On the Satisfiability of Smooth Grid CSPs

Vasily Alferov
National Research University Higher School of Economics, Saint Petersburg, Russia

Mateus de Oliveira Oliveira
University of Bergen, Norway

Abstract

Many important NP-hard problems, arising in a wide variety of contexts, can be reduced straightforwardly to the satisfiability problem for CSPs whose underlying graph is a grid. In this work, we push forward the study of grid CSPs by analyzing, from an experimental perspective, a symbolic parameter called smoothness.

More specifically, we implement an algorithm that provably works in polynomial time on grids of polynomial smoothness. Subsequently, we compare our algorithm with standard combinatorial optimization techniques, such as SAT-solving and integer linear programming (ILP). For this comparison, we use a class of grid-CSPs encoding the pigeonhole principle. We demonstrate, empirically, that these CSPs have polynomial smoothness and that our algorithm terminates in polynomial time.

On the other hand, as strong evidence that the grid-like encoding is not destroying the essence of the pigeonhole principle, we show that the standard propositional translation of pigeonhole CSPs remains hard for state-of-the-art SAT solvers, such as minisat and glucose, and even to state-of-the-art integer linear-programming solvers, such as Coin-OR CBC.

2012 ACM Subject Classification Theory of computation → Discrete optimization

Keywords and phrases Grid CSPs, Smoothness, SAT Solving, Linear Programming

Digital Object Identifier 10.4230/LIPIcs.SEA.2022.18

Supplementary Material *Software*: `https://github.com/AutoProving/GridCSPs`
archived at `swh:1:dir:04dab414f88c68d273bc136840c6286e26e0d0b7`

Funding *Mateus de Oliveira Oliveira*: Research Council of Norway, Grant Numbers 288761 and 326537.

1 Introduction

In this work, we consider constraint satisfaction problems (CSPs) where variables are arranged on an $m \times n$-grid, and where the domain of each variable is the set $\{1, \ldots, k\}$. Constraints are local, in the sense that they can only relate pairs of variables that correspond edges of the grid. In our work, these CSPs are called (m, n, k)-grid CSPs, or simply as grid CSPs when the parameters m, n, k are not relevant.

Grid CSPs have a wide variety of applications, ranging from board games to the simulation of Turing machines running for a given number of steps. From a complexity-theoretic perspective, the problem of determining whether a given grid CSP is satisfiable can be solved in polynomial time for $k \leq 2$ by a straightforward reduction to 2-SAT. On the other hand, for $k \geq 3$ the problem becomes NP-complete. Indeed, several natural NP-complete problems reduce straightforwardly to the satisfiability problem for grid CSPs with constant-size domains, including problems arising in the context of pattern recognition [20, 22], image processing [5, 20], tiling systems [15, 21], formal language theory [9, 12, 16, 20], and many others. Since the satisfiability problem for grid-like CSPs is NP-complete, and can be used to model many classes of interesting problems, the identification of subclasses of grid CSPs that can be solved efficiently is a fundamental quest.

We approach this quest by leveraging on the notion of *smoothness*, a complexity measure defined in [6] in the context of the picture satisfiability problem. In particular, we consider a similar notion of smoothness for grid-CSPs and prove analytically that the satisfiability problem for polynomially-smooth instances is solvable in polynomial time. From an experimental perspective, we implement our algorithm and evaluate the performance of our solver on a class of grid-CSPs encoding the pigeonhole-principle, which essentially states that there is no bijection between a set of $n+1$ and a set of n elements. In particular, we confirm experimentally that grid-CSPs encoding the pigeonhole principle has polynomial smoothness. In particular, our algorithm was able to solve both positive instances (that require mapping n pigeons to n holes), and negative instances (that require mapping $n+1$ pigeons to n holes) in time $O(n^3)$.

In an influential work, Haken showed that a family of propositional formulas H_m encoding the pigeonhole principle requires resolution refutations of exponential size [3, 10]. Superpolynomial lower bounds for this principle have been shown for a variety of proof systems, including constant-depth Frege proof systems [13, 14, 19]. It turns out that the formulas arising from Haken's encoding of the pigeonhole principle are also hard in practice to state-of-the-art SAT-solvers based on the technique of conflict-driven clause-learning (CDCL) [7, 17]. Indeed, it has been shown that certain variants of CDCL-based SAT solvers, such as those introduced in [7, 17], are equivalent in power to resolution-based proof systems [2, 4, 11, 18].

We give strong evidence that our encoding of the pigeonhole principle as grid CSPs preserves its inherent difficulty by showing empirically that the most direct propositional translation of our grid CSPs into SAT instances remain hard for two of the most well known SAT solvers based on the CDCL paradigm (minisat [7] and glucose [1]). Going further, we show experimentally that the most straightforward integer linear-programming formulation of our grid CSPs is hard to the state-of-the-art integer linear programming solver Coin-OR CBC solver [8].

Our theoretical and experimental results give strong evidence that the smoothness of grid-CSPs is a parameter that may be valuable in the study of combinatorial problems involving constraints based on the pigeonhole principle.

2 Preliminaries

We denote by $\mathbb{N} \doteq \{0, 1, \ldots\}$ the set of natural numbers (including zero), and by $\mathbb{N}_+ \doteq \mathbb{N} \setminus \{0\}$ the set of positive natural numbers. For each $c \in \mathbb{N}_+$, we let $[c] \doteq \{1, 2, \ldots, c\}$ and $[\![c]\!] \doteq \{0, 1, \ldots, c-1\}$.

Let Σ be an alphabet and $w \in \mathbb{N}_+$. A (Σ, w)*-layer* is a tuple $B \doteq (\ell, r, T, I, F, \iota, \phi)$, where $\ell \subseteq [\![w]\!]$ is a set of *left states*, $r \subseteq [\![w]\!]$ is a set of *right states*, $T \subseteq \ell \times \Sigma \times r$ is a set of *transitions*, $I \subseteq \ell$ is a set of *initial states*, $F \subseteq r$ is a set of *final states* and $\iota, \phi \in \{0, 1\}$ are Boolean flags satisfying the two following conditions:

1. if $\iota = 0$ then $I = \emptyset$;
2. if $\phi = 0$ then $F = \emptyset$.

In what follows, we may write $\ell(B)$, $r(B)$, $T(B)$, $I(B)$, $F(B)$, $\iota(B)$ and $\phi(B)$ to refer to the sets ℓ, r, T, I and F and to the Boolean flags ι and ϕ, respectively.

We let $\mathcal{B}(\Sigma, w)$ denote the set of all (Σ, w)-layers. Note that, $\mathcal{B}(\Sigma, w)$ is non-empty and has at most $2^{\mathcal{O}(|\Sigma| \cdot w^2)}$ elements. Let $n \in \mathbb{N}_+$. A (Σ, w)*-ordered decision diagram* (or simply, (Σ, w)-*ODD*) of *length* n is a string $D \doteq B_1 \cdots B_n \in \mathcal{B}(\Sigma, w)^n$ of length n over the alphabet $\mathcal{B}(\Sigma, w)$ satisfying the following conditions:

1. for each $i \in [n-1]$, $\ell(B_{i+1}) = r(B_i)$;
2. $\iota(B_1) = 1$ and, for each $i \in \{2, \dots, n\}$, $\iota(B_i) = 0$;
3. $\phi(B_n) = 1$ and, for each $i \in [n-1]$, $\phi(B_i) = 0$.

We note that Condition 2 guarantees that only the first layer of an ODD is allowed to have initial states. Analogously, Condition 3 guarantees that only the last layer of an ODD is allowed to have final states.

The *size* of an ODD $D = B_1 \dots B_n$ is defined as $\mathsf{size}(D) = |\ell(B_1)| + \sum_{j=1}^{n} |r(B_j)|$. For each $n \in \mathbb{N}_+$, we denote by $\mathcal{B}(\Sigma, w)^{\circ n}$ the set of all (Σ, w)-ODDs of length n. The *width* of an ODD $D = B_1 \cdots B_n \in \mathcal{B}(\Sigma, w)^{\circ n}$ is defined as $\mathsf{w}(D) \doteq \max\{|\ell(B_1)|, \dots, |\ell(B_n)|, |r(B_n)|\}$. We remark that $\mathsf{w}(D) \le w$.

Let $D = B_1 \cdots B_n \in \mathcal{B}(\Sigma, w)^{\circ n}$ and $s = \sigma_1 \cdots \sigma_n \in \Sigma^n$. A *valid sequence* for s in D is a sequence of transitions $\langle (\mathfrak{p}_1, \sigma_1, \mathfrak{q}_1), \dots, (\mathfrak{p}_n, \sigma_n, \mathfrak{q}_n) \rangle$ such that $\mathfrak{p}_{i+1} = \mathfrak{q}_i$ for each $i \in [n-1]$, and $(\mathfrak{p}_i, \sigma_i, \mathfrak{q}_i) \in T(B_i)$ for each $i \in [n]$. Such a valid sequence is called *accepting* for s if, additionally, $\mathfrak{p}_1 \in I(B_1)$ and $\mathfrak{q}_n \in F(B_n)$. We say that D *accepts* s if there exists an accepting sequence for s in D. The language of D, denoted by $\mathcal{L}(D)$, is defined as the set of all strings accepted by D, i.e. $\mathcal{L}(D) \doteq \{s \in \Sigma^n : s \text{ is accepted by } D\}$.

A (Σ, w)-layer B is called *deterministic* if the following conditions are satisfied:

1. for each $\mathfrak{p} \in \ell(B)$ and each $\sigma \in \Sigma$, there exists at most one right state $\mathfrak{q} \in r(B)$ such that $(\mathfrak{p}, \sigma, \mathfrak{q}) \in T(B)$;
2. if $\iota(B) = 1$, then $I(B) = \ell(B)$ and $|\ell(B)| = 1$.

On the other hand, a (Σ, w)-layer B is called *complete* if, for each $\mathfrak{p} \in \ell(B)$ and each $\sigma \in \Sigma$, there exists at least one right state $\mathfrak{q} \in r(B)$ such that $(\mathfrak{p}, \sigma, \mathfrak{q}) \in T(B)$. We let $\widehat{\mathcal{B}}(\Sigma, w)$ be the subset of $\mathcal{B}(\Sigma, w)$ comprising all deterministic, complete (Σ, w)-layers.

An ODD $D = B_1 \cdots B_n \in \mathcal{B}(\Sigma, w)^{\circ n}$ is called *deterministic* (*complete*, resp.) if, for each $i \in [n]$, B_i is a deterministic (complete, resp.) layer. We remark that, if D is deterministic, then there exists at most one valid sequence in D for each string in Σ^n. On the other hand, if D is complete, then there exists at least one valid sequence in D for each string in Σ^n. For each $n \in \mathbb{N}_+$, we let $\widehat{\mathcal{B}}(\Sigma, w)^{\circ n}$ be the subset of $\mathcal{B}(\Sigma, w)^{\circ n}$ comprising all deterministic, complete (Σ, w)-ODDs of length n.

We say that an ODD $D = B_1 \dots B_n$ in $\mathcal{B}([k], w)^{\circ n}$ has non-determisitic degree d if all, but at most one, layers of D are deterministic. Additionally, if there is a j such that B_j is not deterministic, then for each state $q \in \ell(B_j)$, and each symbol $\sigma \in [k]$, there is at most d states $q' \in r(B_j)$ such that $(q, \sigma, q') \in T(B_j)$.

The following lemma can be proved by applying the standard power set construction followed by a standard minimization algorithm for ODDs, and by observing that only subsets of size at most d belonging to each frontier are relevant.

▶ **Lemma 1.** *Let D be an ODD in $\mathcal{B}([k], w)^{\circ n}$ be an ODD of nondeterministic degree d. Then one can construct in time $n \cdot w^{O(d)}$ a deterministic ODD D' with minimum number of states with the property that $\mathcal{L}(D') = \mathcal{L}(D)$.*

We will also need the following lemma stating that ODD representatives for synchronized products of languages accepted by two given ODDs can be computed efficiently.

▶ **Lemma 2** (Synchronized Product of Automata)**.** *Let D and D' be ODDs in $\mathcal{B}([k], k)^{\circ n}$. Let $V = \{V_1, \dots, V_n\}$ be a set where for each $i \in [n]$, $V_i \subseteq [k] \times [k]$. Then one can construct in time $O(k^2 \cdot n)$ an ODD $D \otimes_V D'$ accepting the following language over $[k] \times [k]$.*

$$\mathcal{L}(D \otimes_V D') = \{(\sigma_1, \sigma_1') \dots (\sigma_n, \sigma_n') \mid \sigma_1 \dots \sigma_n \in \mathcal{L}(D),\ \sigma_1' \dots \sigma_n' \in \mathcal{L}(D'), (\sigma_i, \sigma_i') \in V_i\}.$$

3 Smooth Grid Constraint Satisfaction Problems

Let m, n and k be positive integers. An (m, n, k)-*grid CSP* is specified by a pair $(\mathcal{V}, \mathcal{H})$ where $\mathcal{V} = \{V_{i,j}\}_{i\in[m-1],j\in[n]}$ is a collection of sets $V_{i,j} \subseteq [k] \times [k]$ called *local vertical constraints*, and $\mathcal{H} = \{H_{i,j}\}_{i\in[m],j\in[n-1]}$ is a collection of sets $H_{i,j} \subseteq [k] \times [k]$ called *local horizontal constraints*. A *solution* for $(\mathcal{V}, \mathcal{H})$ is an $m \times n$ matrix $M \in [k]^{m\times n}$ which satisfies all local vertical and horizontal constraints. More precisely, such a solution M satisfies the following conditions.

1. For each $(i, j) \in [m - 1] \times [n]$, the pair $(M_{i,j}, M_{i+1,j})$ belongs to $V_{i,j}$.
2. For each $(i, j) \in [m] \times [n - 1]$, the pair $(M_{i,j}, M_{i,j+1})$ belongs to $H_{i,j}$.

Let m and n be positive integers. We endow the set $[m] \times [n] = \{(i, j) \mid i \in [m], j \in [n]\}$ with a *lexicographic order* $<$ that sets $(i, j) < (i', j')$ if either $i < i'$, or $i = i'$ and $j < j'$. We write $(i, j) \le (i', j')$ to denote that $(i, j) = (i', j')$ or $(i, j) < (i', j')$. For each $(i, j) \in [m] \times [n]$, we let

$$S(m, n, i, j) = \{(i', j') \in [m] \times [n] \ : \ (i', j') \le (i, j)\}$$

be the set of all positions in $[m] \times [n]$ that are (lexicographically) smaller than or equal to (i, j). Given (m, n, k)-grid CSP $(\mathcal{V}, \mathcal{H})$, we say that a function $M : S(m, n, i, j) \to [k]$ is an (i, j)-partial $(\mathcal{V}, \mathcal{H})$-solution if the following conditions are satisfied.

1. $(M_{i',j'}, M_{i'+1,j'}) \in V_{i',j'}$ for each (i', j'), $(i' + 1, j')$ in $S(m, n, i, j)$.
2. $(M_{i',j'}, M_{i',j'+1}) \in H_{i',j'}$ for each (i', j'), $(i', j' + 1)$ in $S(m, n, i, j)$.

Note that for simplicity, we write $M_{i,j}$ in place of $M(i, j)$ to designate an entry of M. Intuitively, an (i, j)-partial $(\mathcal{V}, \mathcal{H})$-solution for N is a function that colors the positions of N up to the entry (i, j) with elements from Σ in such a way that the vertical and horizontal constraints imposed by $\mathcal{V}$ and $\mathcal{H}$ respectively are respected. If (i, j_1) and (i, j_2) are positions in $S(m, n, i, j)$ with $j_1 < j_2$, then we let $M_{i,[j_1,j_2]} = M_{i,j_1}...M_{i,j_2}$ be the string over $[k]$ formed by all entries at the i-th row of M between positions j_1 and j_2. Now let $(i, j) \in S(m, n, i, j)$ with $(i, j) \ge (1, n)$. The (i, j)-boundary of M is defined as follows.

$$\partial_{i,j}(M) = \begin{cases} M_{i,[1,n]} & \text{if } j = n. \\ M_{i,[1,j]} \cdot M_{i-1,[j+1,n]} & \text{if } j < n. \end{cases} \tag{1}$$

In other words, if $j = n$, then $\partial(M)$ is the string consisting of all entries in the i-th row of M. On the other hand, if $j < n$, then $\partial(M)$ is obtained by concatenating the string corresponding to the first j entries of row i with the last $(n - j)$ entries of row $(i - 1)$. The notion of boundary of a partial solution is illustrated in Figure 1.

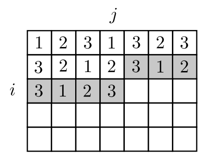


Figure 1 An (i, j)-partial solution M where $i = 3$ and $j = 4$. The grey entries form the boundary of M. Therefore $\partial_{i,j}(M) = 3123312$.

The (i, j)-feasibility boundary of $(\mathcal{V}, \mathcal{H})$, denoted by $\partial_{i,j}(\mathcal{V}, \mathcal{H})$, is defined as follows.

$$\partial_{i,j}(\mathcal{V}, \mathcal{H}) = \{\partial_{i,j}(M) \mid M \text{ is an } (i, j)\text{-partial } (\mathcal{V}, \mathcal{H})\text{-solution}\}. \tag{2}$$

Note that a $(\mathcal{V},\mathcal{H})$-solution exists if and only if $\partial_{m,n}(\mathcal{V},\mathcal{H})$ is non-empty.

▶ **Definition 3.** *We say that an (m,n,k)-grid CSP $(\mathcal{V},\mathcal{H})$ is s-smooth if for each $i \in [m]$ and each $j \in [n]$, there is a deterministic ODD D of size at most s such that $\mathcal{L}(D) = \partial_{i,j}(\mathcal{V},\mathcal{H})$.*

In [6] a similar notion of smoothness has been defined in the context of the picture satisfiability problem. The crucial difference is that our grid CSPs are a much more general combinatorial object, since the vertical constraints $V_{i,j}$ and the horizontal constraints $H_{i,j}$ may depend on the position (i,j), whereas in the context of pictures, all vertical (horizontal) constraints are required to be identical over the whole grid. The main result from [6] established that the satisfiability problem for pictures of polynomial smoothness is solvable in polynomial time. In this section, we generalize this result to the context of general grid CSPs.

The following lemma states that given an (m,n,k)-grid CSP $(\mathcal{V},\mathcal{H})$, one can construct and initial ODD $D(\mathcal{H},1) \in \mathcal{B}([k],k)^{\circ n}$ accepting precisely those strings in $[k]^n$ that satisfy all local horizontal constraints in the first row of $\mathcal{H}$.

▶ **Lemma 4.** *Let $(\mathcal{V},\mathcal{H})$ be an (m,n,k)-grid CSP. There is a deterministic ODD $D(\mathcal{H},1) \in \mathcal{B}([k],k)^{\circ n}$ such that*

$$\mathcal{L}(D(\mathcal{H},1)) = \{\sigma_1 \dots \sigma_n \in [k]^n \ : \ (\sigma_j,\sigma_{j+1}) \in H_{1,j} \text{ for each } j \in [n-1]\}$$

Proof. We let $D(\mathcal{H},1) = B_1 \dots B_n$ be the ODD in $\mathcal{B}([k],k)^{\circ n}$ where
1. $\ell(B_j) = r(B_j) = [k]$ for each $j \in [n]$,
2. $I(B_1) = \{1\}$ and $I(B_j) = \emptyset$ for each $j \in \{2,\dots,n\}$,
3. $F(B_n) = [k]$ and $F(B_j) = \emptyset$ for each $j \in \{1,\dots,n-1\}$,
4. $T(B_j) = \{(\sigma,\sigma',\sigma') \ : \ (\sigma,\sigma') \in H_{1,j}\}$.

Then it should be clear that a string $s = \sigma_1\sigma_2 \dots \sigma_n$ belongs to $\mathcal{L}(D(\mathcal{H},1))$ if and only if

$$\langle(\sigma_1,\sigma_2,\sigma_2)\dots(\sigma_{n-1},\sigma_n,\sigma_n)\rangle$$

is an accepting sequence for s in $D(\mathcal{H},1)$. By construction, this happens if and only if $(\sigma_j,\sigma_{j+1}) \in H_{1,j}$ for each $j \in [n-1]$. ◀

▶ **Lemma 5.** *Let k and w be a positive integers, $V \subseteq [k]\times[k]$ and D be a deterministic ODD in $\mathcal{B}([k],w)^{\circ n}$. Then one can construct in time $O(n \cdot w^k)$ an ODD $\mathsf{Up}(D,V,1)$ accepting the following language*

$$\mathcal{L}(\mathsf{Up}(D,V,1)) = \{aw \ : \ \exists b \in [k],\ bw \in \mathcal{L}(D),\ (b,a) \in V\}.$$

Proof. Let k and w be positive integers, B be a layer in $\mathcal{B}([k],w)$ and $V \subseteq [k]\times[k]$. We let $\mathfrak{l}(B,V)$ be the layer in $\mathcal{B}([k],k\cdot w)$ defined as follows.

1. $\ell(\mathfrak{l}(B,V)) = \ell(B)$ and $I(\mathfrak{l}(B,V)) = I(B)$.
2. $\iota(\mathfrak{l}(B,V)) = \iota(B)$ and $\phi(\mathfrak{l}(B,V)) = \phi(B)$.
3. $r(\mathfrak{l}(B,V)) = \{j\cdot w + j' \ : \ j \in r(B),\ j' \in [\![k]\!]\}$.
4. $F(\mathfrak{l}(B,V)) = \{j\cdot w + j' \ : \ j \in F(B), j' \in [\![k]\!]\}$.
5. $T(\mathfrak{l}(B,V)) = \{(i,b,j\cdot w + b) \ : \ \exists a \in [k], (i,a,j) \in T(B), (a,b) \in V\}$.

Additionally, we let $\mathfrak{r}(B,V)$ be the layer in $\mathcal{B}([k],k\cdot w)$ defined as follows.

1. $r(\mathfrak{r}(B,V)) = r(B)$ and $F(\mathfrak{r}(B,V)) = F(B)$.
2. $\iota(\mathfrak{r}(B,V)) = \iota(B)$ and $\phi(\mathfrak{r}(B,V)) = \phi(B)$.

3. $\ell(\mathfrak{r}(B,V)) = \{i \cdot w + i' \ : \ i \in \ell(B), \ i' \in [\![k]\!]\}$
4. $I(\mathfrak{r}(B,V)) = \emptyset$.
5. $T(\mathfrak{r}(B,V)) = \{(i \cdot w + i', a, j) \ : \ \exists a \in [k], (i,a,j) \in T(B)\}$.

Now, let $D = B_1 \dots B_n$ be a deterministic an ODD in $\mathcal{B}([k], w)^{\circ n}$ and let

$$D' = \mathfrak{l}(B_1, V)\mathfrak{r}(B_2, V)B_3 \dots B_n$$

be the ODD obtained from D by replacing B_1 with $\mathfrak{l}(B_1, V)$ and B_2 with $\mathfrak{r}(B_2, V)$. Then one can verify that

$$\mathcal{L}(D') = \{aw \ : \ \exists b \in [k], \ bw \in \mathcal{L}(D), \ (b,a) \in V\}.$$

Additionally, D' has non-deterministic degree at most k. Now we set the ODD $\mathsf{Up}(D, V, 1)$ as the minimum deterministic ODD such that $\mathcal{L}(\mathsf{Up}(D,V,1)) = \mathcal{L}(D')$. Since D' has non-deterministic degree at most k, we have that $\mathsf{Up}(D,V,1)$ can be constructed in time $O(n \cdot w^k)$. ◀

▶ **Lemma 6.** *Let k and w be a positive integers, $V, H \subseteq [k] \times [k]$, D be a deterministic ODD in $\mathcal{B}([k], w)^{\circ n}$, and $j \in [n-1]$. Then one can construct in time $O(n \cdot w^k)$ an ODD $\mathsf{Up}(D, V, H, j)$ accepting the following language*

$$\mathcal{L}(\mathsf{Up}(D,V,H,j)) = \{ubav \ : \ \exists c \in [k], ubcv \in \mathcal{L}(D), \ |ub| = j-1, \ (b,a) \in H, \ \textit{and} \ (c,a) \in V\}.$$

Proof. Let k and w be positive integers, B be a layer in $\mathcal{B}([k], w)$ and $V, H \subseteq [k] \times [k]$. We let $\mathfrak{l}(B, V, H)$ be the layer in $\mathcal{B}([k], k \cdot w)$ defined as follows.

1. $\ell(\mathfrak{l}(B,V,H)) = \ell(B)$ and $I(\mathfrak{l}(B,V,H)) = I(B)$.
2. $\iota(\mathfrak{l}(B,V,H)) = \iota(B)$ and $\phi(\mathfrak{l}(B,V,H)) = \phi(B)$.
3. $r(\mathfrak{l}(B,V,H)) = \{j \cdot w + j' \ : \ j \in r(B), \ j' \in [\![k]\!]\}$.
4. $F(\mathfrak{l}(B,V,H)) = \{j \cdot w + j' \ : \ j \in F(B), j' \in [\![k]\!]\}$.
5. $T(\mathfrak{l}(B,V,H)) = \{(i, b, j \cdot w + b) \ : \ \exists a \in [k], (i,a,j) \in T(B), (a,b) \in V\}$.

Additionally, we let $\mathfrak{r}(B, V, H)$ be the layer in $\mathcal{B}([k], k \cdot w)$ defined as follows.

1. $r(\mathfrak{r}(B,V,H)) = r(B)$ and $F(\mathfrak{r}(B,V,H)) = F(B)$.
2. $\iota(\mathfrak{r}(B,V,H)) = \iota(B)$ and $\phi(\mathfrak{r}(B,V,H)) = \phi(B)$.
3. $\ell(\mathfrak{r}(B,V,H)) = \{i \cdot w + i' \ : \ i \in \ell(B), \ i' \in [\![k]\!]\}$
4. $I(\mathfrak{r}(B,V,H)) = \emptyset$.
5. $T(\mathfrak{r}(B,V,H)) = \{(i \cdot w + i', a, j) \ : \ \exists a \in [k], (i,a,j) \in T(B)\}$.

Now, let $D = B_1 \dots B_n$ be a deterministic an ODD in $\mathcal{B}([k], w)^{\circ n}$ and $j \in [n-1]$. Let

$$D' = B_1 \dots \mathfrak{l}(B_j, V, H)\mathfrak{r}(B_{j+1}, V, H) \dots B_n$$

be the ODD obtained from D by replacing B_j with $\mathfrak{l}(B_j, V, H)$ and B_{j+1} with $\mathfrak{r}(B_{j+1}, V, H)$. Then one can verify that

$$\mathcal{L}(D') = \{ubav \ : \ \exists c \in [k], ubcv \in \mathcal{L}(D), \ |ub| = j-1, \ (b,a) \in H, \ \text{and} \ (c,a) \in V\}.$$

Additionally, D' has non-deterministic degree at most k. Now we set the ODD $\mathsf{Up}(D, V, H, j)$ as the minimum deterministic ODD such that $\mathcal{L}(\mathsf{Up}(D,V,H,j)) = \mathcal{L}(D')$. Since D' has non-deterministic degree at most k, we have that $\mathsf{Up}(D,V,H,j)$ can be constructed in time $O(n \cdot w^k)$. ◀

The following corollary is a consequence of Lemmas 5 and 6.

▶ **Corollary 7.** *Let $(\mathcal{V},\mathcal{H})$ be an (m,n,k)-grid CSP, and let D be an ODD in $\mathcal{B}([k],w)^{\circ n}$.*

1. *If $\mathcal{L}(D) = \partial_{i,n}(\mathcal{V},\mathcal{H})$ for some $i \in [m-1]$, then*

$$\mathcal{L}(\mathsf{Up}(D, V_{i+1,1}, 1)) = \partial_{i+1,1}(\mathcal{V},\mathcal{H}).$$

2. *If $\mathcal{L}(D) = \partial_{i,j}(\mathcal{V},\mathcal{H})$ for some $i \in [m]$ and some $j \in [n-1]$, then*

$$\mathcal{L}(\mathsf{Up}(D, V_{i,j+1}, H_{i,j+1}, j+1)) = \partial_{i,j+1}(\mathcal{V},\mathcal{H}).$$

Algorithm 1 Decision algorithm for grid-like CSPs.

```
Data: An $(m,n,k)$-grid CSP $(\mathcal{V},\mathcal{H})$.
Result: Yes if $(\mathcal{V},\mathcal{H})$ is satisfiable. No otherwise.
$D_1 \leftarrow D(\mathcal{H},1)$;
for $i = 2 \ldots m$ do
    $D_i \leftarrow \mathsf{Up}(D_{i-1}, V_{i-1,1}, 1)$;
    for $j = 2 \ldots n$ do
        $D_i \leftarrow \mathsf{Up}(D_i, V_{i-1,j}, H_{i,j}, j)$;
    end
end
Return Yes if $\mathcal{L}(D_m) \neq \emptyset$ and No otherwise.
```

The algorithm described in Algorithm 1 can be used to determine whether a given (m,n,k)-grid like CSP (V,H) is satisfiable. It turns out that the sequence of ODDs $D_1,...,D_m$ can be used to construct an actual solution in case it exists. The description of the construction is given below in Algorithm 2.

Algorithm 2 Construction of a solution of a satisfiable grid-like CSP.

```
Data: ODDs $D_1,\ldots,D_m$ constructed in Algorithm 1 where $\mathcal{L}(D_m) \neq \emptyset$.
Result: A solution for the $(m,n,k)$-grid like CSP $(V,H)$ input to Algorithm 1.
Let $s$ be a string in $\mathcal{L}(D_m)$. ;
$M_m \leftarrow s$;
for $i = m-1 \ldots 1$ do
    Let $s \in \mathcal{L}(D_i)$ be such that $s \otimes_V M_{i+1}$ belongs to $\mathcal{L}(D_i \otimes_V D_{i+1})$.;
    $M_i \leftarrow s$;
end
Return the $(m,n,k)$-matrix $M$ whose rows are $M_1,\ldots,M_m$. ;
```

From Corollary 7 and Algorithms 1 and 2, we infer the following theorem.

▶ **Theorem 8.** *Let $(\mathcal{V},\mathcal{H})$ be an s-smooth (m,n,k)-grid CSP. Then one can determine whether $(\mathcal{V},\mathcal{H})$ has a solution using Algorithm 1 in time $s^{O(k)} \cdot m^{O(1)} \cdot n^{O(1)}$. In case a solution exists, it can be constructed within the same time bounds using Algorithm 2.*

4 Experiments

In this section, we evaluate the performance of our algorithm and compare it with two general-purpose SAT solvers (minisat and glucose) and the integer-programming solver Coin-OR CBC.

We start by defining the notion of Pigeonhole grid-CSPs, a CSP with uniform vertical and horizontal constraints over an alphabet of size 5 encoding the pigeonhole principle. A more contrieved version of this CSP was studied in [6], under the name of *pigeonhole pictures*. It was proved in [6] that these pigeonhole pictures have polynomial smoothness, and that the straightforward propositional translation of these CSPs is hard for the bounded-depth Frege proof system.

In this section, we defined a simpler notion of pigeonhole grid CSP than the one employed in [6]. The simplicity of our definition is due to the fact that in our setting local constraints may vary according to the position in the grid, while in [6] local constraints were required to be uniform. Both the fact that these CSPs have polynomial smoothness and the fact that they are hard to bounded Frege are inherited from the corresponding results in [6]. In this section, we confirm empirically both of these theoretical results. Additionally, we show empirically that the straightforward integer-programming formulation of the Pigeonhole grid-CSP is hard for state-of-the-art integer programming tools such as Cplex and Coin-OR CBC.

4.1 The Pigeonhole Grid CSP

For each two positive integers m and n we define an $(m, n, 5)$-grid CSP

$$\mathsf{PHP}(m,n) = (\mathcal{V}(m,n), \mathcal{H}(m,n))$$

encoding the principle that m pigeons are placed into n holes. Clearly such a CSP should be satisfiable if and only if $m \leq n$. To make the definition more intuitive, instead of defining the local vertical and horizontal constraints as subsets of $[5] \times [5]$, we let $\Sigma = \{bb, bg, gb, gg, rr\}$ and assume that these local constraints are defined as subsets of $\Sigma \times \Sigma$. First, for each $i \in [m-1]$ and any $j \in [n]$, we let the local vertical constraint $V_{i,j}$ be equal to the following relation.

$$V = \{\ (xb, yb),\ (xb, rr),\ (rr, xg),\ (xg, yg)\ \mid\ x, y \in \{b, g\}\ \}.$$

Now, there are three types of local horizontal constraints. For $i \in [m]$, we set $H_{i,1}$ equal to the relation

$$H_{left} = \{\ (bx, by),\ (bx, rr),\ (rr, gy)\ :\ x, y \in \{b, g\}\ \}. \tag{3}$$

For each $i \in [m]$ and each $j \in [n-2]$, we set $H_{i,j}$ equal to the relation.

$$H_{middle} = \{\ (bx, by),\ (bx, rr),\ (rr, gy),\ (gx, gy)\ \ :\ \ x, y \in \{b, g\}\ \} \tag{4}$$

Finally, for each $i \in [m]$, we set $H_{i,n-1}$ equal to the relation

$$H_{right} = \{\ (bx, rr),\ (rr, gy),\ (gx, gy)\ \ :\ \ x, y \in \{b, g\}\ \}. \tag{5}$$

Intuitively, if M is a solution for $\mathsf{PHP}(m,n)$ then M has one row for each pigeon and one column for each hole. The $M_{i,j} = rr$ indicates that the i-th pigeon is placed at the hole j. On the other hand, $M_{i,j} = bx$ for some $x \in \{g, b\}$ indicates that the i-th pigeon is placed in some hole greater than j, while $M_{i,j} = gx$ for some $x \in \{b, g\}$ indicates that the i-th pigeon is placed in some hole smaller than j. Analogously, if $M_{i,j} = xb$ for some $x \in \{b, g\}$, then the pigeon that is placed at the j-th hole is greater than i, while if $M_{i,j} = xg$, then the pigeon that is placed at the j-th hole is smaller than i.

Note that the way in which the horizontal constraints are defined guarantees that exactly one pigeon must occur in each row of a solution. This is because there is no allowed pair $(xy, x'y')$ where x is blue and x' is green. Therefore in a solution, each row must have at least one entry with value rr. Additionally, the vertical constraints guarantee that that at most one pigeon will occur in each column. Indeed, if some pigeon occurs in a position (i, j) then the second color in each entry below (i, j) must be green, while the second color of each entry above (i, j) must be blue. Therefore no two pigeons are allowed to appear on the same column of a satisfying assignment. In Figure 2 we depict a solution to the pigeonhole CSP PHP$(4, 4)$, while one can readily check that the CSP PHP$(4, 3)$ has no solution.

i)

bb	rr	gb	gb
bb	bg	bb	rr
bb	bg	rr	gg
rr	gg	gg	gg

ii)

rr	gb	gb
bg	bb	rr
bg	rr	gg
bg	bg	

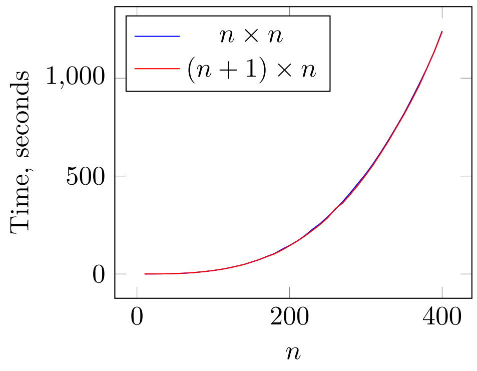
Figure 2 *i)* A solution to the pigeonhole CSP PHP$(4, 4)$. *ii)* A maximal partial solution to PHP$(4, 3)$. In this last case, it is not possible to assign a value to the entry $(4, 3)$ of the matrix in such a way that both the constraints $V_{3,3}$ and $H_{4,2}$ are satisfied.

▶ **Theorem 9** (Follow from results in [6]). *Let* PHP(m, n) *be the pigeonhole grid CSP defined above.*

1. PHP(m, n) *has a solution if and only if* $m \leq n$.
2. *The smoothness of* PHP *is bounded by* $m^{O(1)} \cdot n$.
3. *For each fixed* $d \in \mathbb{N}$, PHP$(m + 1, m)$ *require depth-d Frege proofs of superpolynomial size.*

4.2 Solving Pigeonhole Pictures with the ODD solver

Since the PHP(m, n) has smoothness upper bounded by $m^{O(1)} \cdot n$, there is no visible difference between the performance of the solver on barely-satisfiable instances PHP(m, m) and the performance of the solver on barely-unsatisfiable instances PHP$(m + 1, m)$. This is confirmed empirically in Figure 3.



n	w(D)	$m \times m$, s	$(m + 1) \times m$, s
50	51	2.172	2.220
100	101	17.834	17.780
150	151	60.715	60.552
200	201	145.587	144.179
250	251	291.692	286.952
300	301	511.754	505.581
350	351	818.724	813.929
400	401	1241.258	1236.734

Figure 3 (a) Performance of the ODD solver on PHP(m, m) and on PHP$(m + 1, m)$. The running times grows as $O(m^3)$. The plots corresponding to both cases almost match. (b) Execution times (in seconds) and maximum width of minimized deterministic ODDs occurring during the execution of the ODD algorithm. Note that the maximum width is identical in both test cases.

4.3 Experiments with SAT Solvers

First we describe the straightforward translation from (m, n, k)-grid CSPs to CNFs. We note that the obtained CNFs have width at most k. Given such a CSP $(\mathcal{V}, \mathcal{H})$, the formula $\psi(\mathcal{V}, \mathcal{H})$ has a variable $x_{ij\sigma}$ for each $(i, j) \in [m] \times [n]$, and each $\sigma \in [k]$. Intuitively, the variable $x_{ij\sigma}$ is true if the position (i, j) of a solution is set to σ. The following set of clauses specifies that in a satisfying assignment, precisely value is associated with entry (i, j).

$$\mathrm{OneSymbol}(M, i, j) \equiv \bigvee_{s\in[k]} x_{ijs} \;\wedge \bigwedge_{s,s'\in[k],s\neq s'} (\overline{x}_{ijs} \vee \overline{x}_{ijs'}) \tag{6}$$

The next set of clauses expresses the fact that no pair $(\sigma, \sigma') \notin H_{i,j}$ occurs in consecutive horizontal positions at row i.

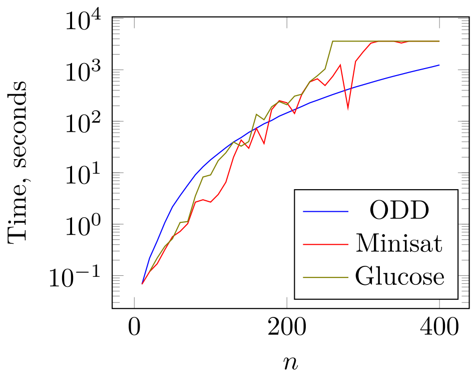
$$\mathrm{Horizontal}(M, i) \equiv \bigwedge_{(\sigma,\sigma')\notin H_{i,j}, j\in[n-1]\}} (\overline{x}_{ij\sigma} \vee \overline{x}_{i(j+1)\sigma'}) \tag{7}$$

Similarly, the following set of of clauses expresses the fact that that no pair $(\sigma, \sigma') \notin V_{i,j}$ occurs in consecutive vertical positions at column j.

$$\mathrm{Vertical}(M, j) \equiv \bigwedge_{(\sigma,\sigma')\notin V_{i,j}, i\in[m-1]\}} (\overline{x}_{ij\sigma} \vee \overline{x}_{(i+1)j\sigma'}) \tag{8}$$

Finally, we set the formula $\psi(M)$ as follows.

$$\psi(\mathcal{V}, \mathcal{H}) \equiv \bigwedge_{i=1}^{m} \mathrm{Horizontal}(M, i) \;\wedge\; \bigwedge_{j=1}^{n} \mathrm{Vertical}(M, j) \wedge \bigwedge_{ij} \mathrm{OneSymbol}(M, i, j) \tag{9}$$



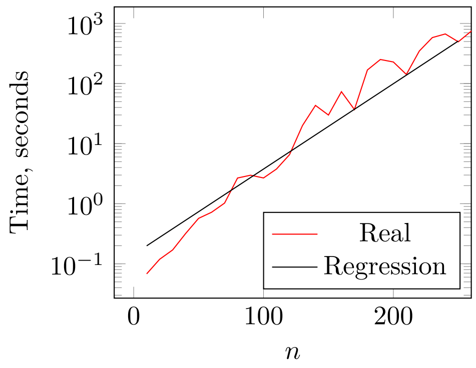
n	ODD, s	Minisat, s	Glucose, s
40	1.072	0.319	0.370
70	5.744	1.024	1.123
100	17.834	2.683	9.057
130	39.440	19.997	39.525
160	73.322	73.149	136.082
190	125.788	251.317	241.092
220	195.433	348.214	336.554
250	291.692	495.185	1038.347

Figure 4 Performances of ODD, Minisat and Glucose solvers on Pigeonhole picture of size $n \times n$.

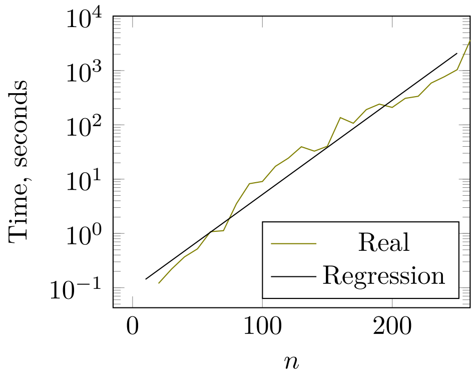
The test cases corresponding to the CNF translation of the grid CSPs $\mathsf{PHP}(m, m)$ and $\mathsf{PHP}(m + 1, m)$ were given as input to the SAT solvers Minisat 2.2[1] and Glucose 4.1[2].

The performance of these solvers on the barely satisfiable case is plotted on Figure 4. The timeout for each experiment was set at 3600 seconds. As it can be seen, Glucose solver times out for $n > 250$, and Minisat times out for $n > 350$. The performance of SAT solvers seems to be exponential, while ODD solver performs clearly in polynomial time.

[1] `http://www.minisat.se/Main.html`
[2] `https://www.labri.fr/perso/lsimon/glucose/`



(a) Minisat.

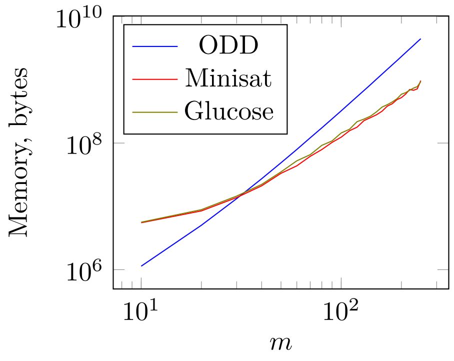


(b) Glucose.

Figure 5 Exponential regressions for SAT solver running times.

Based on the results of the tests, the empirical exponential approximations were estimated for running times of Minisat and Glucose solvers. For Minisat, it is $O^*(1.033^n)$, while for Glucose it is $O^*(1.041^n)$. The regressions are plotted on Figure 5.

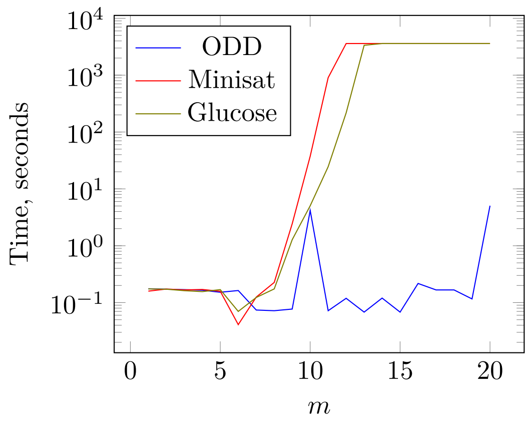
Interestingly, the amount of memory used by the ODD solver is significantly larger than the amount of memory used by SAT solvers. As it is clearly shown on Figure 6, both amounts are polynomial, but the degree of the polynomial for the ODD solver is larger. SAT solvers most probably use linear amount of memory in terms of the formula length (which is quadratic with respect to m). However, the ODD-based solver needs to store ODDs for all layers in order to be able to restore the solution, so the amount of memory used by this solver on this test is cubic.



n	ODD, Mb	Minisat, Mb	Glucose, Mb
40	27.273	21.082	22.202
70	121.126	61.706	65.559
100	323.002	123.988	144.478
130	677.117	226.307	239.545
160	1227.509	318.518	370.232
190	2007.272	485.630	500.627
220	3077.601	702.075	739.147
250	4458.830	969.132	934.675

Figure 6 Memory used by ODD, Minisat and Glucose solvers on Pigeonhole picture of size $m \times m$.

A more expressive difference is achieved for the barely unsatisfiable case, $\mathsf{PHP}(m+1, m)$. The performance is plotted in figure 7. Both solvers time out in this case even for for $n = 14$. Therefore, we can conclude that CNF encodings corresponding to $\mathsf{PHP}(m+1, m)$ are extremely hard for the tested SAT solvers.



m	ODD, s	Minisat, s	Glucose, s
5	0.152	0.158	0.169
6	0.163	0.041	0.070
7	0.074	0.125	0.123
8	0.072	0.226	0.174
9	0.077	2.436	1.277
10	4.169	37.477	5.043
11	0.072	903.836	24.553
12	0.119	3598.044	217.029

Figure 7 Performances of ODD, Minisat and Glucose solvers on Pigeonhole picture of size $(m+1) \times m$. Both SAT solvers timed out for $m \geq 14$, while the ODD solver run without problems in all test sizes (up to $m \leq 400$).

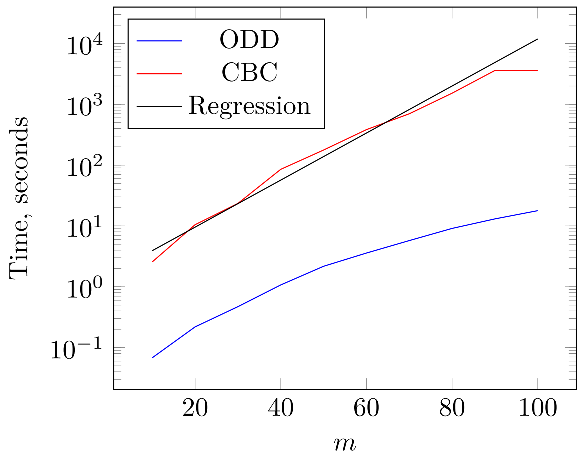
4.4 Integer Programming Translation

The ILP encoding of grid CSPs is done in a similar way to the CNF encoding. More precisely, a variable $x_{i,j,s} \in \{0,1\}$ is created for each $(i,j) \in [m] \times [n]$ and each $s \in [k]$. Then, the following constraints are added:

$$\text{OneSymbol}(M,i,j) \equiv \sum_{s \in [k]} x_{ijs} = 1 \tag{5'}$$

$$\text{Horizontal}(M,i,j,\sigma,\sigma') \equiv \ x_{ij\sigma} + x_{i(j+1)\sigma'} < 2, \quad \text{if } (\sigma,\sigma') \notin H_{i,j} \tag{6'}$$

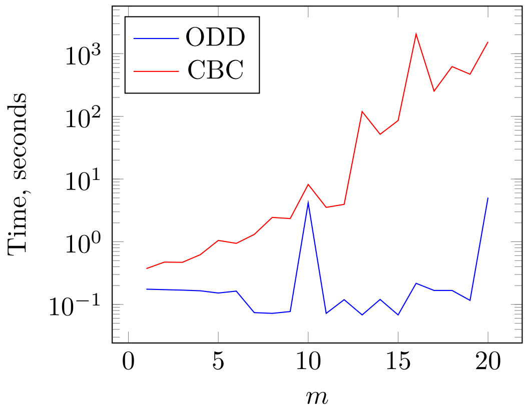
$$\text{Vertical}(M,i,j,\sigma,\sigma') \equiv \ x_{ij\sigma} + x_{(i+1)j\sigma'} < 2, \quad \text{if } (\sigma,\sigma') \notin V_{i,j} \tag{7'}$$



n	ODD, s	CBC, s
10	0.068	2.565
20	0.219	10.489
30	0.471	23.465
40	1.072	85.323
50	2.172	177.186
60	3.584	383.363
70	5.744	698.603
80	9.105	1522.991

Figure 8 Performances of ODD and CBC solvers on Pigeonhole picture if size $m \times m$.

The resulting ILP instances were given to the Coin-OR CBC[3] solver. The performance is plotted on Figure 8. For $m \times m$ instances the solver timed out for $m = 90$. The running time of the solvers grows clearly as an exponential function. On each of the test cases, the running time of the ILP sover is several orders of magitude above the running time of the ODD-based solver. For the ILP solver empirical exponential approximation of $O^*(1.093^n)$ was also built. The regression is shown on Figure 8. As in the case of SAT solvers, for the barely unsatisfiable case the ILP solver timed out much earlier (for $m = 21$). The results for $m \leq 20$ are plotted on Figure 9.



m	ODD, s	CBC, s
6	0.163	0.942
8	0.072	2.422
10	4.169	8.201
12	0.119	3.946
14	0.120	51.762
16	0.217	2046.332
18	0.167	622.360
20	5.060	1543.712

Figure 9 Performances of ODD and CBC solvers on Pigeonhole picture if size $(m + 1) \times m$.

5 Conclusion

In this work, we have lifted the notion of smoothness for pictures (grid-CSPs with uniform vertical and horizontal constraints) to the context of general grid CSPs, where the vertical and horizontal constraints at each position (i, j) may depend on (i, j). We have shown that the satisfiability problem for grid-CSPs of polynomial smoothness can be solved in polynomial time. Additionally, we have given evidence for the relevance of the concept of smoothness in practical situations by demonstrating empirically that the class for pigeonhole grids can be solved in cubic time.

This opens up the possibility of applying our algorithm to grid CSPs involving constraints where one particular object can appear at most once in any row/column, such as the problem of placing r towers in an $m \times n$ chess grid in such a way that no tower attacks each other. It can be shown in this case that the ODDs occurring in the execution of the algorithm have size polynomial in m, n and r.

References

1 Gilles Audemard and Laurent Simon. On the glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.

2 Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, pages 319–351, 2004.

[3] https://github.com/coin-or/Cbc

3 Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 274–282. IEEE, 1996.

4 Samuel R Buss, Jan Hoffmann, and Jan Johannsen. Resolution trees with lemmas: Resolution refinements that characterize DLL algorithms with clause learning. *Logical Methods in Computer Science*, 4, 2008.

5 Alessandra Cherubini, Stefano Crespi Reghizzi, Matteo Pradella, and Pierluigi San Pietro. Picture languages: Tiling systems versus tile rewriting grammars. *Theoretical Computer Science*, 356(1):90–103, 2006.

6 Mateus de Oliveira Oliveira. Satisfiability via smooth pictures. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 13–28. Springer, 2016.

7 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and applications of satisfiability testing*, pages 502–518. Springer, 2003.

8 John Forrest and Robin Lougee-Heimer. Cbc user guide. In *Emerging theory, methods, and applications*, pages 257–277. INFORMS, 2005.

9 Dora Giammarresi and Antonio Restivo. Recognizable picture languages. *International Journal of Pattern Recognition and Artificial Intelligence*, 6(2&3):241–256, 1992.

10 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

11 Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, and Allen Van Gelder. Clause learning can effectively P-simulate general propositional resolution. In *Proc. of the 23rd National Conference on Artificial Intelligence (AAAI 2008)*, pages 283–290, 2008.

12 Changwook Kim and Ivan Hal Sudborough. The membership and equivalence problems for picture languages. *Theoretical Computer Science*, 52(3):177–191, 1987.

13 Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *The Journal of Symbolic Logic*, 59(01):73–86, 1994.

14 Jan Krajíček, Pavel Pudlák, and Alan Woods. An exponential lower bound to the size of bounded depth frege proofs of the pigeonhole principle. *Random Structures & Algorithms*, 7(1):15–39, 1995.

15 Michel Latteux and David Simplot. Recognizable picture languages and domino tiling. *Theoretical computer science*, 178(1):275–283, 1997.

16 Hermann A Maurer, Grzegorz Rozenberg, and Emo Welzl. Using string languages to describe picture languages. *Information and Control*, 54(3):155–185, 1982.

17 Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

18 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, 2011.

19 Toniann Pitassi, Paul Beame, and Russell Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational complexity*, 3(2):97–140, 1993.

20 Azriel Rosenfeld. *Picture languages: formal models for picture recognition.* Academic Press, 2014.

21 David Simplot. A characterization of recognizable picture languages by tilings by finite sets. *Theoretical Computer Science*, 218(2):297–323, 1999.

22 Gift Stromoney, Rani Siromoney, and Kamala Krithivasan. Abstract families of matrices and picture languages. *Computer Graphics and Image Processing*, 1(3):284–307, 1972.