth}(P)$ denotes the *depth* of P which is the number of edges on the longest path in the dag associated with P. This measure characterizes the maximum number of nested propagation steps in P. Since CSP-derivations model the propagation process mentioned in the introduction, there is a CSP-refutation of width $k-1$ if and only if k-consistency cannot be established.

Furthermore, every propagation algorithm produces some CSP-derivation P. The total number of propagation steps performed by this algorithm is $|\mathrm{Prop}(P)|$ and the maximum number of nested propagation steps is $\mathrm{depth}(P)$. Let A and B be two relational structures such that k-consistency cannot be established. We define the *propagation depth* $\mathrm{depth}^k(\mathsf{A}, \mathsf{B}) := \min_P \mathrm{depth}(P)$ where the minimum is taken over all CSP-refutations P of width at most $k-1$. Hence, the $\mathrm{depth}^k(\mathsf{A}, \mathsf{B}) \leq |V(\mathsf{A})|^{k-1}|V(\mathsf{B})|^{k-1}$ is the number of sequential propagation steps that have to be performed by any sequential or parallel propagation algorithm for k-consistency.

[1] Note that this implies $|p| \leq \mathrm{width}(P) + 1$ for all axioms p used in the derivation P. However, the size of the axioms can always be bounded by the maximum arity of the relations in A and B.

2.2 Results and Related Work

Our main theorem is a tight lower bound on the propagation depth.

Theorem 1. *For every integer $k \geq 2$ there exists a constant $\varepsilon > 0$ and two positive integers n_0, m_0 such that for every $n \geq n_0$ and $m \geq m_0$ there exist two binary structures A_n and B_m with $|V(\mathsf{A}_n)| = n$ and $|V(\mathsf{B}_m)| = m$ such that* $\mathrm{depth}^k(\mathsf{A}_n, \mathsf{B}_m) \geq \varepsilon n^{k-1} m^{k-1}$.

We are aware of two particular cases that have been discovered earlier. First, for the case $k = 2$ (arc consistency) the theorem can be shown by rather simple examples that occurred very early in the AI-community. The structure of this exceptional case is discussed in deep in a joint work of Oleg Verbitsky and the author of this paper [4]. Second, for $k = 3$ Ladkin and Maddux [14] showed that there is a fixed finite binary structure B and an infinite sequence of binary structures A_i such that $\mathrm{depth}^3(\mathsf{A}_i, \mathsf{B}) = \Omega(|V(\mathsf{A}_i)|^2)$. They used this result to argue that every parallel propagation algorithm for path consistency needs at least a quadratic number of steps. This is tight only for fixed structures B, Theorem 1 extends their result to the case when B is also given as input.

Other related results investigate the decision complexity of the k-consistency test. To address this more general question one analyzes the computational complexity of the following decision problem.

k-Cons
Input: Two binary relational structures A and B.
Question: Can k-consistency be established for A and B?

Kasif [10] showed that 2-Cons is complete for PTIME under LOGSPACE reductions. Kolaitis and Panttaja [11] extended this result to every fixed $k \geq 2$. Moreover, they established that the problem is complete for EXPTIME if k is part of the input. In [3] the author showed that k-Cons cannot be decided in $O(n^{\frac{k-3}{12}})$ on deterministic multi-tape Turing machines, where n is the overall input size. Hence, any algorithm solving k-Cons (regardless of whether it performs constraint propagation or not) cannot be much faster than the standard propagation approach. It also follows from this result that, parameterized by the number of pebbles k, k-Cons is is complete for the parameterized complexity class XP. It is also worth noting that Gaspers and Szeider [9] investigated the parameterized complexity of other parameterized problems related to k-consistency.

2.3 The Existential Pebble Game

In this paragraph we introduce a third view on the k-consistency heuristic in terms of a combinatorial pebble game. The *existential k-pebble game* [12] is played by two players *Spoiler* and *Duplicator* on two relational structures A and B. There are k pairs of pebbles $(p_1, q_1), \ldots, (p_k, q_k)$ and during the game Spoiler moves the pebbles $p_1, \ldots, p_k$ to elements of $V(\mathsf{A})$ and Duplicator moves

the pebbles $q_1, \ldots, q_k$ to elements of $V(\mathsf{B})$. At the beginning of the game, Spoiler places pebbles $p_1, \ldots, p_k$ on elements of $V(\mathsf{A})$ and Duplicator answers by putting pebbles $q_1, \ldots, q_k$ on elements of $V(\mathsf{B})$. In each further round Spoiler picks up a pebble pair (p_i, q_i) and places p_i on some element in $V(\mathsf{A})$. Duplicator answers by moving the corresponding pebble q_i to one element in $V(\mathsf{B})$. Spoiler wins the game if he can reach a position where the mapping defined by $p_i \mapsto q_i$ is not a partial homomorphism from A to B.

The connection between the existential k-pebble game and the k-consistency heuristic was made by Kolaitis and Vardi [13]. They showed that one can establish k-consistency by computing a winning strategy for Duplicator. Going a different way, the next lemma states that there is also a tight correspondence between Spoiler's strategy and CSP-refutations. The proof is a straightforward induction over the depth and included in the full version of the paper [2].

Lemma 2. *Let* A *and* B *be two relational structures. There is a CSP-refutation for* A *and* B *of width* $k-1$ *and depth* d *if and only if Spoiler has a strategy to win the existential* k*-pebble game on* A *and* B *within* d *rounds.*

Using this lemma it suffices to prove lower bounds on the number of rounds in the existential pebble game in order to prove Theorem 1. To argue about strategies in the existential pebble game we use the framework developed in [3]. We start with a formal definition of strategies for Duplicator.

Definition 3. *A* critical strategy *for Duplicator in the existential* k*-pebble game on structures* A *and* B *is a nonempty family* $\mathcal{H}$ *of partial homomorphisms from* A *to* B *together with a set* $\mathrm{crit}(\mathcal{H}) \subseteq \mathcal{H}$ *of critical positions satisfying the following properties:*

1. *All critical positions are* $(k-1)$*-partial homomorphisms.*
2. *If* $h \in \mathcal{H}$ *and* $g \subset h$*, then* $g \in \mathcal{H}$*.*
3. *For every* $g \in \mathcal{H} \setminus \mathrm{crit}(\mathcal{H})$*,* $|g| < k$*, and every* $x \in V(\mathsf{A})$ *there is an* $a \in V(\mathsf{B})$ *such that* $g \cup \{x \mapsto a\} \in \mathcal{H}$*.*

If $\mathrm{crit}(\mathcal{H}) = \emptyset$*, then* $\mathcal{H}$ *is a* winning strategy*.*

The set $\mathcal{H}$ is the set of good positions for Duplicator (therefore they are all partial homomorphisms). Non-emptiness and the closure property (2.) ensure that $\mathcal{H}$ contains the start position $\emptyset$. Furthermore, the closure property guarantees that the current position remains a good position for Duplicator when Spoiler picks up pebbles. The extension property (3.) ensures that, from every non-critical position, Duplicator has an appropriate answer if Spoiler puts a free pebble on x. It follows that if there are no critical positions, then Duplicator can always answer accordingly and thus wins the game. Otherwise, if Spoiler reaches a critical position, then Duplicator may not have an appropriate answer and the game reaches a critical state. In the next lemma we describe how to use critical strategies to prove lower bounds on the number of rounds.

Lemma 4. *If* $\mathcal{H}_1, \ldots, \mathcal{H}_l$ *is a sequence of critical strategies on the same pair of structures and for all* $i < l$ *and all* $p \in \mathrm{crit}(\mathcal{H}_i)$ *it holds that* $p \in \mathcal{H}_j \setminus \mathrm{crit}(\mathcal{H}_j)$ *for some* $j \leq i+1$*, then Duplicator wins the* l*-round existential* k*-pebble game.*

Proof. Starting with $i = 1$, Duplicator answers according to the extension property of $\mathcal{H}_i$, if the current position p is non-critical in $\mathcal{H}_i$. Otherwise, p is non-critical in $\mathcal{H}_j$ for some $j \leq i+1$ and Duplicator answers according to the extension property of $\mathcal{H}_j$. This allows Duplicator to survive for at least l rounds. □

The two structures A and B we construct are vertex colored graphs. They are built out of smaller graphs, called *gadgets*. Every gadget Q consists of two graphs Q_S and Q_D for Spoiler's and Duplicator's side, respectively. Hence, Q_S and Q_D will be subgraphs of A and B in the end. The gadgets contain *boundary vertices*, which are the vertices shared with other gadgets. To combine two strategies on two connected gadgets we need to ensure that the strategies agree on the boundary of the gadgets. Formally, let a *boundary function* of a strategy $\mathcal{H}$ on a gadget Q be a mapping β from the boundary of Q_S to the boundary of Q_D such that $\beta(z) = h(z)$ for all $h \in \mathcal{H}$ and all z in the domain of β and h. We say that two strategies $\mathcal{G}$ and $\mathcal{H}$ on gadgets Q and Q' are *connectable*, if their boundary functions agree on the common boundary vertices of Q and Q'. If $\mathcal{G}$ and $\mathcal{H}$ are two connectable critical strategies on gadgets $Q = (Q_S, Q_D)$ and $Q' = (Q'_S, Q'_D)$ it is not hard to see that the *composition*

$$\mathcal{G} \uplus \mathcal{H} = \{g \cup h \mid g \in \mathcal{G}, h \in \mathcal{H}\}$$

is a critical strategy on $Q_S \cup Q'_S$ and $Q_D \cup Q'_D$ with $\mathrm{crit}(\mathcal{G} \uplus \mathcal{H}) = \mathrm{crit}(\mathcal{G}) \cup \mathrm{crit}(\mathcal{H})$. Intuitively, playing according to the strategy $\mathcal{G} \uplus \mathcal{H}$ on Q and Q' means that Duplicator uses strategy $\mathcal{G}$ on Q and strategy $\mathcal{H}$ on Q'.

3 The Construction

3.1 Overview of the Construction

In this section we prove Theorem 1 for $k \geq 3$. We let $\mathsf{k} := k-1 \geq 2$ and construct two vertex colored graphs A_n and B_m with $O(n)$ and $O(m)$ vertices such that Spoiler needs $\Omega(n^{\mathsf{k}} m^{\mathsf{k}})$ rounds to win the existential $(\mathsf{k}+1)$-pebble game. We color the vertices of both graphs such that the colors partition the vertex set into independent sets, i. e. every vertex gets one color and there is no edge between vertices of the same color. The basic building blocks in our construction are sets of vertices which allow to store $n^{\mathsf{k}} m^{\mathsf{k}}$ partial homomorphisms with k pebbles.

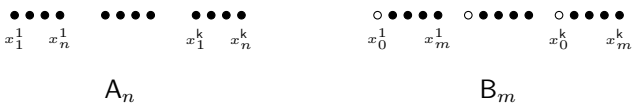

Fig. 1. Basic vertex blocks. Two vertices x_j^i and $x_{j'}^{i'}$ get the same color iff $i = i'$.

We introduce vertices x_j^i ($i \in [\mathsf{k}]$, $j \in [n]$) in A_n and vertices x_j^i ($i \in [\mathsf{k}]$, $j \in [m] \cup \{0\}$) in B_m. For every $i \in [\mathsf{k}]$ the vertices x_j^i form a *block* and are

colored with the same color (say P_{x^i}), which is different from any other color in the entire construction. The vertices x_0^i in structure B_m play a special role in our construction and are visualized by $\circ$ instead of $\bullet$ in the pictures. However, they are colored with the same color P_{x^i} as the other vertices x_j^i. Because of the coloring, Duplicator has to answer with some $x_{j'}^i$ whenever Spoiler pebbles a vertex x_j^i. Since there are nm positions for one pebble pair on $\bullet$ vertices in one block, we get $n^{\mathsf{k}}m^{\mathsf{k}}$ positions if every block has exactly one pebble pair on $\bullet$ vertices. The $\circ$ vertices are used by Duplicator whenever Spoiler does not play the intended way. That is, if Spoiler pebbles a vertex in block i that he is not supposed to pebble now, then Duplicator answers with x_0^i. The construction will have the property that this is always a good situation for Duplicator.

To describe pebble positions on such vertex blocks, we define mappings $\mathfrak{a}\colon [\mathsf{k}] \to [n]$ and $\mathfrak{b}\colon [\mathsf{k}] \to [m]$ and call the pebble position $\{(x^i_{\mathfrak{a}(i)}, x^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$ *valid*. If such valid position is on the board, then Duplicator answers with $x^i_{\mathfrak{b}(i)}$ if Spoiler pebbles $x^i_{\mathfrak{a}(i)}$ and with x_0^i if Spoiler pebbles x_j^i for some $j \neq \mathfrak{a}(i)$. We also need to name positions where Duplicator answers with x_0^i for every vertex in block i and let T be the set of blocks where this happens. For $\mathfrak{a}\colon [\mathsf{k}] \to [n]$, $\mathfrak{b}\colon [\mathsf{k}] \to [m]$ and $T \subseteq [\mathsf{k}]$ we call $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ a *configuration*. The configuration $\mathfrak{q}$ is *valid* if $T = \emptyset$ and *invalid* otherwise. For every configuration $\mathfrak{q}$ and a set of x_j^i vertices as in Figure 1 we define the following homomorphism that describes Duplicator's behavior:

$$h^x_{\mathfrak{q}}(x_j^i) = \begin{cases} x^i_{\mathfrak{b}(i)}, \text{ if } j = \mathfrak{a}(i) \text{ and } i \notin T, \\ x_0^i, \text{ otherwise.} \end{cases}$$

By $h^x_{\mathbf{0}}$ we denote the homomorphism $h^x_{\mathbf{0}}(x_j^i) := x_0^i$ for all $i \in [\mathsf{k}], j \in [n]$. We say that a position of (at most $\mathsf{k}+1$) pebble pairs on these vertices is *invalid* if it is a subset of $h^x_{\mathfrak{q}}$ for some invalid configuration $\mathfrak{q}$. For valid configurations $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ we say "$\mathfrak{q}$ on x" to name the valid pebble position $\{(x^i_{\mathfrak{a}(i)}, x^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$. Note that valid pebble positions are not invalid.[2]

In the entire construction there is one unique copy of the x_j^i-vertices, which are denoted by x_j^i. Our goal is to force Spoiler to pebble every valid position on x before he wins the game. He is supposed to do so in a specific predefined order. To fix this order we define a bijection α between valid configurations $(\mathfrak{a}, \mathfrak{b}, \emptyset)$ and the numbers $0, \ldots, n^{\mathsf{k}}m^{\mathsf{k}} - 1$:

$$\alpha(\mathfrak{q}) := m^{\mathsf{k}} \sum_{i=1}^{\mathsf{k}} (\mathfrak{a}(i) - 1) n^{\mathsf{k}-i} + \sum_{i=1}^{\mathsf{k}} (\mathfrak{b}(i) - 1) m^{\mathsf{k}-i}.$$

Thus, $\alpha(\mathfrak{q})$ is the rank of the tuple $(\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \mathfrak{b}(1), \ldots, \mathfrak{b}(\mathsf{k}))$ in lexicographical order. If $\alpha(\mathfrak{q}) < n^{\mathsf{k}}m^{\mathsf{k}} - 1$, we define the successor $\mathfrak{q}^+ = (\mathfrak{a}^+, \mathfrak{b}^+, \emptyset)$ to be the unique valid configuration satisfying $\alpha(\mathfrak{q}^+) = \alpha(\mathfrak{q}) + 1$. In the sequel we introduce gadgets to make sure that:

[2] There are pebble positions on the x_j^i vertices that are neither valid nor invalid. However, such positions will not occur in our strategies.

– Spoiler can reach the position $\alpha^{-1}(0)$ on x from $\emptyset$,
– Spoiler can reach $\alpha^{-1}(i+1)$ on x from $\alpha^{-1}(i)$ on x and
– Spoiler wins from $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on x.

If we have these properties, we know that Spoiler has a winning strategy in the $(\mathsf{k}+1)$-pebble game. To show that Spoiler needs at least $n^{\mathsf{k}}m^{\mathsf{k}}$ rounds we argue that this is essentially the only way for Spoiler to win the game.

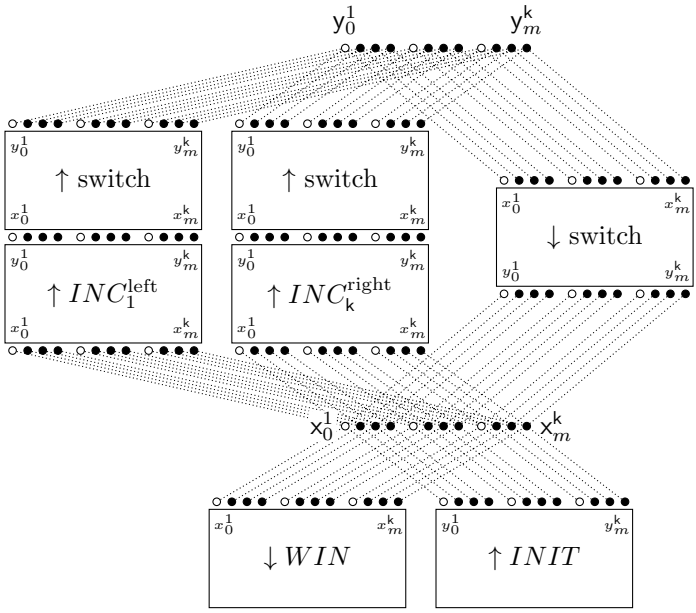

Fig. 2. The graph B_m. The boundaries of the gadgets are connected as indicated by the dotted lines (which need to be contracted). The arrows point from the input to the output vertices of the gadgets.

We start with an overview of the gadgets and how they are glued together to form the structures A_n and B_m. The boundary of our gadgets consists of *input* vertices and *output* vertices. For every gadget the set of input (output) vertices is a copy of the vertex set in Figure 1 and we write x_j^i (y_j^i) to name them. This enables us to glue together the gadgets at their input and output vertices. The overall construction for the graph B_m is shown in Figure 2. The schema for A_n is similar, it contains Spoiler's side of the corresponding gadgets which are glued together the same way as in B_m (just replace m by n and drop the $\circ$ vertices). There are four types of gadgets: the initialization gadget, the winning gadget, several increment gadgets and the switch.

The *initialization gadget* ensures that Spoiler can reach $\alpha^{-1}(0)$ on x, i. e. the pebble position $\{(\mathsf{x}_1^1, \mathsf{x}_1^1), \ldots, (\mathsf{x}_1^k, \mathsf{x}_1^k)\}$. This gadget has only output boundary vertices and is used by Spoiler at the beginning of the game. There are *increment*

gadgets $\mathrm{INC}_i^{\mathrm{left}}$ and $\mathrm{INC}_i^{\mathrm{right}}$ for all $i \in [\mathsf{k}]$. The input vertices of every increment gadget are identified with the x vertices as depicted in Figure 2. The increment gadgets (all together) ensure that Spoiler can increment a configuration. More precisely, for every valid configuration $\mathfrak{q}$ with $\alpha(\mathfrak{q}) < n^{\mathsf{k}}m^{\mathsf{k}} - 1$, there is one increment gadget INC such that Spoiler can reach $\mathfrak{q}^+$ on the output of INC from $\mathfrak{q}$ on the input. Every increment gadget is followed by a copy of the *switch*. The input of $2\mathsf{k}$ switches is identified with the output of the $2\mathsf{k}$ increment gadgets and the output of these switches is identified with a unique block of y-vertices and the input of one additional *single switch* (see Figure 2). The output of this switch is in turn identified with the unique block of x-vertices. The switches are used to perform the transition in the game from $\alpha^{-1}(i)$ on x to $\alpha^{-1}(i+1)$ on x. Spoiler can pebble a valid position through one switch: from $\mathfrak{q}$ on the input of a switch Spoiler can reach $\mathfrak{q}$ on the output of that switch. Hence, Spoiler can simply pebble the incremented position $\alpha^{-1}(i+1)$ from the output of an increment gadget through two switches to the x-block.

Finally, the *winning gadget* ensures that from $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on x Spoiler wins the game. The winning gadget has only input vertices, which are identified with the x-vertices. From $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on the input, Spoiler can win the game by playing on this gadget. On the other hand, the gadget ensures that Spoiler can *only* win from $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on x and Duplicator does not lose from any other configuration on x.

3.2 The Gadgets

We now describe the winning gadget and the increment gadgets in detail and provide strategies for Spoiler and Duplicator on them. Afterwards we briefly discuss the switch and the initialization gadget. In the next section we combine the partial strategies on the gadgets to prove Theorem 1.

The *winning gadget* is shown in Figure 3. On Spoiler's side there is just one additional vertex a, which is connected to x_n^i for all $i \in [\mathsf{k}]$. On Duplicator's side there are k additional vertices $a^i, i \in [\mathsf{k}]$. Every a^i is connected to all input vertices except x_m^i. We use one new vertex color to color the vertex a and all vertices a_i. From the position $\{(x_n^1, x_m^1), \ldots, (x_n^k, x_m^k)\}$ "$\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on x" Spoiler wins the game by placing the $(\mathsf{k}+1)$st pebble on a. Duplicator has to answer with some a_i (because of the coloring). Since there is an edge between x_n^i and a in WIN_S but none between x_m^i and a_i in WIN_D, Spoiler wins immediately. It is also not hard to see that for any other position where at least one pebble pair (x_n^j, x_m^j) is missing Duplicator can survive by choosing a_j.

The *increment gadgets* enable Spoiler to reach the successor $\mathfrak{q}^+$ from $\mathfrak{q}$. Recall that we identify every valid configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ with the tuple $(\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \mathfrak{b}(1), \ldots, \mathfrak{b}(\mathsf{k})) \in [n]^k \times [m]^k$ and define $\alpha(\mathfrak{q})$ to be the rank (from 0 to $n^k m^k - 1$) of this tuple in lexicographical order. Let $\mathfrak{q}$ be a valid configuration with $\alpha(\mathfrak{q}) < n^{\mathsf{k}}m^{\mathsf{k}} - 1$ and successor $\mathfrak{q}^+ = (\mathfrak{a}^+(1), \ldots, \mathfrak{a}^+(\mathsf{k}), \mathfrak{b}^+(1), \ldots, \mathfrak{b}^+(\mathsf{k}))$. We use two types of increment gadgets, *left* and *right*, depending on whether the left-hand side of the tuple changes after incrementation or not. There are k increment gadgets of each type. Spoiler uses them depending on which position

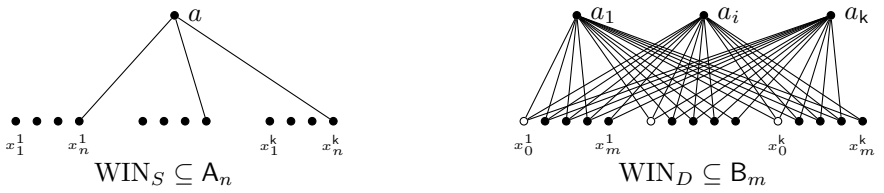

Fig. 3. The winning gadget

the last carryover occurs. If

$$\mathfrak{q} = (\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \quad \mathfrak{b}(1), \ldots, \mathfrak{b}(\ell-1), \quad \mathfrak{b}(\ell) < m, \quad m, \ldots, m) \text{ and hence}$$
$$\mathfrak{q}^+ = (\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \quad \mathfrak{b}(1), \ldots, \mathfrak{b}(\ell-1), \quad \mathfrak{b}(\ell)+1, \quad 1, \ldots, 1),$$

then Spoiler uses the increment gadget $\mathrm{INC}_\ell^{\mathrm{right}}$ to reach $\mathfrak{q}^+$ on the output from $\mathfrak{q}$ on the input. If

$$\mathfrak{q} = (\mathfrak{a}(1), \ldots, \mathfrak{a}(\ell-1), \quad \mathfrak{a}(\ell) < n, \quad n, \ldots, n, \quad m, \ldots, m) \text{ and hence}$$
$$\mathfrak{q}^+ = (\mathfrak{a}(1), \ldots, \mathfrak{a}(\ell-1), \quad \mathfrak{a}(\ell)+1, \quad 1, \ldots, 1, \quad 1, \ldots, 1),$$

then Spoiler uses $\mathrm{INC}_\ell^{\mathrm{left}}$. Thus, for every valid configuration $\mathfrak{q}$ with $\alpha(\mathfrak{q}) < n^{\mathsf{k}} m^{\mathsf{k}} - 1$ there is exactly one *applicable* increment gadget. The increment gadgets

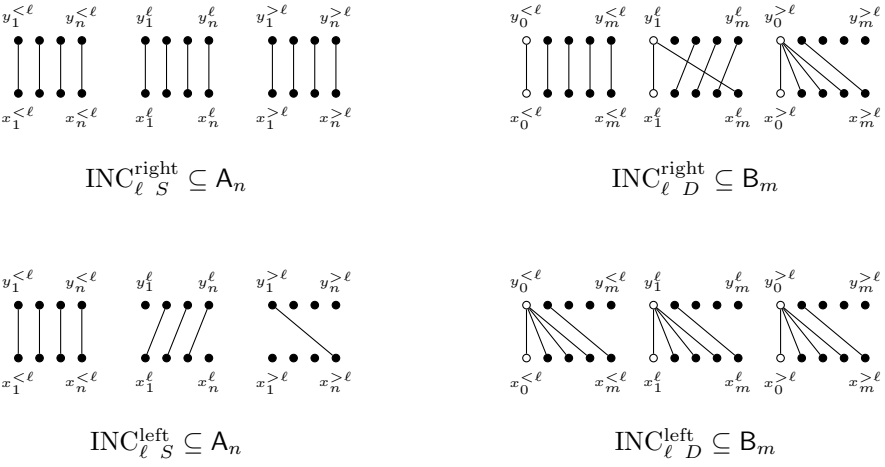

Fig. 4. The increment gadgets

are shown in Figure 4. All input vertices x^i_j have at most one output vertex $y^i_{j'}$ as neighbor. Furthermore, if the gadget is applicable to a valid configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$, then the unique neighbor of $x^i_{\mathfrak{a}(i)}$ is $y^i_{\mathfrak{a}^+(i)}$ and the unique neighbor of $x^i_{\mathfrak{b}(i)}$ is $y^i_{\mathfrak{b}^+(i)}$. This enables Spoiler to reach $\mathfrak{q}^+$ on the output from $\mathfrak{q}$ on the input by the following procedure. First, Spoiler places the remaining pebble on $y^1_{\mathfrak{a}^+(1)}$. Since this vertex is adjacent to $x^1_{\mathfrak{a}(1)}$, Duplicator has to answer with $y^1_{\mathfrak{b}^+(1)}$, the only vertex that is adjacent to $x^1_{\mathfrak{b}(1)}$. Afterwards, Spoiler picks up the pebble pair from $(x^1_{\mathfrak{a}(1)}, x^1_{\mathfrak{b}(1)})$. On the second block Spoiler proceeds the same way: he pebbles $y^2_{\mathfrak{a}^+(2)}$, forces the position $(y^2_{\mathfrak{a}^+(2)}, y^2_{\mathfrak{b}^+(2)})$ and picks up the pebbles from $(x^2_{\mathfrak{a}(2)}, x^2_{\mathfrak{b}(2)})$. By iterating this procedure Spoiler reaches $\mathfrak{q}^+$ on the output.

If Spoiler tries to move a configuration through one increment gadget that is *not* applicable, then Duplicator can answer with an invalid configuration on the output as follows. On the one hand, if the gadget is not applicable because some $\mathfrak{b}(i)$ does not have the specified value, then $x^i_{\mathfrak{b}(i)}$ is adjacent to y^i_0. On the other hand, if some $\mathfrak{a}(i)$ has the wrong value, then $x^i_{\mathfrak{a}(i)}$ is not adjacent to an output vertex. In both cases Duplicator can safely pebble y^i_0 if Spoiler queries some y^i_j and hence maintain an invalid output position. The next lemma summarizes the strategies on the increment gadget.

Lemma 5. *Let* $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ *be a configuration and* INC *an increment gadget.*

1. *If* INC *is applicable to* $\mathfrak{q}$*, then Spoiler can reach* $\mathfrak{q}^+$ *on the output from* $\mathfrak{q}$ *on the input.*
2. *If* INC *is applicable to* $\mathfrak{q}$*, then there is a winning strategy for Duplicator with boundary function* $h^x_{\mathfrak{q}}$ *on the input and* $h^y_{\mathfrak{q}^+}$ *on the output.*
3. *If* INC *is not applicable to* $\mathfrak{q}$*, then there is a winning strategy for Duplicator with boundary function* $h^x_{\mathfrak{q}}$ *on the input and* $h^y_{\mathfrak{q}_{\text{inv}}}$ *on the output for an invalid configuration* $\mathfrak{q}_{\text{inv}}$*.*

The *switch* is an extension of the "multiple input one-way switch" defined in [3] (which in turn is a generalization of [11]). The difference is that the old switch can only be used for the case $n = 1$. It requires some work to adjust the old switch to make it work for the more general setting. But since these modifications require a deeper inspection into this technical construct (and are not the main contribution of this paper), we refer to the full version of the paper [2] and use the switch as black box at this point.

We briefly explain the strategies on the switch and provide them in Lemma 6. As mentioned earlier, Spoiler can simply move a valid position from the input to the output of the switch (Lemma 6(i)). Duplicator has a winning strategy called *output strategy*, where any position is on the output and $h^x_{\mathfrak{0}}$ is on the input (Lemma 6(ii)). This ensures that Spoiler cannot move backwards to reach $\mathfrak{q}$ on the input from $\mathfrak{q}$ on the output. Hence, this strategy forces Spoiler to play through the switches in the intended direction (as indicated by arrows Figure 2). Furthermore, for every invalid $\mathfrak{q}_{\text{inv}}$ Duplicator has a winning strategy where

$h^x_{\mathfrak{q}_{\text{inv}}}$ is on the input and $h^y_{\mathbf{0}}$ is on the output (Lemma 6(iii)), which ensures that Spoiler cannot move invalid positions through the switch. This strategy is used by Duplicator whenever Spoiler plays on an increment gadget that is not applicable. By Lemma 5, Duplicator can force an invalid configuration on the output of that increment gadget and hence on the input of the subsequent switch.

To ensure that Spoiler picks up all pebbles when reaching $\mathfrak{q}$ on the output from $\mathfrak{q}$ on the input, Duplicator has a critical *input strategy* with $\mathfrak{q}$ on the input and $h^y_{\mathbf{0}}$ on the output (Lemma 6(iv)). The critical positions are either contained in an output strategy, where $\mathfrak{q}$ is on the output, or (for technical reasons) in a *restart strategy*. If Duplicator plays according to this input strategy, the only way for Spoiler to bring $\mathfrak{q}$ from the input to the output is to pebble an output critical position inside the switch (using all the pebbles) and force Duplicator to switch to the corresponding output strategy.

Lemma 6. *For every configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$, the following statements hold in the existential $(\mathsf{k}+1)$-pebble game on the switch:*

(i) If $\mathfrak{q}$ is valid, then Spoiler can reach $\mathfrak{q}$ on the output from $\mathfrak{q}$ on the input.
(ii) Duplicator has a winning strategy $\mathcal{H}^{out}_{\mathfrak{q}}$ with boundary function $h^x_{\mathbf{0}} \cup h^y_{\mathfrak{q}}$.
(iii) If $\mathfrak{q}$ is invalid, then Duplicator has a winning strategy $\mathcal{H}^{restart}_{\mathfrak{q}}$ with boundary function $h^x_{\mathfrak{q}} \cup h^y_{\mathbf{0}}$.
(iv) If $\mathfrak{q}$ is valid, then Duplicator has a critical strategy $\mathcal{H}^{in}_{\mathfrak{q}}$ with boundary function $h^x_{\mathfrak{q}} \cup h^y_{\mathbf{0}}$ and sets of restart critical positions $\mathcal{C}^{restart\text{-}crit}_{\mathfrak{q},t}$ (for $t \in [\mathsf{k}]$) and output critical positions $\mathcal{C}^{out\text{-}crit}_{\mathfrak{q}}$ such that:
  (a) $\operatorname{crit}(\mathcal{H}^{in}_{\mathfrak{q}}) = \bigcup_{t\in[\mathsf{k}]} \mathcal{C}^{restart\text{-}crit}_{\mathfrak{q},t} \cup \mathcal{C}^{out\text{-}crit}_{\mathfrak{q}}$,
  (b) $\mathcal{C}^{restart\text{-}crit}_{\mathfrak{q},t} \subseteq \mathcal{H}^{restart}_{(\mathfrak{a},\mathfrak{b},\{t\})}$ and
  (c) $\mathcal{C}^{out\text{-}crit}_{\mathfrak{q}} \subseteq \mathcal{H}^{out}_{\mathfrak{q}}$.

At the beginning of the game we want that Spoiler can reach the start configuration $\alpha^{-1}(0)$ on x, which is the pebble position $\{(\mathsf{x}^1_1, \mathsf{x}^1_1), \ldots, (\mathsf{x}^k_1, \mathsf{x}^k_1)\}$. To ensure this, we use the *initialization gadget* and identify its output vertices y^i_j with the block of x^i_j vertices. As for the switch, this gadget is an extension of the initialization gadget presented in [3] and we use it as a black box here. The strategies on this gadget are provided in Lemma 7, the proof of Lemma 7 is given in the full version of the paper [2]. The main property of the gadget is that Spoiler can reach the start position $\mathfrak{q}$ at the boundary (i) and Duplicator has a corresponding counter strategy (ii) in this situation. Furthermore, if an arbitrary position occurs at the boundary during the game, Duplicator has a strategy to survive (iii). This is only a critical strategy, but Duplicator can switch to the initial strategy (hence "restart" the game) if Spoiler moves to one of the critical positions.

Lemma 7. *Let $\mathfrak{q} = \alpha^{-1}(0)$. The following holds in the existential $(\mathsf{k}+1)$-pebble game on* INIT*:*

(i) Spoiler can reach $\mathfrak{q}$ on the output.

(ii) There is a winning strategy $\mathcal{I}^{init}$ for Duplicator with boundary function $h^y_{\mathfrak{q}}$.
(iii) For every (valid or invalid) configuration $\mathfrak{q}'$ there is a critical strategy $\mathcal{I}^{init}_{\mathfrak{q}'}$ with boundary function $h^y_{\mathfrak{q}'}$ and $\mathrm{crit}(\mathcal{I}^{init}_{\mathfrak{q}'}) \subseteq \mathcal{I}^{init}$.

3.3 Proof of Theorem 1

The size of the vertex set in every gadget is linear in n on Spoiler's side and linear in m on Duplicator's side. Since the overall construction uses a constant number of gadgets it follows that $|V(\mathsf{A}_n)| = O(n)$ and $|V(\mathsf{B}_m)| = O(m)$. To prove the lower bound on the number of rounds Spoiler needs to win the existential $(\mathsf{k}+1)$-pebble game we provide a sequence of critical strategies in Lemma 8 satisfying the properties stated in Lemma 4. For a critical strategy $\mathcal{S}$ we let $\widehat{\mathcal{S}} := \mathcal{S} \setminus \mathrm{crit}(\mathcal{S})$.

Lemma 8. *Spoiler has a winning strategy in the existential $(\mathsf{k}+1)$-pebble game on A_n and B_m. Furthermore, there is a sequence of critical strategies for Duplicator $\mathcal{G}^{\mathrm{start}}, \mathcal{F}_1, \mathcal{G}_1, \mathcal{F}_2, \mathcal{G}_2, \ldots, \mathcal{G}_{n^{\mathsf{k}}m^{\mathsf{k}}-2}, \mathcal{F}_{n^{\mathsf{k}}m^{\mathsf{k}}-1}$ such that*

$$\begin{aligned} \mathrm{crit}(\mathcal{G}^{\mathrm{start}}) &\subseteq \widehat{\mathcal{F}}_1, \\ \mathrm{crit}(\mathcal{G}_i) &\subseteq \widehat{\mathcal{F}}_{i+1} \cup \widehat{\mathcal{G}}^{\mathrm{start}}, && 1 \le i \le n^{\mathsf{k}}m^{\mathsf{k}} - 2, \\ \mathrm{crit}(\mathcal{F}_i) &\subseteq \widehat{\mathcal{G}}_i \cup \widehat{\mathcal{G}}^{\mathrm{start}}, && 1 \le i \le n^{\mathsf{k}}m^{\mathsf{k}} - 2. \end{aligned}$$

Proof (Proof of Theorem 1). For $k = 2$ the theorem follows from [4]. For $k \ge 3$ consider the structures A_n and B_m (for $\mathsf{k} = k - 1$) defined above. By Lemma 8 Spoiler wins the existential k-pebble game on A_n and B_m. Furthermore, it follows via Lemma 4 that Spoiler needs at least $\Omega(n^{k-1}m^{k-1})$ rounds to win the game. To get structures with exactly n and m vertices we take the largest n', m' such that $|V(\mathsf{A}_{n'})| \le n$, $|V(\mathsf{B}_{m'})| \le m$ and fill up the structures with an appropriate number of isolated vertices. □

Proof (Proof of Lemma 8). To show that Spoiler has a winning strategy it suffices to prove the following three statements:

(1) Spoiler can reach the position $\alpha^{-1}(0)$ on x from $\emptyset$,
(2) Spoiler can reach $\alpha^{-1}(i+1)$ on x from $\alpha^{-1}(i)$ on x (for $i < n^{\mathsf{k}}m^{\mathsf{k}} - 1$) and
(3) Spoiler wins from $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}} - 1)$ on x.

Assertion (1) follows from Lemma 7 and (3) is ensured by the winning gadget. For (2), Spoiler starts with the position $\mathfrak{q} = \alpha^{-1}(i)$ on x. Since $i < n^{\mathsf{k}}m^{\mathsf{k}} - 1$ there is exactly one increment gadget applicable to $\mathfrak{q}$. Spoiler uses Lemma 5 to reach $\mathfrak{q}^+ = \alpha^{-1}(i+1)$ on the output of that gadget. By applying Lemma 6.(i) twice, Spoiler can pebble $\mathfrak{q}^+$ through the two switches to the x vertices.

To define the sequence of global critical strategies we combine the partial critical strategies on the gadgets using the $\uplus$-operator. There are three types of strategies: $\mathcal{G}^{\mathrm{start}}$, $\mathcal{F}_i$ and $\mathcal{G}_i$. To define $\mathcal{G}_i$ we let $\mathfrak{q} = \alpha^{-1}(i)$. Duplicator plays according to $h^{\mathsf{x}}_{\mathfrak{q}}$ on x and according to $h^{\mathsf{y}}_{\mathbf{0}}$ on y. She plays according to this

strategy in the case when Spoiler reaches "$\mathfrak{q}$ on x". The critical strategy $\mathcal{G}_i$ is the combination of the following (pairwise connectable) strategies on the gadgets:

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathfrak{q}}$ on the initialization gadget (Lemma 7).
- The winning strategy with boundary $h^x_{\mathfrak{q}}$ and $h^y_{\mathfrak{q}^+}$ on the increment gadget applicable to $\mathfrak{q}$ (Lemma 5).
- The critical input strategy $\mathcal{H}^{\text{in}}_{\mathfrak{q}^+}$ on the switch following the applicable increment gadget (Lemma 6).
- The winning strategy with boundary $h^x_{\mathfrak{q}}$ and $h^y_{\mathfrak{q}_{\text{inv}}}$ on the other increment gadgets not applicable to $\mathfrak{q}$ (Lemma 5).
- The winning strategy $\mathcal{H}^{\text{restart}}_{\mathfrak{q}_{\text{inv}}}$ on the switches following the inapplicable increment gadgets (Lemma 6). Here, $\mathfrak{q}_{\text{inv}}$ is the invalid configuration on the output of the corresponding increment gadget.
- The output winning strategy $\mathcal{H}^{\text{out}}_{\mathfrak{q}}$ on the single switch (Lemma 6).

If in the above setting Spoiler increments $\mathfrak{q}$ through the applicable increment gadget and moves $\mathfrak{q}^+ = \alpha^{-1}(i+1)$ through the subsequent switch, then Duplicator switches to the strategy $\mathcal{F}_{i+1}$. To define $\mathcal{F}_i$ we fix $\mathfrak{q} = \alpha^{-1}(i)$. In this strategy, Duplicator plays according to $h^{\mathsf{x}}_{\mathbf{0}}$ on x and according to $h^{\mathsf{y}}_{\mathfrak{q}}$ on y. This critical strategy is the combination of the following strategies on the gadgets.

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathbf{0}}$ on the initialization gadget.
- The winning strategy with boundary $h^x_{\mathbf{0}}$ and $h^y_{\mathbf{0}}$ on the increment gadgets.
- The output strategy $\mathcal{H}^{\text{out}}_{\mathfrak{q}}$ on the switches following the increment gadgets.
- The critical input strategy $\mathcal{H}^{\text{in}}_{\mathfrak{q}}$ on the single switch.

The critical positions in the strategies $\mathcal{G}_i$ and $\mathcal{F}_i$ are inside the switches and the initialization gadget. Recall that by Lemma 6.(iv) the critical positions on the switch can be divided into restart critical positions and output critical positions. Furthermore, all output critical positions of $\mathcal{G}_i$, which are inside the switch following the applicable increment gadget, are contained as non-critical positions in $\mathcal{F}_{i+1}$. All output critical position in $\mathcal{F}_i$, which are inside the single switch, are contained as non-critical positions in $\mathcal{G}_i$. Now we define $\mathcal{G}^{\text{start}}$, which contains all other critical positions of $\mathcal{G}_i$ and $\mathcal{F}_i$. The critical strategy $\mathcal{G}^{\text{start}}$ is the union of several other global strategies. The first one is $\mathcal{G}^{\text{init}}$, which is defined as $\mathcal{G}_0$ except that it contains the winning strategy $\mathcal{I}^{\text{init}}$ on the initialization gadget. Thus, by Lemma 7, it contains every critical position on the initialization gadget as non-critical position. Note that the output critical positions of $\mathcal{G}^{\text{init}}$ are contained as non-critical positions in $\mathcal{F}_1$. Since $\mathcal{G}^{\text{init}}$ handles the critical positions on the initialization gadget and we discussed the output critical positions on the switches, it remains to consider the restart critical positions of the strategies. For this we construct a strategy $\mathcal{G}^{\text{restart}}_i$ to handle the restart critical positions of $\mathcal{G}_i$ (for $i \geq 1$) and of $\mathcal{G}^{\text{init}}$ (for $i = 0$). Furthermore, we define for every $i \geq 1$ a strategy $\mathcal{F}^{\text{restart}}_i$ to handle the restart critical positions of $\mathcal{F}_i$.

For $0 \leq i \leq n^{\mathsf{k}} m^{\mathsf{k}} - 2$ and $t \in [\mathsf{k}]$ we let $\mathfrak{q} = \alpha^{-1}(i) = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ and $\mathfrak{q}_t$ be the invalid configuration $(\mathfrak{a}, \mathfrak{b}, \{t\})$. The global strategy $\mathcal{G}^{\text{restart}}_{i,t}$ is the combination of the following strategies on the gadgets.

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathfrak{q}_t}$ on the initialization gadget.
- The winning strategy with boundary $h^x_{\mathfrak{q}_t}$ and $h^y_{\mathfrak{q}_{\text{inv}}}$ on the increment gadgets. Note that, since $\mathfrak{q}_t$ is invalid, no increment gadget is applicable to $\mathfrak{q}_t$.
- The winning strategy $\mathcal{H}^{\text{restart}}_{\mathfrak{q}_{\text{inv}}}$ on the switches following the increment gadgets. Again, $\mathfrak{q}_{\text{inv}}$ is the invalid configuration at the output of the preceding increment gadget.
- The output winning strategy $\mathcal{H}^{\text{out}}_{\mathfrak{q}_t}$ on the single switch.

Finally, we let $\mathcal{G}^{\text{restart}}_i := \bigcup_{i\in[\mathsf{k}]} \mathcal{G}^{\text{restart}}_{i,t}$. Note that by Lemma 6.(iv) every restart critical position of $\mathcal{G}_i$ is contained in $\mathcal{G}^{\text{restart}}_i$ and every restart critical position of $\mathcal{G}^{\text{init}}$ is contained in $\mathcal{G}^{\text{restart}}_0$. Now we define for $1 \le i \le n^{\mathsf{k}}m^{\mathsf{k}} - 2$, $t \in [\mathsf{k}]$, $\mathfrak{q} = \alpha^{-1}(i) = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ and $\mathfrak{q}_t := (\mathfrak{a}, \mathfrak{b}, \{t\})$ the strategy $\mathcal{F}^{\text{restart}}_{i,t}$ analogously. It consists of the following partial strategies.

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathbf{0}}$ on the initialization gadget.
- The winning strategy with boundary $h^x_{\mathbf{0}}$ and $h^y_{\mathbf{0}}$ on the increment gadgets.
- The winning strategy $\mathcal{H}^{\text{restart}}_{\mathbf{0}}$ on the switches after the increment gadgets.
- The winning strategy $\mathcal{H}^{\text{restart}}_{\mathfrak{q}_t}$ on the single switch.

In the end we let $\mathcal{F}^{\text{restart}}_i$ be the union of all $\mathcal{F}^{\text{restart}}_{i,t}$. Note that every restart critical position of $\mathcal{F}_i$ is contained as non-critical position in $\mathcal{F}^{\text{restart}}_i$. Finally, let

$$\mathcal{G}^{\text{start}} := \mathcal{G}^{\text{init}} \cup \bigcup_{0\le i\le n^{\mathsf{k}}m^{\mathsf{k}}-2} \mathcal{G}^{\text{restart}}_i \cup \bigcup_{1\le i\le n^{\mathsf{k}}m^{\mathsf{k}}-2} \mathcal{F}^{\text{restart}}_i.$$

To conclude the proof note that the critical positions of $\mathcal{G}^{\text{restart}}_i$ and $\mathcal{F}^{\text{restart}}_i$ are inside the initialization gadget and hence contained in $\widehat{\mathcal{G}}^{\text{init}}$. Thus they are not critical positions of $\mathcal{G}^{\text{start}}$. Hence, $\text{crit}(\mathcal{G}^{\text{start}}) = \text{crit}(\mathcal{G}^{\text{init}}) \subseteq \widehat{\mathcal{F}}_1$. □

4 Conclusion

We have proven an optimal lower bound of $\Omega(n^{k-1}d^{k-1})$ on the number of nested propagation steps in the k-consistency procedure on constraint networks with n variables and domain size d. It follows that every parallel propagation algorithm has to perform at least $\Omega(n^{k-1}d^{k-1})$ sequential steps. Using $(n+d)^{O(k)}$ processors (one for every instance of the inference rule), k-consistency can be computed in $O(n^{k-1}d^{k-1})$ parallel time, which is optimal for propagation algorithms. In addition, the best sequential algorithm runs in $O(n^k d^k)$. The overhead compared to the parallel approach is mainly caused by the time needed to search for the next inconsistent assignment that might be propagated – and this seems to be the only task that can be parallelized.

Although we have proven an optimal lower bound in the general setting, it might be interesting to investigate the propagation depth of k-consistency on restricted classes of structures. Especially, if in such cases the propagation depth is bounded by $O(\log(n+d))$, we know that k-consistency is in NC and hence parallelizable.

References

1. Atserias, A., Kolaitis, P.G., Vardi, M.Y.: Constraint propagation as a proof system. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 77–91. Springer, Heidelberg (2004)
2. Berkholz, C.: The Propagation Depth of Local Consistency. ArXiv e-prints (2014), http://arxiv.org/abs/1406.4679
3. Berkholz, C.: Lower bounds for existential pebble games and k-consistency tests. Logical Methods in Computer Science 9(4) (2013), http://arxiv.org/abs/1205.0679
4. Berkholz, C., Verbitsky, O.: On the speed of constraint propagation and the time complexity of arc consistency testing. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 159–170. Springer, Heidelberg (2013)
5. Cooper, M.C.: An optimal k-consistency algorithm. Artificial Intelligence 41(1), 89–95 (1989)
6. Dechter, R., Pearl, J.: A problem simplification approach that generates heuristics for constraint-satisfaction problems. Tech. rep., Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles (1985)
7. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. SIAM Journal on Computing 28(1), 57–104 (1998)
8. Freuder, E.C.: Synthesizing constraint expressions. Commun. ACM 21, 958–966 (1978)
9. Gaspers, S., Szeider, S.: The parameterized complexity of local consistency. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 302–316. Springer, Heidelberg (2011)
10. Kasif, S.: On the parallel complexity of discrete relaxation in constraint satisfaction networks. Artificial Intelligence 45(3), 275–286 (1990)
11. Kolaitis, P.G., Panttaja, J.: On the complexity of existential pebble games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 314–329. Springer, Heidelberg (2003)
12. Kolaitis, P.G., Vardi, M.Y.: On the expressive power of datalog: Tools and a case study. J. Comput. Syst. Sci. 51(1), 110–134 (1995)
13. Kolaitis, P.G., Vardi, M.Y.: A game-theoretic approach to constraint satisfaction. In: Proc AAAI/IAAI 2000, pp. 175–181 (2000)
14. Ladkin, P.B., Maddux, R.D.: On binary constraint problems. J. ACM 41(3), 435–469 (1994), http://doi.acm.org/10.1145/176584.176585
15. Samal, A., Henderson, T.: Parallel consistent labeling algorithms. International Journal of Parallel Programming 16, 341–364 (1987)
16. Susswein, S., Henderson, T., Zachary, J., Hansen, C., Hinker, P., Marsden, G.: Parallel path consistency. International Journal of Parallel Programming 20(6), 453–473 (1991), http://dx.doi.org/10.1007/BF01547895