Optimal Discretization is Fixed-parameter Tractable*

Stefan Kratsch† Tomáš Masařík‡ Irene Muzi§ Marcin Pilipczuk¶ Manuel Sorge‖

Abstract

Given two disjoint sets W_1 and W_2 of points in the plane, the Optimal Discretization problem asks for the minimum size of a family of horizontal and vertical lines that separate W_1 from W_2, that is, in every region into which the lines partition the plane there are either only points of W_1, or only points of W_2, or the region is empty. Equivalently, Optimal Discretization can be phrased as a task of discretizing continuous variables: We would like to discretize the range of x-coordinates and the range of y-coordinates into as few segments as possible, maintaining that no pair of points from $W_1 \times W_2$ are projected onto the same pair of segments under this discretization.

We provide a fixed-parameter algorithm for the problem, parameterized by the number of lines in the solution. Our algorithm works in time $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$, where k is the bound on the number of lines to find and n is the number of points in the input.

Our result answers in positive a question of Bonnet, Giannopolous, and Lampis [IPEC 2017] and of Froese (PhD thesis, 2018) and is in contrast with the known intractability of two closely related generalizations: the Rectangle Stabbing problem and the generalization in which the selected lines are not required to be axis-parallel.

1 Introduction

For three numbers $a, b, c \in \mathbb{Q}$, we say that b *is between* a *and* c if $a < b < c$ or $c < b < a$. The input to Optimal Discretization consists of two sets $W_1, W_2 \subseteq \mathbb{Q} \times \mathbb{Q}$ and an integer k. A pair (X, Y) of sets $X, Y \subseteq \mathbb{Q}$ is called a *separation* (of W_1 and W_2) if for every $(x_1, y_1) \in W_1$ and $(x_2, y_2) \in W_2$ there exists an element of X between x_1 and x_2 or an element of Y between y_1 and y_2. We also call the elements of $X \cup Y$ *lines*, for the geometric interpretation of a separation (X, Y) is as follows. We draw $|X|$ vertical lines at x-coordinates from X and $|Y|$ horizontal lines at y-coordinates from Y and focus on the partition of the plane into $(|X| + 1)(|Y| + 1)$ regions given by the the drawn lines. We require that the closure of every such region does not contain both a point from W_1 and a point from W_2. The optimization version of Optimal Discretization asks for a separation (X, Y) minimizing $|X| + |Y|$; the decision version takes also an integer k as an input and looks for a separation (X, Y) with $|X| + |Y| \leq k$.

In this work we establish fixed-parameter tractability of Optimal Discretization by showing the following.

Theorem 1.1. Optimal Discretization *can be solved in time* $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$, *where* k *is the upper bound on the number of lines and* n *is the number of input points.*

Related work. Optimal Discretization stems from two lines of research: the name-giving preprocessing technique of discretization from machine learning, and geometric covering problems.

In a geometric covering problem we are given a universe U of objects, for example, a set of axis-parallel rectangles in the plane, and a set system over U, for example for each axis-parallel line the set of rectangles intersected, or stabbed, by that line. We are then asked to select a minimal number of subsets of the set system that cover U. Our example yields the well-studied Rectangle Stabbing problem [12, 13, 16, 18]. Geometric covering problems arise in many different

* This research is a part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme Grant Agreement no. 714704 (TM, IM, MP, and MS) and 648527 (IM). TM also supported by Charles University, student grant number SVV–2017–260452. MS also supported by Alexander von Humboldt-Stiftung. MS' main work done while with University of Warsaw and funded by the ERC.

† Institut für Informatik, Humboldt-Universität zu Berlin, Germany, kratsch@informatik.hu-berlin.de

‡ Faculty of Mathematics and Physics, Charles University, Czech Republic & Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland, masarik@kam.mff.cuni.cz

§ Technische Universität Berlin, Germany, irene.muzi@tu-berlin.de

¶ Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland, malcin@mimuw.edu.pl

‖ Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland and Institute of Logic and Computation, TU Wien, Austria, manuel.sorge@mimuw.edu.pl

applications such as reducing interference in cellular networks, facility location, or railway-network maintenance and are subject to intensive research (see e.g. [1–4, 7, 8, 12, 16, 17, 19]). Within that context, the objects to be covered in Optimal Discretization are the axis-parallel rectangles defined by taking each of pair of points $w_1 \in W_1$ and $w_2 \in W_2$ as two antipodal vertices of a rectangle. These rectangles then need to be stabbed by at most k horizontal or vertical lines. Focusing on the parameterized complexity of geometric covering problems, one could roughly classify them into fixed-parameter tractable when the elements of the universe are pairwise disjoint [16, 17, 19, 21] but W[1]-hard when they may non-trivially overlap [6, 12, 16]. Two particular examples of the latter category are the W[1]-hard Rectangle Stabbing parameterized by the number of lines to select [12], and the variant of Optimal Discretization in which the lines to select may have arbitrary slopes [6]. Optimal Discretization also belongs to the second category, yet, somewhat surprisingly, turns out to be fixed-parameter tractable. Bonnet et al. [6] had explicitly posed this as an open question. They proved tractability under a weaker parameterization, namely the cardinality of the smaller of the sets W_1 and W_2 and conjectured that Optimal Discretization is fixed-parameter tractable with respect to k.

Discretization is a preprocessing technique in machine learning in which continuous features of the elements of a data set are discretized in order to make the data set amenable to classification by clustering algorithms that work only with discrete features, to speed up algorithms whose running time is sensitive to the number of different feature values, or to improve interpretability of learning results [15, 22–24]. Various discretization techniques have been studied and are implemented in standard machine learning frameworks [15, 23]. Optimal Discretization is a formalization of the so-called supervised discretization [24] for two features and two classes; herein, we are given a data set labeled with classes and want to discretize each continuous feature into a minimum number of distinct values so as to avoid mapping two data points with distinct classes onto the same discretized values [11, 14]. Within this language, fixed-parameter tractability of Optimal Discretization was posed as an open question by Froese [14, Section 5.5].

Approach. In the proof of Theorem 1.1 we proceed as follows. Let (X_0, Y_0) be an approximate solution (that can be obtained via e.g. the iterative compression technique or a known polynomial-time 2-approximation algorithm [9]). Let (X, Y) be an optimal solution. For every two consecutive elements of X_0 we guess how many (if any) elements of X are between them and similarly for every two consecutive elements of Y_0. This gives us a general picture of the *layout* of the lines of X, X_0, Y, and Y_0.

Consider all $\mathcal{O}(k^2)$ *cells* in which the vertical lines with x-coordinates from $X_0 \cup X$ and the horizontal lines with y-coordinates from $Y_0 \cup Y$ partition the plane. Similarly, consider all $\mathcal{O}(k^2)$ *supercells* in which the vertical lines with x-coordinates from X_0 and the horizontal lines with y-coordinates from Y_0 partition the plane. Every cell is contained in exactly one supercell. For every cell, guess whether it is empty or contains a point of $W_1 \cup W_2$. Note that the fact that (X_0, Y_0) is a solution implies that every supercell contains only points from W_1, only points from W_2, or is empty. Hence, for each nonempty cell we can deduce whether it contains only points of W_1 or only points of W_2. Check Figure 1 for an example of such a situation.

We treat every element of $X \cup Y$ as a variable with a domain being all rationals between the closest lines of X_0 or Y_0, respectively.

If we know that there exists an optimal solution (X, Y) such that between every two consecutive elements of X_0 there is at most one element of X and between every two consecutive elements of Y_0 there is at most one element of Y, we can proceed as above. For every two consecutive elements of X_0 we guess (trying both possibilities) whether there is an element of X between them and similarly for every two consecutive elements of Y_0. This ensures that every cell has at most two borders coming from $X \cup Y$. Also, as before, for every cell we guess if it is empty. Thus, for every cell $\mathcal{C}$ that is guessed to be empty and every point p in the supercell containing $\mathcal{C}$ we add a constraint binding the at most two borders of $\mathcal{C}$ from $X \cup Y$, asserting that p does not land in $\mathcal{C}$.

The crucial observation is that the CSP instance constructed in this manner admits the median as a so-called majority polymorphism and such CSPs are polynomial-time solvable (for more on majority polymorphisms, which are ternary near-unanimity polymorphisms, see e.g. [5] or [10]). We remark that Agrawal et al. [2] recently obtained a fixed-parameter algorithm for the Art Gallery problem by reducing to an equivalent CSP variant, which they called Monotone 2-CSP and directly proved to be polynomial-time solvable.

However, the above approach breaks down if there are multiple lines of X between two consecutive elements of X_0. One can still construct a CSP instance with variables corresponding to the lines of $X \cup Y$ and constraints asserting that the content of the cells is as we guessed it to be. However, it is possible to show that the constructed CSP instance no longer admits a majority polymorphism.

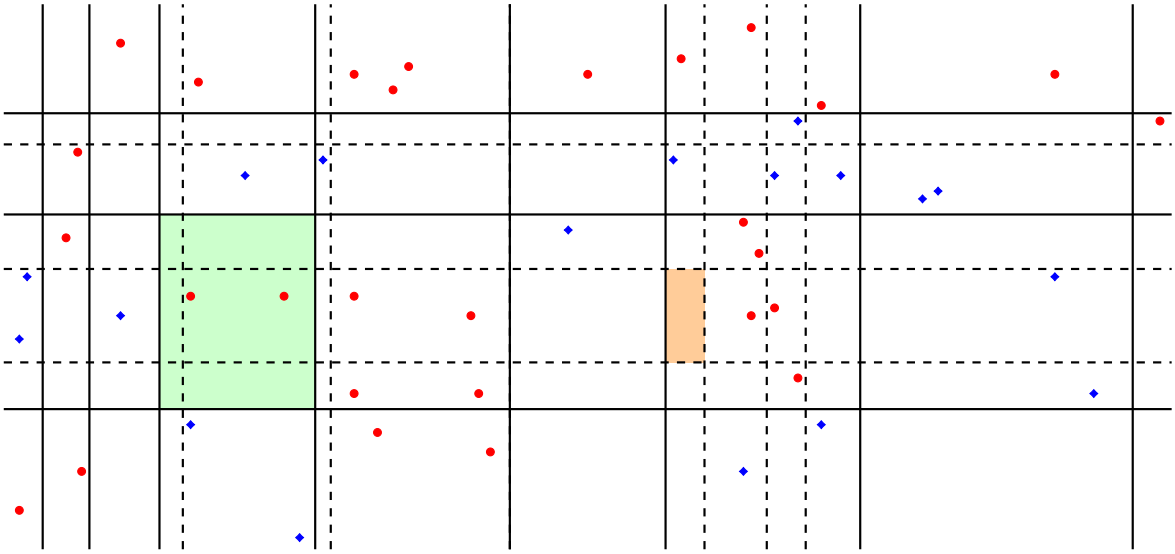

Figure 1: Example of a basic situation. An approximate solution (X_0, Y_0) is denoted by solid lines, an optimal solution (X, Y) by dashed lines. A supercell is marked in green and a cell in orange.

To cope with that, we perform an involved series of branching and color-coding steps on the instance to clean up the structure of the constructed constraints and obtain a tractable CSP instance. We were not able to reduce to a known tractable case; instead, in Section 4 we introduce a special CSP variant and prove its tractability via a nontrivial branching algorithm.

Organization of the paper. This extended abstract features a detailed overview of the algorithm in Section 2, a full description of the algorithm for the auxiliary CSP problem in Sections 3 and 4, and a discussion of future research directions in Section 5. A complete and standalone proof of the main result, Theorem 1.1, can be found in the arXiv version [20].

2 Overview

Let (W_1, W_2, k) be an input Optimal Discretization instance.

Layout and cell content. As discussed in the introduction, we start by computing a 2-approximate solution (X_0, Y_0) and, in the first branching step, guess the *layout* of (X_0, Y_0) and the sought solution (X, Y), that is, how many elements of X are between two consecutive elements of X_0 and how many elements of Y are between two consecutive elements of Y_0. By adding a few artificial elements to X_0 and Y_0 that bound the picture, we can assume that the first and the last element of $X_0 \cup X$ is from X_0 and the first and the last element of $Y_0 \cup Y$ is from Y_0. Also, simple discretization steps ensure that all elements of X_0, Y_0, X, and Y are integers and $X_0 \cap X = \emptyset$, $Y_0 \cap Y = \emptyset$.

In the layout, we have $\mathcal{O}(k^2)$ *cells* in which the vertical lines with x-coordinates from $X_0 \cup X$ and the horizontal lines with y-coordinates from $Y_0 \cup Y$ partition the plane. If we only look at the way in which the lines from X_0 and Y_0 partition the plane, we obtain $\mathcal{O}(k^2)$ *apx-supercells*. If we only look at the way in which the lines from X and Y partition the plane, we obtain $\mathcal{O}(k^2)$ *opt-supercells*. Every cell is in exactly one apx-supercell and exactly one opt-supercell.

A second branching step is to guess, for every cell, whether it is empty or not. Note that, since (X_0, Y_0) is an approximate solution, a nonempty cell contains points only from W_1 or only from W_2, and we can deduce which one is the case from the instance. At this moment we verify whether the guess indeed leads to a solution (X, Y): We reject the current guess if there is an opt-supercell containing both a cell guessed to have an element of W_1 and a cell guessed to have an element of W_2. Consequently, if we choose X and Y to ensure that the cells guessed to be empty are indeed empty, (X, Y) will be a solution to the input instance.

CSP formulation. We phrase the problem resulting from adding the information guessed above as a CSP instance as follows. For every sought element x of X, we construct a variable whose domain is all integers between the (guessed) closest elements of X_0. Similarly, for every sought element y of Y, we construct a variable whose domain is all integers between the (guessed) closest elements of Y_0. If between the same two ele-

ments of X_0 there are multiple elements of X, we add binary constraints between them that force them to be ordered as we planned in the layout, and similarly for Y_0 and Y. Furthermore, for every cell cell guessed to be empty and for every point p in the apx-supercell containing cell, we add a constraint that binds the borders of cell from $X \cup Y$ asserting that p is not in cell.

Clearly, the constructed CSP instance is equivalent to choosing the values of $X \cup Y$ such that the layout is as guessed and every cell that is guessed to be empty is indeed empty. This ensures that a satisfying assignment of the constructed CSP instance yields a solution to the input Optimal Discretization instance and, in the other direction, if the input instance is a yes-instance and the guesses were correct, the constructed CSP instance is a yes-instance. It "only" remains to study the tractability of the class of constructed CSP instances.

The instructive polynomial-time solvable case. As briefly argued in the introduction, if no two elements of X are between two consecutive elements of X_0 and no two elements of Y are between two consecutive elements of Y_0, then the CSP instance admits the median as a majority polymorphism and therefore is polynomial-time solvable [5].[1] Let us have a more in-depth look at this argument.

In the above described case, every cell has at most two borders from $X \cup Y$ and thus every introduced constraint is of arity at most 2. As an example, consider a cell cell between $x \in X_0$ and $x_i \in X$ and between $y_j \in Y$ and $y \in Y_0$ that is guessed to be empty. For every point $p = (x^p, y^p)$ in the apx-supercell containing cell, we add a constraint that p is not in cell. Observe that this constraint is indeed equivalent to $(x_i < x^p) \vee (y_j > y^p)$, see the right panel of Figure 2. It is straightforward to verify that a median of three satisfying assignments in this constraint yields a satisfying assignment as well. That is, the median is a majority polymorphism. Observe also that the constraints yielded for cells cell with different configurations of exactly two borders from $X_0 \cup Y_0$ and exactly two borders from $X \cup Y$ yield similar constraints.

As a second example, consider a cell cell with exactly one border from $X \cup Y$, say between $x \in X$ and $x_i \in X_0$ and between $y, y' \in Y_0$; see the left panel of Figure 2. If cell is guessed to be empty, then for every p in the apx-supercell containing cell we add a constraint that $p = (x^p, y^p)$ is not in cell. This constraint is a unary constraint on x_i, in this case $(x_i < x^p)$. We can replace this constraint with a filtering step that removes from the domain of x_i the values that do not satisfy it.

This concludes the sketch why the problem is tractable if there are no two elements of X between two consecutive elements of X_0 and no two elements of Y between two consecutive elements of Y_0.

Difficult constraints. Let us now have a look what breaks down in the general picture.

First, observe that monotonicity constraints, constraints ensuring that the lines are in the correct order, are constraints of the form $(x_i < x_{i+1})$ and $(y_i < y_{i+1})$, and thus are simple. To see this, observe either that the median is again a majority polymorphism for them, or that $(x_i < x_{i+1})$ can be expressed as a conjunction of constraints $(x_i \leq a) \vee (x_{i+1} > a)$ over all a in the domain of x_i.

Second, consider a cell cell that has all four borders from $X \cup Y$. This cell is actually an opt-supercell as well, contained in a single apx-supercell. Observe that, since (X_0, Y_0) is a solution, this opt-supercell will never contain both a point of W_1 and a point of W_2, regardless of the choices of the values of the borders of cell within their domains. Thus, we may ignore the constraints for such cells.

The only remaining cells are cells with three borders from $X \cup Y$. As shown on Figure 3, they can be problematic: In this particular example, we want the striped cell to be empty of red points, to allow its neighbor to the right to contain a blue point. In the construction above, such a cell yields complicated ternary constraints on its three borders in $X \cup Y$.

Reduction to binary constraints. Our first step to tackle cells with three borders from $X \cup Y$ (henceforth called *ternary cells*) is to break down the constraints imposed by them into binary constraints.

Consider the striped empty cell as in Figure 3. It has three borders from $X \cup Y$: two from Y and one from X. The two borders from Y are two consecutive elements from Y between the same two elements of Y_0 and the border from X is the *last* element of X in the segment between two elements of X_0. Thus, a constraint imposed by a ternary cell always involves two consecutive elements of X or Y and first or last element of Y resp. X, where first/last refers to a segment between two consecutive elements of Y_0 resp. X_0.

Consider now an example in Figure 4: The top panel represents the solution, whereas the bottom panel represents the (correctly guessed) layout and cell con-

[1] We do not define majority polymorphisms here, since they are only instructive in getting an intuition for the difficulties of the problem and will not be needed in the formal proof. Intuitively, a CSP with ordered domains admits the median as a majority polymorphism if for any three satisfying assignments, taking for each variable the median of the three assigned values yields another satisfying assignment. A polynomial-time algorithm for the special case of CSP considered in this paragraph can also be obtained by a reduction to satisfiability of 2-CNF SAT formulas.

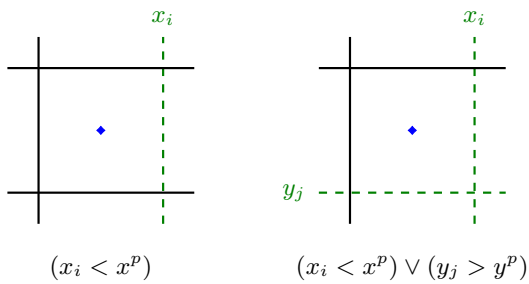

Figure 2: Constraints for blocks with one or two borders from $X \cup Y$.

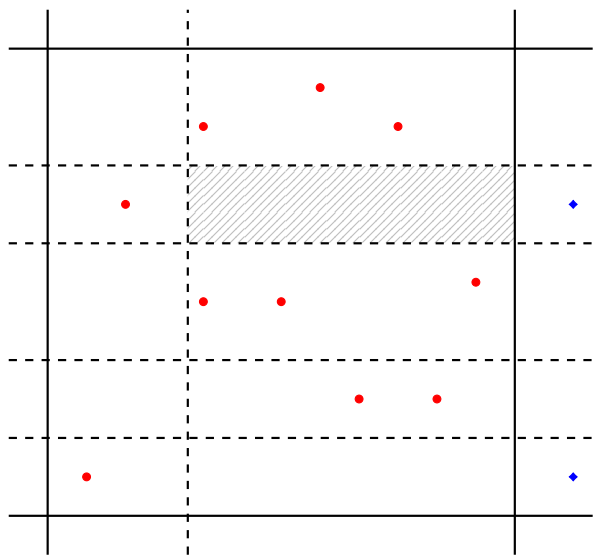

Figure 3: Problematic empty cell with three borders from $X \cup Y$.

tent. Assume now that, apart from the layout and cell content, we have somehow learned the value of x_1 (the position of the first vertical green line). Consider the area between x_1 and x_2 in this figure between the top and bottom black line. Then the set of red points between x_1 and x_2 is fixed; moving x_2 around only changes the set of blue points. Furthermore, the guessed information about layout and cell content determines, if one scans the area in question from top to bottom, how many *alternations* of *blocks* of red and blue points one should encounter. For example, scanning the area between x_1 and x_2 in Figure 4 from top to bottom one first encounters a block of blue points, then a block of red points, then blue, red, blue, and red in the end.

A crucial observation is that, for a fixed value of x_1, the set of values of x_2 that give the correct alternation (as the guessed cell content has predicted) is a segment: Putting x_2 too far to the left gives too few alternations due to too few blue points and putting x_2 too far to the right gives too many alternations due to too many blue points in the area of interest. Furthermore, as the value of x_1 moves from left to right, the number of red points decrease, and the aforementioned "allowed interval" of values of x_2 moves to the right as well (one needs more blue points to give the same alternation in the absence of some red points).

In the scenario as in Figure 4, let us introduce a binary constraint binding x_1 and x_2, asserting that the alternation in the discussed area is as the cell-content guess predicted. From the discussion above we infer that this constraint is of the same type as the constraints for cells with two borders of $X \cup Y$ and thus simple: It can be expressed as a conjunction of a number of clauses of the type $(x_1 < a) \vee (x_2 > b)$ and $(x_1 > a) \vee (x_2 < b)$ for constants a and b. See Figure 5.

The second crucial observation is that, if one fixes the value of x_1, then not only does this determine the set of red points in the discussed area, but also how they are

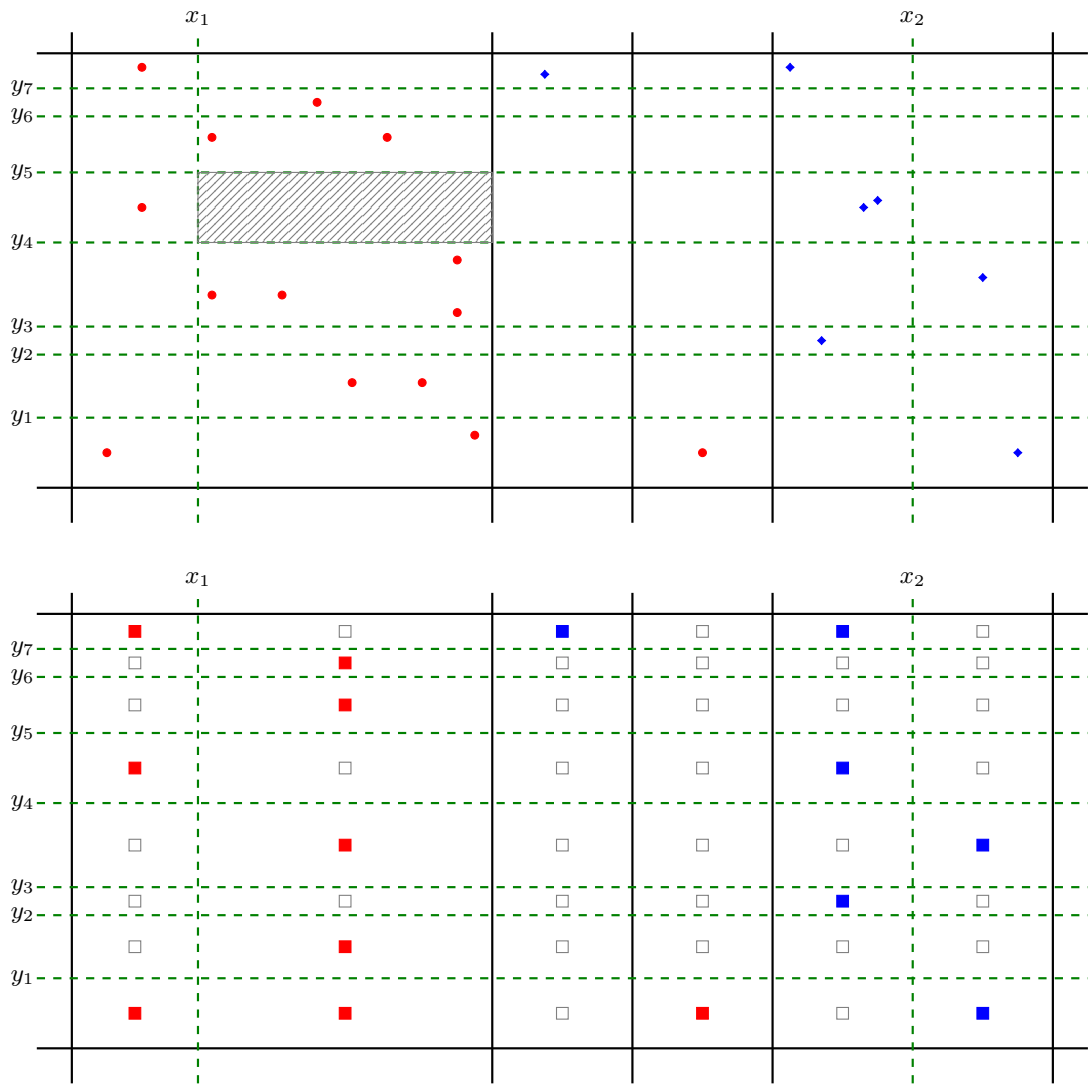

Figure 4: A more complicated scenario where a problematic cell (striped) occurs. The panel below represents the guessed cell content.

partitioned into blocks in the said alternation. Indeed, moving x_2 to the right only adds more blue points, but since the alternation is fixed, more blue points join existing blue blocks instead of creating new blocks (as this would increase the number of alternations). Hence, the value of x_1 itself implies the partition of the red points into blocks.

Now consider the striped cell in Figure 4, between lines y_4 and y_5. It is guessed to be empty. Instead of handling it with a ternary constraint as before, we handle it as follows: Apart from the constraint binding x_1 and x_2 asserting that the alternation is as guessed, we add a constraint binding x_1 and y_4, asserting that, for every fixed value of x_1, line y_4 is above the second (from the top) red block, and a constraint binding x_1 and y_5, asserting that for every fixed value of x_1, line y_5 is below the first (from the top) red block. It is not hard to verify that the introduced constraints, together with monotonicity constraints ($y_i < y_{i+1}$), imply that the striped cell is empty.

Thus, it remains to understand how complicated the constraints binding x_1 and y_4/y_5 are. Unfortunately, they may not have the easy "tractable" form as the constraints described so far (e.g., like in Figure 5). Consider an example in Figure 6, where a number of possible positions of the solution lines in $X \cup Y$ (green) have been depicted with various line styles. The constraint between the left vertical line and the middle

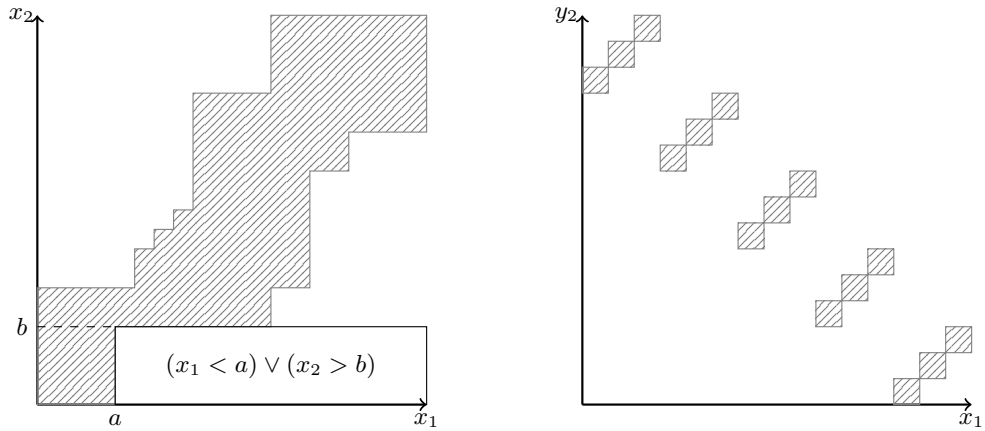

Figure 5: Left panel: An example of the space of allowed pairs of values (striped) for a constraint binding x_1 and x_2, asserting that the alternation of the area in between is as guessed. Such a constraint can be expressed as a conjunction of simple constraints as in the figure. Right panel: An example of the constraint binding x_1 (left vertical green line) and y_2 (middle horizontal green line) in Figure 6, which is not that simple.

horizontal line has been depicted in the right panel of Figure 5. For such a constraint, the median is not necessarily a majority polymorphism. In particular, such a constraint cannot be expressed in the style in which we expressed all other constraints so far.

Our approach is now as follows:

1. Introduce a class of CSP instances that allow constraints both as in the left and right panel of Figure 5 and show that the problem of finding a satisfying assignment is fixed-parameter tractable when parameterized by the number of variables.
2. By a series of involved branching and color coding steps, reduce the Optimal Discretization instance at hand to a CSP instance from the aforementioned tractable class. In some sense, our reduction shows that the example of Figure 6 is the most complicated picture one can encode in an Optimal Discretization instance.

In the remainder of this overview we focus on the first part above. As we shall see in a moment, there is strong resemblance of the introduced class to the constraints of Figure 5. The highly technical second part, spanning over most of [20, Section 4], takes the analysis of Figure 4 as its starting point and investigates deeper how the red/blue blocks change if one moves the lines x_1 and x_2 around.

Tractable CSP class. An instance of Forest CSP consists of a forest G, where the vertices $V(G)$ are variables, a domain $[n_T] = \{1, 2, \ldots, n_T\}$ for every connected component T of G, shared among all variables of T, and a number of constraints, split into two families: segment-reversion and downwards-closed constraints.

A permutation π of $[n]$ is a *segment reversion*, if its matrix representation looks for example like this:

$$\begin{matrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{matrix}$$

Formally, π is a segment reversion if there exist integers $1 = a_1 < a_2 < \ldots < a_r = n + 1$ such that for every $x \in [n]$, if $i \in [r - 1]$ is the unique index such that $a_i \leq x < a_{i+1}$, then $\pi(x) = a_{i+1} - 1 - (x - a_i)$. That is, π reverses a number of disjoint segments in the domain $[n]$. Note the resemblance of the matrix above and the right panel of Figure 5.

With every edge $e = uv \in E(G)$ in a component T, the Forest CSP instance contains a segment reversion π_e of $[n_T]$ and a constraint asserting that $\pi_e(u) = v$. Note that segment reversions are involutions, so $\pi_e(u) = v$ is equivalent to $\pi_e(v) = u$. One can think of the whole component T of G as a single super-variable: Setting the value of a single variable in a component T propagates the value over the segment reversions on the edges to the entire tree. Thus, every tree T has n_T different allowed assignments.

A relation $R \subseteq [n_1] \times [n_2]$ is *downwards-closed* if

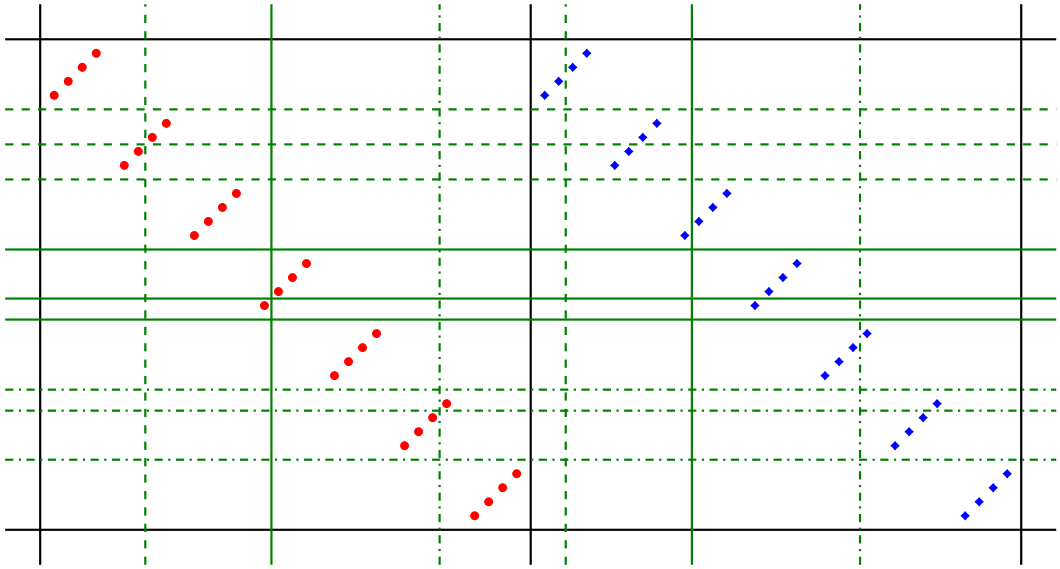

Figure 6: An example where the constraint between the left vertical solution line in X and the middle horizontal solution line in Y is complicated (both green). The three different line styles represent three different possible solutions (among many others). Moreover, each of the three depicted solutions has the same alternation (from top to bottom: Blue, Red, Blue, Red).

$(x, y) \in R$ and $(x' \leq x) \wedge (y' \leq y)$ implies $(x', y') \in R$. For every pair of two distinct vertices $u, v \in V(G)$, a Forest CSP instance may contain a downwards-closed relation $R_{u,v}$ and a constraint binding u and v asserting that $(u, v) \in R_{u,v}$. Such a constraint is henceforth called a *downwards-closed constraint*. Note that an intersection of two downwards-closed relations is again downwards-closed; thus it would not add more expressive power to the problem to allow multiple downwards-closed constraints between the same pair of variables.

Observe that if one for every $u \in V(G)$ adds a clone u', connected to u with an edge uu' with a segment reversion $\pi_{uu'}$ that reverses the whole domain, then with the four downwards closed constraints in $\{u, u'\} \times \{v, v'\}$ one can express any constraint as in the left panel of Figure 5.

This concludes the description of the Forest CSP problem that asks for a satisfying assignment to the input instance. We now argue that Forest CSP is fixed-parameter tractable when parameterized by the number of variables.

The algorithm for Forest CSP. As a preprocessing step, note that we can assume that no downwards-closed constraint binds two variables of the same component T. Indeed, if u and v is in the same component T and a constraint with a downwards-closed relation $R_{u,v}$ is present, then we can iterate over all n_T assignments to the variables of T and delete those that do not satisfy $R_{u,v}$. (Deleting a value from a domain requires some tedious renumbering of the domains, but does not lead us out of the Forest CSP class of instances.)

Similarly, we can assume that for every downwards-closed constraint binding u and v with relation $R_{u,v}$, for every possible value x of u, there is at least one satisfying value of v (and vice versa), as otherwise one can delete x from the domain of u and propagate.

First, guess whether there is a variable v such that setting $v = 1$ extends to a satisfying assignment. If yes, guess such v and simplify the instance, deleting the whole component of v and restricting the domains of other variables accordingly.

Second, guess whether there is an edge $uv \in E(G)$ such that there is a satisfying assignment where the value of u or the value of v is at the endpoint of a segment of the segment reversion π_{uv}. If this is the case, guess the edge uv, guess whether the value of u or v is at the endpoint, and guess whether it is the left or right endpoint of the segment. Restrict the domains according to the guess: If, say, we have guessed that the value of u is at the right endpoint of a segment of π_{uv}, restrict the domain of u to only the right endpoints of segments of π_{uv} and propagate the restriction through the whole component of u. The crucial observation now is that, due to this step, π_{uv} becomes an identity permutation. Thus, we can contract the edge uv,

reducing the number of variables by one.

In the remaining case, we assume that for every satisfying assignment, no variable is assigned 1 and no variable is assigned a value that is an endpoint of a segment of an incident segment reversion constraint. Pick a variable a and look at a satisfying assignment ϕ that minimizes $\phi(a)$. Try changing the value of a to $\phi(a)-1$ (which belongs to the domain, as $\phi(a) \neq 1$) and propagate it through the component T containing a. Observe that the assumption that no value is at the endpoint of a segment of an incident segment reversion implies that for every $b \in T$, the value of b changes from $\phi(b)$ to either $\phi(b)+1$ or $\phi(b)-1$.

By the minimality of ϕ, some constraint is not satisfied if we change the value of a to $\phi(a)-1$ and propagate it through T. This violated constraint has to be a downwards-closed constraint binding u and v with relation $R_{u,v}$ where $|\{u,v\} \cap T| = 1$. Without loss of generality, assume $v \in T$ and $u \notin T$. Furthermore, to violate a downwards-closed constraint, the change to the value of v has to be from $\phi(v)$ to $\phi(v)+1$.

Let S be the component of u. Define a function $f : [n_S] \to [n_T]$ as $f(x) = \max\{y \in [n_T] \mid (x,y) \in R_{u,v}\}$. Note that $\phi(v) = f(\phi(u))$, as with u set to $\phi(u)$, the value $\phi(v)$ for v satisfies $R_{u,v}$ while the value $\phi(v)+1$ violates $R_{u,v}$. Thus, the value of v (and, by propagation, the values in the entire component T) are a function of the value of u (i.e., the value of the component S). Hence, in some sense, by guessing the violated constraint $R_{u,v}$ we have reduced the number of components of G (i.e., the number of super-variables).

However, adding a constraint "$\phi(v) = f(\phi(u))$" leaves us outside of the class of Forest CSPs. Luckily, this can be easily but tediously fixed. Thanks to the fact that f is a non-increasing function, one can bind u and v with a segment-reversion constraint that reverses the whole $[n_S]$, replace the domains of all nodes of T with $[n_S]$, and re-engineer all constraints binding the nodes of T to the new domain.

Hence, in the last branch, after guessing the violated downwards-closed constraint $R_{u,v}$, we have reduced the number of components of G by one. This finishes the sketch of the fixed-parameter algorithm for Forest CSP.

Reduction to Forest CSP. Let us now go back to the problematic constraints and sketch how to cast them into the Forest CSP setting. Recall Figure 4. We discussed that a fixed position of the left vertical line at x_1 fixes a partition of the red points to the right of this line into blocks. If we want to keep the striped area between x_1, y_4, and y_5 empty, we can add two constraints: one between x_1 and y_4, keeping, for a fixed position of x_1, the line y_4 above the red block below it, and one between x_1 and y_5, keeping, again for a fixed position of x_1, the line y_5 below the red block above it.

To simplify these constraints we perform an additional guessing step. In Figure 4, the rightmost red point in every block is the *leader* of the block and, similarly, the leftmost blue point in every block is the *leader* of the block. Since the position of x_1 fixes the red blocks via the guessed alternation, it also fixes the red leaders. Furthermore, since, for a fixed position of x_1, varying x_2 only increases or decreases the blue blocks without merging or splitting them, the position of x_1 also determines the leaders of the blue blocks. In the branching step, we guess the left-to-right order of the red leaders and the left-to-right order of the blue leaders, deleting from the domains of x_1 and x_2 positions, where the order is different than guessed.

To understand what this branching step gives us, let us move to the larger example in Figure 7. We think of the left vertical line — denoted in the figure as p_1 — as sliding continuously right-to-left from position x_1' to position x_1. As we slide, the set of red points to the right of the line grows. To keep the alternation as guessed, the next vertical line (denoted p_2) slides as well, on the way shrinking the set of blue points between p_1 and p_2.

During this slide, a blue block may disappear, like the third or fourth block with yellow background in Figure 7. A disappearing blue block merges the two neighboring red blocks into one. To keep the alternation as guessed, a new red block needs to appear (e.g., the second and third block with green background in Figure 7) at the same time splitting a blue block into two smaller blocks.

The important observation is as follows: Since the order of the leaders is as guessed, the first blue block to disappear is the block with the rightmost blue leader, and, in general, the order of disappearance of blue blocks is exactly the right-to-left order of their leaders. Naturally, not all blue blocks need to disappear, but only those that have their leaders between the two considered positions of p_2.

Similarly, if during the move s blue blocks disappear, then the number of red blocks also decreased by s and exactly s new red blocks need to appear. These newly appearing blocks will be exactly the s blocks with the leftmost leaders.

Consequently, for every possible scenario as in Figure 7 with lines p_1 and p_2 (but different choices of x_1 and x_1'), the same blocks will start to merge and appear, only the number of the merged/appearing blocks s can differ. In Figure 7, the third block with brown background gets first merged into the fourth block with brown background and then the second block with brown background gets merged into the resulting block,

ending up with the fourth block with green background. The above description is general: Whenever we start with the line p_1, if the alternation and order of leaders is as we guessed, then sliding p_1 to the right will first merge the third red block into the fourth (and, also merge the sixth into the fifth) and then the second into the resulting block.

This merging order allows us to define a rooted auxiliary tree on the red blocks; in this tree, a child block is supposed to merge to the parent block. Figure 7 depicts an exemplary such tree.

The root B of the auxiliary tree is the block with the rightmost leader; it never gets merged into another block, its leader stays constant, and the block B only absorbs other blocks. Thus, the y-coordinate of its top border only increases and the y-coordinate of its bottom border only decreases as one moves p_1 from right to left.

Consider now a child B' of the root block B and assume B' is above B. As one moves p_1 from right to left, once in a while B' gets merged into B and a new block B' appears. As B only grows, the new appearance of B' is always above the previous one. Meanwhile, between the moments when B' is merged into B, B' grows, but its leader stays constant. Hence, between the merges the y-coordinate of the bottom border of B' decreases, only to jump and increase during each merge.

As one goes deeper in the auxiliary tree, the above behavior can nest. Consider for example a child B'' of B' that is below B', that is, is between B' and the root B. When B' is merged with B, B'' is merged as well and B'' jumps upwards to a new position together with B'. However, between the merges of B and B', B'' can merge multiple times with B'; every such merge results in B'' jumping — this time downwards — to a new position. If one looks at the y-coordinate of the top border of B'', then:

- between the merges of B' and B'', it increases;
- during every merge of B' and B'', it decreases;
- during every merge of B and B', it increases.

The above reasoning can be made formal into the following: If a block is at depth d in the auxiliary tree, then the y-coordinate of its top or bottom border, as a function of the position of p_1, can be expressed as a composition of a nondecreasing function and $d + \mathcal{O}(1)$ segment reversions. This is the way we cast the leftover difficult constraints into the Forest CSP world.

Final remark. We conclude this overview with one remark. As mentioned, in [20, Section 4] by a series of color-coding and branching steps we reduce Optimal Discretization to an instance of Forest CSP. It may be tempting to backwards-engineer the algorithm for Forest CSP back to the setting of Optimal Discretization. However, we think that this is a dead end; in particular, the second branching step, when one contracts an edge, merging two variables, seems to have no good analog in the Optimal Discretization setting. Furthermore, we think that an important conceptual contribution of this work is the isolation of the Forest CSP problem as an island of tractability behind the tractability of Optimal Discretization.

3 Segments, segment reversions, and segment representations

We now give the basic definitions that we need for presenting the algorithm for the auxiliary CSP problem.

3.1 Basic definitions and observations

Definition 1. *For a finite totally ordered set $(D, \leq)$ and two elements $x, y \in D$, $x \leq y$, the* segment between x and y *is $D[x, y] = \{z \in D \mid x \leq z \leq y\}$. Elements x and y are the* endpoints *of the segment $D[x, y]$.*

We often write just $[x, y]$ for the segment $D[x, y]$ if the set $(D, \leq)$ is clear from the context.

Definition 2. *Let $(D, \leq)$ be a finite totally ordered set and let $D = \{a(1), a(2), \ldots, a(|D|)\}$ with $a(i) < a(j)$ if and only if $i < j$.*

A permutation $\pi : D \to D$ is a segment reversion *of D if there exist integers $1 = i_1 < i_2 < \ldots < i_\ell = |D| + 1$ such that for every $j \in [\ell]$ and every integer x with $i_j \leq x < i_{j+1}$ we have $\pi(a(x)) = a(i_{j+1} - 1 - (x - i_j))$. In other words, a segment reversion is a permutation that partitions the domain D into segments $[a(i_1), a(i_2 - 1)], [a(i_2), a(i_3 - 1)], \ldots, [a(i_\ell), a(i_\ell - 1)]$ and reverses every segment independently.*

A segment representation of depth k *of a permutation π of D is a sequence of k segment reversions $\pi_1, \pi_2, \ldots, \pi_k$ of D such that their composition satisfies $\pi = \pi_k \circ \pi_{k-1} \circ \ldots \circ \pi_1$. A permutation $\pi : D \to D$ is of* depth *at most k if π admits a segment representation of depth at most k.*

A segment representation of depth k *of a function $\phi : D \to \mathbb{N}$ is a tuple of k segment reversions $\pi_1, \pi_2, \ldots, \pi_k$ of D and a nondecreasing function ϕ' such that their composition satisfies $\phi = \phi' \circ \pi_1 \circ \pi_2 \circ \ldots \circ \pi_k$.*

Definition 3. *Let $(D, \leq)$ be a finite totally ordered set. A* segment partition *is a family $\mathcal{P}$ of segments of $(D, \leq)$ which is a partition of D. If for two segment partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ we have that for every $P_1 \in \mathcal{P}_1$ there exists $P_2 \in \mathcal{P}_2$ with $P_1 \subseteq P_2$ then we say that* $\mathcal{P}_1$ is more refined than $\mathcal{P}_2$ *or* $\mathcal{P}_2$ is coarser than $\mathcal{P}_1$. *The notion of a coarser partition turns the family of all segment partitions into a partially ordered set with*

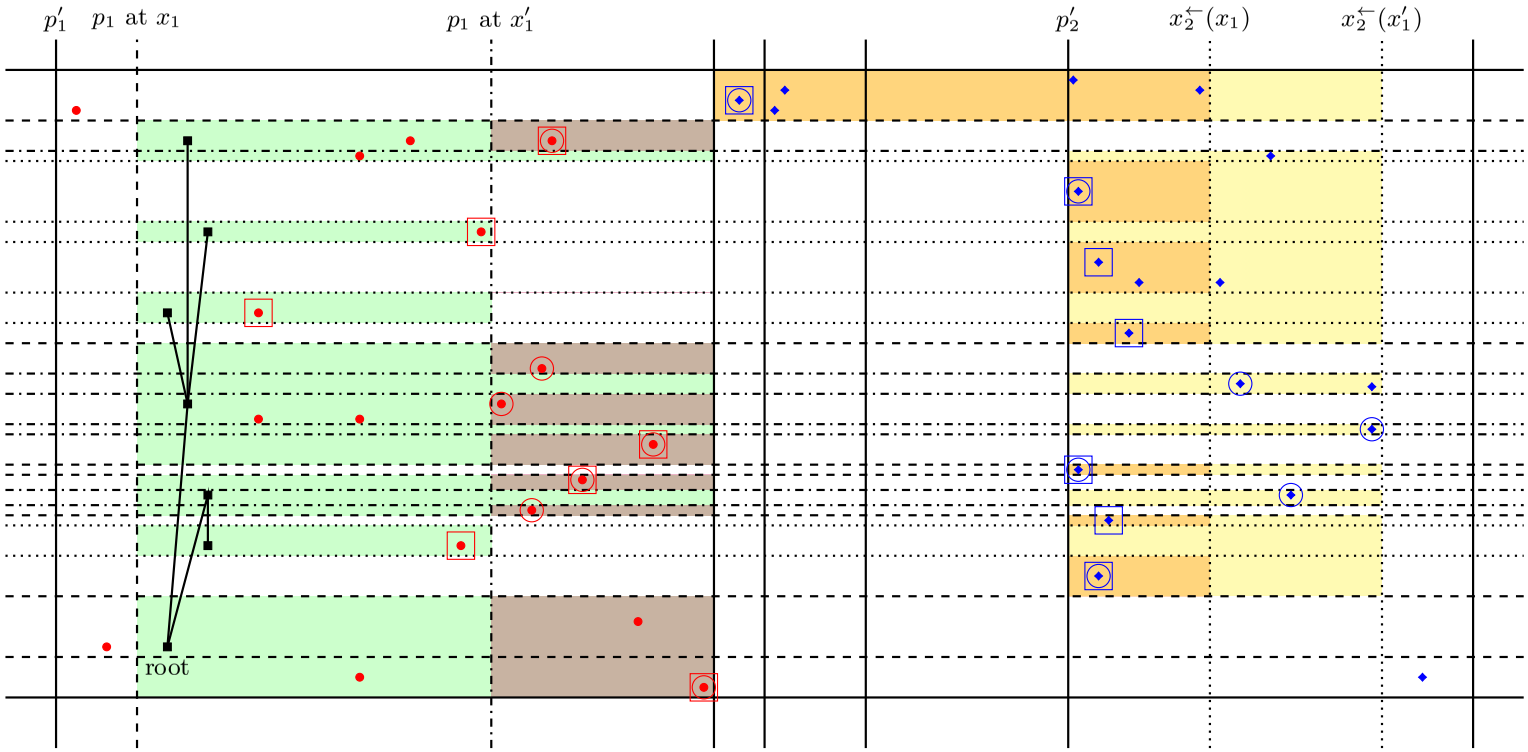

Figure 7: More complex example of, where vertical p_1 is either at position x_1 or at position x_1' for $x_1 < x_1'$. The horizontal lines for x_1 are denoted with dashed and dotted lines. The horizontal lines for x_1' are denoted with dashed and dash-dotted lines. Blocks given by positioning p_1 at x_1' are depicted by brown color with circled leaders and blocks given by positioning p_1 at x_1 are depicted by the union of green and brown color with squared leaders. Let $x_2^{\leftarrow}(x_1)$ and $x_2^{\leftarrow}(x_1')$ denote the position of the next vertical line p_2 that gives the correct alternation if p_1 is placed at x_1 and x_1', respectively. Blocks given by positioning p_2 at $x_2^{\leftarrow}(x_1)$ are depicted by orange color with squared leaders and blocks given by positioning p_1 at $x_2^{\leftarrow}(x_1')$ are depicted by the union of yellow and orange color with circled leaders. An auxiliary rooted tree T for red blocks is also visualized. There, blocks B, B', and B'' are marked according to Section 2—Reduction to Forest CSP part.

two extremal values, the most coarse *partition with one segment and the* most refined *partition with all segments being singletons.*

Note that every segment partition $\mathcal{P}$ induces a segment reversion that reverses the segments of $\mathcal{P}$. We will denote this segment reversion as $g_{\mathcal{P}}$.

DEFINITION 4. *Let $(D_i, \leq_i)$ for $i = 1, 2$ be two finite totally ordered sets.*

A relation $R \subseteq D_1 \times D_2$ is downwards-closed *if for every $(a, b) \in R$ and $a' \leq_1 a$, $b' \leq_2 b$ it holds that $(a', b') \in R$.*

A relation $R \subseteq D_1 \times D_2$ is of depth at most *k if there exists a permutation π_1 of D_1 of depth at most k_1, a permutation π_2 of D_2 of depth at most k_2, and a downwards-closed relation $R' \subseteq D_1 \times D_2$ such that $k_1 + k_2 \leq k$ and $(a, b) \in R$ if and only if $(f_1(a), f_2(b)) \in R'$. A* segment representation *of R consists of R', a segment representation of π_1 of depth at most k_1 and a segment representation of π_2 of depth at most k_2.*

We make two straightforward observations regarding some relations that are of small depth.

OBSERVATION 1. *Let $(D_1, \leq_1)$ and $(D_2, \leq_2)$ be two finite totally ordered sets. For $i = 1, 2$, let $(a_i^j)_{j=1}^{\ell}$ be a sequence of elements of D_i. Then a relation $R \subseteq D_1 \times D_2$ defined as $(x_1, x_2) \in R$ if and only if:*

- *$\bigwedge_{j=1}^{\ell}(x_1 \leq_1 a_1^j) \vee (x_2 \leq_2 a_2^j)$ is downwards-closed and thus of depth 0;*
- *$\bigwedge_{j=1}^{\ell}(x_1 \leq_1 a_1^j) \vee (x_2 \geq_2 a_2^j)$ is of depth 1, using $k_1 = 0$ and $k_2 = 1$ and a segment reversion with one segment reversing the whole D_2;*
- *$\bigwedge_{j=1}^{\ell}(x_1 \geq_1 a_1^j) \vee (x_2 \leq_2 a_2^j)$ is of depth 1, using $k_1 = 1$ and $k_2 = 0$ and a segment reversion with one segment reversing the whole D_1;*
- *$\bigwedge_{j=1}^{\ell}(x_1 \geq_1 a_1^j) \vee (x_2 \geq_2 a_2^j)$ is of depth 2, using $k_1 = 1$ and $k_2 = 1$ and segment reversions each with one segment reversing the whole D_1 and the whole D_2, respectively.*

Thus, a conjunction of an arbitrary finite number of the above relations can be expressed as a conjunction of at most four relations, each of depth at most 2.

OBSERVATION 2. *Let $D_1, D_2 \subseteq D$ for a totally ordered set $(D, \leq)$. We treat D_i as a totally ordered set with the order inherited from $(D, \leq)$. Then a relation $R \subseteq D_1 \times D_2$ defined as $R = \{(x_1, x_2) \in D_1 \times D_2 \mid x_1 < x_2\}$ is of depth at most 1 and a segment representation of this depth can be computed in polynomial time.*[2]

[2]Throughout, for some relation $\leq$ we use $x < y$ to denote $x \leq y$ and not $x = y$.

Proof. Let π_2 be a segment reversion of D_2 with one segment, that is, π_2 reverses the domain D_2. Observe that $\{(a, \pi_2(b)) \mid a \in D_1 \wedge b \in D_2 \wedge a < b\}$ is a downwards-closed subrelation of $D_1 \times D_2$. ∎

3.2 Operating on segment representations

We will need the following two technical lemmas.

LEMMA 3.1. *Let $(D_1, \leq_1)$ and $(D_2, \leq_2)$ be two finite totally ordered sets, $f : D_1 \to D_2$ be a nondecreasing function*[3]*, and $g : D_2 \to D_2$ be a segment reversion. Then there exists a nondecreasing function $f' : D_1 \to D_2$ and a segment reversion $g' : D_1 \to D_1$ such that $g \circ f = f' \circ g'$. Furthermore, such f' and g' can be computed in polynomial time, given $(D_1, \leq_1)$, $(D_2, \leq_2)$, f, and g.*

Proof. Let $(D_2[a_i, b_i])_{i=1}^{r}$ be the segments of the segment reversion g in increasing order. For every $i \in \{1, 2, \ldots, r\}$, let

$$c_i = \min\{c \in D_1 \mid f(c) \geq a_i\},$$
$$d_i = \max\{d \in D_1 \mid f(d) \leq b_i\}.$$

Let $\mathcal{Q}$ be the family of those segments $D_1[c_i, d_i]$ for which both c_i and d_i are defined and $c_i \leq_1 d_i$ (which is equivalent to the existence of $x \in D_1$ with $f(x) \in D_2[a_i, b_i]$). From the definition of c_is and d_is we obtain that $\mathcal{Q}$ is a segment partition of $(D_1, \leq_1)$. We put $g' = g_{\mathcal{Q}}$ and

$$f' = g \circ f \circ g'.$$

The desired equation $g \circ f = f' \circ g'$ follows directly from the definition of f' and the fact that the segment reversion g' is an involution.[4] Clearly, f' and g' are computable in polynomial time. It remains to check that f' is nondecreasing.

Let $x <_1 y$ be two elements of D_1. We consider two cases. In the first case, we assume that x and y belong to the same segment $D_1[c_i, d_i]$ of $\mathcal{Q}$. Then, $g'(x)$ and $g'(y)$ also lie in $D_1[c_i, d_i]$ and $g'(x) >_1 g'(y)$ by the definition of the segment reversion $g' = g_{\mathcal{Q}}$. Since f is nondecreasing, $f(g'(x)) \geq_2 f(g'(y))$. By the definition of c_i and d_i, we have that both $f(g'(x))$ and $f(g'(y))$ lie in the segment $D_2[a_i, b_i]$. Hence, since $D_2[a_i, b_i]$ is a segment of the segment reversion g, we have $g(f(g'(x))) \leq_2 g(f(g'(y)))$, as desired.

In the second case, let $x \in D_1[c_i, d_i]$ and $y \in D_1[c_j, d_j]$ for some $i \neq j$. From the definition of the

[3]A function f on a domain and codomain that are totally ordered by $\leq_1$ and $\leq_2$, respectively, is called *nondecreasing* if for every x, x' in the domain we have that $x \leq_1 x'$ implies $f(x) \leq_2 f(x')$.

[4]An *involution* is a function ϕ which is its own inverse, that is, $\phi \circ \phi$ is the identity.

c_is and d_is we infer that $x <_1 y$ implies $i < j$. By the definition of $g' = g_{\mathcal{Q}}$, we have $g'(x) \in D_1[c_i, d_i]$ and $g'(y) \in D_1[c_j, d_j]$. Since f is nondecreasing, $f(g'(x)) \leq_2 f(g'(y))$. By the definition of the c_is and d_is, we have that $f(g'(x)) \in D_2[a_i, b_i]$ and $f(g'(y)) \in D_2[a_j, b_j]$. Since $D_2[a_i, b_i]$ and $D_2[a_j, b_j]$ are segments of g, we have $g(f(g'(x))) \leq_2 g(f(g'(y)))$, as desired.

This finishes the proof that f' is nondecreasing and concludes the proof of the claim. ∎

LEMMA 3.2. *Let $(D_i, \leq_i)$ for $i = 1, 2, 3$ be three finite totally ordered sets, $f : D_1 \to D_2$ be a nondecreasing function, and $R \subseteq D_2 \times D_3$ be a downwards-closed relation. Then the relation*

$$R' = \{(x, y) \in D_1 \times D_3 \mid (f(x), y) \in R\}$$

is also downwards-closed.

Proof. If $(x, y) \in R'$, $x' \leq_1 x$, and $y' \leq_2 y$, then $f(x') \leq_2 f(x)$ as f is nondecreasing, $(f(x'), y') \in R$ as $(f(x), y) \in R$ and R is downwards closed, and thus $(x', y') \in R'$ by the definition of R'. ∎

4 Auxiliary CSP

In this section we will be interested in checking the satisfiability of the following constraint satisfaction problem (CSP).

DEFINITION 5. *An* auxiliary CSP instance *is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ consisting of a set $\mathcal{X} = \{x_1, x_2, \ldots, x_k\}$ of k* variables*, a totally ordered finite* domain *$(D_i, \leq_i) \in \mathcal{D}$ for every variable x_i, and a set $\mathcal{C}$ of binary* constraints*. Each constraint $C \in \mathcal{C}$ is a tuple $(x_{i(C,1)}, x_{i(C,2)}, R_C)$ consisting of two variables $x_{i(C,1)}$ and $x_{i(C,2)}$, and a relation $R_C \subseteq D_{i(C,1)} \times D_{i(C,2)}$ given as a segment representation of some depth. We say that constraint C* binds *$x_{i(C,1)}$ and $x_{i(C,2)}$. An* assignment *is a function $\phi\colon \mathcal{X} \to \mathcal{D}$ such that for each $x_i \in \mathcal{X}$ we have $\phi(x_i) \in D_i$. An assignment ϕ is* satisfying *if for each constraint $C = (x_{i(C,1)}, x_{i(C,2)}, R_C) \in \mathcal{C}$ we have $(\phi(x_{i(C,1)}), \phi(x_{i(C,2)})) \in R_C$.*

Qualitatively, the main result of this section is the following.

THEOREM 4.1. *Checking satisfiability of an auxiliary CSP instance is fixed-parameter tractable when parameterized by the sum of the number of variables, the number of constraints, and the depths of all segment representations of constraints.*

To prove Theorem 4.1 we show a more general result stated in Lemma 4.1 below. For this and to state precisely the running time bounds of the obtained algorithm, we need a few extra definitions. For a forest F, $\mathsf{trees}(F)$ is the family of trees (connected components) of F. For $y \in V(F)$, $\mathsf{tree}_F(y)$ is the tree of F that contains y. We omit the subscript if it is clear from the context.

DEFINITION 6. *A* forest-CSP instance *is a tuple consisting of a forest F with its vertex set $V(F)$ being the set of* variables *of the instance, an ordered finite* domain *$(D_T, \leq_T)$ for every $T \in \mathsf{trees}(F)$ (that is, one domain shared between all vertices of T), for every $e \in E(T)$ and $T \in \mathsf{trees}(F)$ a segment reversion g_e that is a segment reversion of D_T, and a family of* constraints *$\mathcal{C}$. Each constraint $C \in \mathcal{C}$ is a tuple (y_1, y_2, R_C) where $y_1, y_2 \in V(F)$ are variables and $R_C \subseteq D_{\mathsf{tree}(y_1)} \times D_{\mathsf{tree}(y_2)}$ is a downwards-closed relation. We say that C* binds *y_1 and y_2.*

An assignment *is a function $\phi\colon V(F) \to \mathcal{D}$ such that for each $y \in V(F)$ we have $\phi(y) \in D_{\mathsf{tree}(y)}$. An assignment ϕ* satisfies *the forest-CSP instance if for every edge $yy' \in E(F)$ we have $g_e(\phi(y)) = \phi(y')$ and for every constraint $C = (y_1, y_2, R_C)$ we have $(\phi(y_1), \phi(y_2)) \in R_C$.*

The apparent size *of a forest-CSP instance is the sum of the number of variables, number of trees of F, and the number of constraints.*

We will show the following result.

LEMMA 4.1. *There exists an algorithm that, given a forest-CSP instance $\mathcal{I}$ of apparent size s, in $2^{\mathcal{O}(s \log s)}|\mathcal{I}|^{\mathcal{O}(1)}$ time computes a satisfying assignment of $\mathcal{I}$ or correctly concludes that $\mathcal{I}$ is unsatisfiable.*

To see that Lemma 4.1 implies Theorem 4.1, we translate an auxiliary CSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ with k variables into an equivalent forest-CSP instance $(F, \mathcal{D}', \mathcal{C}')$. Start with $\mathcal{D} = \emptyset$, $\mathcal{C}' = \emptyset$, and a forest F consisting of k components $T_1, T_2, \ldots, T_k$ where T_i is an isolated vertex $x_i \in \mathcal{X}$. Define the domain $(D_{T_i}, \leq_{T_i}) \in \mathcal{D}'$ of tree T_i as $(D_{T_i}, \leq_{T_i}) := (D_i, \leq_i) \in \mathcal{D}$. Recall that for every constraint $C = (x_{i(C,1)}, x_{i(C,2)}, R_C) \in \mathcal{C}$ there is a segment representation, that is, there are $\ell_1, \ell_2 \in \mathbb{N}$, segment reversions $g_1^1, g_1^2, \ldots, g_1^{\ell_1}$ and $g_2^1, g_2^2, \ldots, g_2^{\ell_2}$, and a downwards-closed relation R'_C such that

$$(a_1, a_2) \in R_C \Leftrightarrow (g_1^{k_1} \circ g_1^{k_1-1} \circ \ldots \circ g_1^1(a_1), \\ g_2^{k_2} \circ g_2^{k_2-1} \circ \ldots \circ g_2^1(a_2)) \in R'_C).$$

For each constraint $C \in \mathcal{C}$ as above, proceed as follows:

1. For both $j = 1, 2$ attach to $x_{i(C,j)}$ in the tree $T_{i(C,j)}$ a path of length k_j with vertices $x_{i(C,j)} = y_j^0, y_j^1, \ldots, y_j^{k_j}$, wherein $y_j^1, \ldots, y_j^{k_j}$ are new variables, and label the each edge $y_j^{i-1} y_j^i$ with the segment reversion g_j^i.

2. Add a constraint $C' = (y_1^{k_1}, y_2^{k_2}, R'_C)$ to $\mathcal{C}'$.

A direct check shows that a natural extension of a satisfying assignment to the input auxiliary CSP instance $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ satisfies the resulting forest-CSP instance $(F, \mathcal{D}', \mathcal{C}')$ and, in the other direction, a restriction to $\{x_1, x_2, \ldots, x_k\}$ of any satisfying assignment to $(F, \mathcal{D}', \mathcal{C}')$ is a satisfying assignment to $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Furthermore, if the input auxiliary CSP instance has k variables, c constraints, and p is the sum of the depths of all segment representations, then the apparent size of the resulting forest-CSP instance is $p + 2k + c$. Thus, Theorem 4.1 follows from Lemma 4.1.

The rest of this section is devoted to the proof of Lemma 4.1.

4.1 Fixed-parameter algorithm for forest CSPs

In what follows, *to solve* a forest-CSP instance means to check its satisfiability and, in case of a satisfiable instance, produce one satisfying assignment. The algorithm for Lemma 4.1 is a branching algorithm that at every recursive call performs a number of preprocessing steps and then branches into a number of subcases. Every recursive call will be performed in polynomial time and will lead to a number of subcalls that is polynomial in s. Every recursive call will be given a forest-CSP instance $\mathcal{I}$ and will either solve $\mathcal{I}$ directly or produce forest-CSP instances and pass them to recursive subcalls while ensuring that (i) the input instance $\mathcal{I}$ is satisfiable if and only if one of the instances passed to the recursive subcalls is satisfiable, and (ii) given a satisfying assignment of an instance passed to a recursive subcall, one can produce a satisfying assignment to $\mathcal{I}$ in polynomial time. In that case, we say that the recursive call is *correct*. In every recursive subcall the apparent size s will decrease by at least one, bounding the depth of the recursion by s. In that case, we say that the recursive call is *diminishing*. Observe that these two properties guarantee the correctness of the algorithm and the running time bound of Lemma 4.1.

We will often phrase a branching step of a recursive algorithm as *guessing* a property of a hypothetical satisfying assignment. Formally, at each such step, the algorithm checks all possibilities iteratively.

It will be convenient to assume that every domain $(D_T, \leq_T)$ equals $\{1, 2, \ldots, |D_T|\}$ with the order $\leq_T$ inherited from the integers. (This assumption can be reached by a simple remapping argument and we will maintain it throughout the algorithm.) Thus, henceforth we always use the integer order $<$ for the domains.

Let us now focus on a single recursive call. Assume that we are given a forest-CSP instance

$$\mathcal{I} = (F, (D_T)_{T \in \mathsf{trees}(F)}, (g_e)_{e \in E(F)}, \mathcal{C})$$

of size s. For two nodes $y, y' \in V(F)$ in the same tree T of F, we denote

$$g_{y \to y'} = g_{e_r} \circ g_{e_{r-1}} \circ \ldots \circ g_{e_1},$$

where $e_1, e_2, \ldots, e_r$ is the unique path from y to y' in T. Thus, if ϕ is a satisfying assignment, then $\phi(y') = g_{y \to y'}(\phi(y))$. (And, moreover, $\phi(y) = g_{y' \to y}(\phi(y'))$ since each segment reversion g_e satisfies $g_e = g_e^{-1}$.) In other words, a fixed value of one variable in a tree T fixes the values of all variables in that tree. Thus, there are $|D_T|$ possible assignments of all variables of a tree T and we can enumerate them in time $\mathcal{O}(|T| \cdot |D_T|)$. We need the following auxiliary operations.

Forbidding a value. We define the operation of *forbidding value* $a \in D_{\mathsf{tree}(y)}$ *for variable* $y \in V(F)$ as follows. Let $T = \mathsf{tree}(y)$. Intuitively, we would like to delete a from the domain of y and propagate this deletion to all $y' \in V(T)$ and constraints binding variables of T. Formally, we let $D'_T = \{1, 2, \ldots, |D_T| - 1\}$. For every $y' \in V(T)$, we define $\alpha_{y'} : D'_T \to D_T$ as $\alpha_{y'}(b) = b$ if $b < g_{y \to y'}(a)$ and $\alpha_{y'}(b) = b + 1$ if $b \geq g_{y \to y'}(a)$. In every constraint $C = (y_1, y_2, R_C)$ and $j \in \{1, 2\}$, if $y_j \in V(T)$, then we replace R_C with R'_C defined as follows,

$$R'_C = \{(x_1, x_2) \in D'_T \times D_{\mathsf{tree}(y_2)} \mid (\alpha_{y_1}(x_1), x_2) \in R_C\} \quad \text{if } j = 1,$$
$$R'_C = \{(x_1, x_2) \in D_{\mathsf{tree}(y_1)} \times D'_T \mid (x_1, \alpha_{y_1}(x_2)) \in R_C\} \quad \text{if } j = 2.$$

(Note that y_1 and y_2 are not necessarily in different trees.) Observe that each domain remains of the form $\{0, 1, \ldots, \ell\}$ for some $\ell \in \mathbb{N}$. It is straightforward to verify that R'_C is downwards-closed as R_C is downwards-closed. Furthermore, a direct check shows that:

1. If ϕ is a satisfying assignment to the original instance such that $\phi(y) \neq a$, then $\phi(y') \neq g_{y \to y'}(a)$ for every $y' \in V(T)$. Moreover, the assignment ϕ' defined as $\phi'(y') = \alpha_{y'}^{-1}(\phi(y'))$ for every $y' \in V(T)$ and $\phi'(y') = \phi(y')$ for every $y' \in V(F) \setminus V(T)$ is a satisfying assignment to the resulting instance.
2. If ϕ' is a satisfying assignment to the resulting instance, then ϕ defined as $\phi(y') = \alpha_{y'}(\phi'(y'))$ for every $y' \in V(T)$, and $\phi(y') = \phi'(y')$ for every $y' \in V(F) \setminus V(T)$ is a satisfying assignment to the original instance.

Restricting the domain D_T *of a variable* $y \in V(T)$ *to* $A \subseteq D_T$ means forbidding all values of $D_T \setminus A$ for y.

We now describe the steps performed in the recursive call and argue in parallel that the recursive call is correct and diminishing.

Preprocessing steps. We perform the following preprocessing steps exhaustively.

1. If there are either no variables (hence a trivial empty satisfying assignment) or a variable with an empty domain (hence an obvious negative answer), solve the instance directly.

   Thus, henceforth we assume $V(F) \neq \emptyset$ and that every domain is nonempty.

2. For every constraint C that binds two variables from the same tree T, we iterate over all $|D_T|$ possible assignments of all variables in T and forbid those that do not satisfy C. (Recall that fixing the value of one variable of a tree fixes the values of all other variables of that tree.) Finally, we delete C.

   Thus, henceforth we assume that every constraint binds variables from two distinct trees of F.

3. For every constraint C, for both variables y_j, $j = 1, 2$, that are bound by C, and for every $a \in D_{\mathsf{tree}(y_j)}$, if there is no $b \in D_{\mathsf{tree}(y_{3-j})}$ such that (a, b) satisfies C, we forbid a for the variable y_j.

   Thus, henceforth we assume that for every constraint C, every variable it binds, and every possible value a of this variable, there is at least one value of the other variable bound by C that together with a satisfies C.

Clearly, the above preprocessing steps can be performed exhaustively in polynomial time and they do not increase the apparent size of the instance.

We next perform three branching steps. Ultimately, in each of the subcases we consider we will make a recursive call. However, the branching steps 1 and 2 both hand one subcase down for treatment in the later branching steps.

For every $T \in \mathsf{trees}(F)$, pick arbitrarily some node $x_T \in V(T)$. Assume that $\mathcal{I}$ is satisfiable and let ϕ be a satisfying assignment that is minimal in the following sense. For every $T \in \mathsf{trees}(F)$, we require that either $\phi(x_T) = 1$ or if we replace the value $\phi(x_T)$ with $\phi(x_T)-1$ and the value $\phi(y)$ with $g_{x_T \to y}(\phi(x_T) - 1)$ for every $y \in V(T)$, we violate some constraint. Note that if $\mathcal{I}$ is satisfiable then such an assignment exists, because each domain D_T has the form $\{1, 2, \ldots, |D_T|\}$ and thus $\phi(x_T) - 1, g_{x_T \to y}(\phi(x_T) - 1) \in D_T$.

First branching step. We branch into $1 + |\mathsf{trees}(T)| \leq s+1$ subcases, guessing whether there exists a tree T such that the variable x_T satisfies $\phi(x_T) = 1$ and which tree it is precisely. If we have guessed that no such tree exists, we proceed to the next steps of the algorithm with the assumption that $\phi(x_T) > 1$ for every $T \in \mathsf{trees}(F)$. The other subcases are labeled by the trees of F. In the subcase for $T \in \mathsf{trees}(F)$, we guess that $\phi(x_T) = 1$. For every constraint $C = (y_1, y_2, R_C)$ that binds $y_j \in V(T)$ with another variable $y_{3-j} \notin V(T)$, we restrict the domain $D_{\mathsf{tree}(y_{3-j})}$ of y_{3-j} to only values b such that $(g_{x_T \to y_j}(1), b) \in R_C$. Finally, we delete the tree T and all constraints binding variables of $V(T)$, and invoke a recursive call on the resulting instance.

To see that this step is diminishing, note that, due to the deletion of T, the apparent size in the recursive call is reduced by at least one. For correctness, clearly, if $\phi(x_T) = 1$, then the resulting instance is satisfiable and any satisfying assignment to the resulting instance can be extended to a satisfying assignment of $\mathcal{I}$ by assigning $g_{x_T \to y}(1)$ to y for every $y \in V(T)$.

Second branching step. We guess whether there exists an edge $yy' \in E(F)$ such that $\phi(y)$ is an endpoint of a segment of $g_{yy'}$. If we have guessed that no such edge yy' exists, we proceed to the next steps of the algorithm. Otherwise, we guess $yy' \in E(F)$, one endpoint y, and whether $\phi(y)$ is the left or the right endpoint of a segment of $g_{yy'}$, leading to at most $4|E(F)| \leq 4s$ subcases. (Note that $|E(F)| \leq |V(F)| \leq s$.) We restrict the domain $D_{\mathsf{tree}(y)}$ of y to only those values a such that a is the left/right (according to the guess) endpoint of a segment of $g_{yy'}$. Observe that now $g_{yy'}$ is an identity, as each of its segment has been reduced to a singleton. Consequently, we do not change the set of satisfying assignments if we contract the edge yy' in the tree $\mathsf{tree}(y)$ and, for every constraint binding y or y', modify C to bind instead the image of the contraction of the edge yy'. This decreases s by one and we pass the resulting instance to a recursive subcall.

Third branching step. Hence, we proceed to the last branching step with the case where no edge yy' as in branching step 2 exists. Recall that also from the first branching step we can assume that $\phi(x_T) > 1$ for every $T \in \mathsf{trees}(F)$. Pick an arbitrary tree $T \in \mathsf{trees}(F)$. Using the minimality of ϕ, we now guess which constraint $\Gamma = (y_1, y_2, R_\Gamma)$ is violated if we replace $\phi(x_T)$ with $\phi(x_T) - 1$ and $\phi(y)$ with $g_{x_T \to y}(\phi(x_T) - 1)$ for every $y \in V(T)$. By symmetry, assume $y_1 \in V(T)$. Since, due to preprocessing, every constraint binds variables of two distinct trees, $y_2 \notin V(T)$. Let $S = \mathsf{tree}(y_2)$. Note that we have at most s subcases in this branching step.

We now aim to show that assigning a value to y_1 fixes the value of y_2 via constraint Γ. Consequently, we will be able to remove Γ and merge the trees S and T,

resulting in a smaller forest-CSP instance, which we can solve recursively.

Recall that for every $a \in D_S$ there exists at least one $b \in D_T$ with $(b, a) \in R_\Gamma$, by preprocessing step 3. Since R_Γ is a downwards-closed relation, there exists a nonincreasing function $f' : D_S \to D_T$ such that

$$R_\Gamma = \{(b, a) \in D_T \times D_S \mid b \leq f'(a)\}.$$

The crucial observation is the following.

Claim 1. *Assume that ϕ exists and all guesses in the current recursive call have been made correctly. Then, $\phi(y_1) = f'(\phi(y_2))$.*

Proof. Since we made a correct guess at the second branching step, for every edge yy' on the path in T from x_T to y_1 (with y' closer than y to x_T), the value $\phi(y) = g_{x_T \to y}(\phi(x_T))$ is not an endpoint of $g_{yy'}$. Inductively from x_T to y_1, we infer that for every y on the path from y_1 to x_T we have that $\phi(y) = g_{x_T \to y}(\phi(x_T))$ and $g_{x_T \to y}(\phi(x_T) - 1)$ are two consecutive integers. In particular, $\phi(y_1)$ and $g_{x_T \to y_1}(\phi(x_T) - 1)$ are two consecutive integers.

By choice of Γ, we have $(g_{x_T \to y_1}(\phi(x_T) - 1), \phi(y_2)) \notin R_\Gamma$ but $(\phi(y_1), \phi(y_2)) \in R_\Gamma$. Since R_Γ is downwards-closed, this is only possible if $g_{x_T \to y_1}(\phi(x_T) - 1) = \phi(y_1) + 1$ and hence $\phi(y_1) = f'(\phi(y_2))$. This concludes the proof of the claim. ∎

Claim 1 implies that by fixing an assignment of the tree S, we induce an assignment of T via the function f'. We would like to merge the two trees S and T via an edge y_1y_2, labeled with f'. However, f' is not a segment reversion, but a nonincreasing function. Thus, we need to perform some work to get back to a forest-CSP instance representation. For this, we will leverage Lemma 3.1.

Let g° be a segment reversion with one segment, reversing the whole D_S. Let $f'' = f' \circ g^\circ$, that is, $f'' : D_S \to D_T$ and $f'' \circ g^\circ = f'$. Observe that since f' is nonincreasing, f'' is nondecreasing.

We perform the following operation on T that will result in defining segment reversions g'_e of D_T for every $e \in E(T)$ and nondecreasing functions $f_y : D_S \to D_T$ for every $y \in V(T)$ as follows. We temporarily root T at y_1. We initiate $f_{y_1} = f''$. Then, in a top-to-bottom manner, for every edge yy' between a node y and its parent y' such that $f_{y'}$ is already defined, we invoke Lemma 3.1 to $f_{y'}$ and the segment reversion $g_{yy'}$, obtaining a segment reversion $g'_{yy'}$ of D_S and a nondecreasing function $f_y : D_S \to D_T$ such that

$$g_{yy'} \circ f_{y'} = f_y \circ g'_{yy'}. \tag{4.1}$$

We merge the trees S and T into one tree T' by adding an edge y_1y_2 and define $g'_{y_1y_2} = g^\circ$. We set $D_{T'} = D_S$; observe that all g'_e for $e \in E(T)$ as well as $g'_{y_1y_2}$ are segment reversions of D_S. Let F' be the resulting forest. For every $e \in E(F) \setminus E(T)$, we define $g'_e = g_e$. Similarly as we defined $g_{y \to y'}$, we define $g'_{y \to y'}$ for every two vertices y, y' of the same tree of F' as $g'_{e_r} \circ g'_{e_{r-1}} \circ \ldots \circ g'_{e_1}$ where $e_1, e_2, \ldots, e_r$ are the edges on the path from y to y' in F'. Note that $g'_{y \to y'} = g_{y \to y'}$ when $y, y' \notin V(T')$ or $y, y' \in V(S)$.

We now define a modified set of constraints $\mathcal{C}'$ as follows. Every constraint $C \in \mathcal{C}$ that does not bind any variable of T we insert into $\mathcal{C}'$ without modifications. For every constraint $C \in \mathcal{C}$ that binds a variable of T, we proceed as follows. By symmetry, assume that $C = (z_1, z_2, R_C)$ with $z_1 \in V(T)$ and $z_2 \notin V(T)$. Recall that $R_C \subseteq D_T \times D_{\mathsf{tree}(z_2)}$ and $f_{z_1} : D_S \to D_T$. We apply Lemma 3.2 to R_C and f_{z_1}, obtaining a downwards-closed relation $R'_C \subseteq D_S \times D_{\mathsf{tree}(z_2)}$ such that

$$(a, b) \in R'_C \Leftrightarrow (f_{z_1}(a), b) \in R_C.$$

We insert $C' := (z_1, z_2, R'_C)$ into $\mathcal{C}'$.

Let $\mathcal{I}' = (F', (D_T)_{T \in \mathsf{trees}(F)}, (g'_e)_{e \in E(F')}, \mathcal{C}')$ be the resulting forest-CSP instance. Note that $|V(F')| \leq |V(F)|$, $|\mathcal{C}'| \leq |\mathcal{C}|$, while $|\mathsf{trees}(F')| < |\mathsf{trees}(F)|$. Thus, the apparent size of $\mathcal{I}'$ is smaller than the apparent size of $\mathcal{I}$. We pass $\mathcal{I}'$ to a recursive subcall.

To complete the proof of Lemma 4.1, it remains to show correctness of branching step 3. This is done in the next two claims.

Claim 2. *Let ζ' be a satisfying assignment to $\mathcal{I}'$. Define an assignment ζ to $\mathcal{I}$ as follows. For every $y \in V(F) \setminus V(T)$, set $\zeta(y) = \zeta'(y)$. For every $y \in V(T)$, set $\zeta(y) = f_y(\zeta'(y))$. Then ζ is a satisfying assignment to $\mathcal{I}$.*

Proof. To see that ζ is an assignment, that is, maps each variable into its domain, since every function f_y for $y \in V(T)$ has domain $D_S = D_{T'}$ and codomain D_T, every $y \in V(T)$ satisfies $\zeta(y) \in D_T$.

To see that ζ is a satisfying assignment, consider first the condition on the forest edges. Pick $e = yy' \in E(F)$. If $e \notin E(T)$, then $\zeta(y) = \zeta'(y)$, $\zeta(y') = \zeta'(y')$, $g_e = g'_e$, and obviously $\zeta(y') = g'_e(\zeta(y))$. Otherwise, assume without loss of generality that y' is closer than y to y_1 in T. Then (4.1) ensures that

$$\begin{aligned} g_{yy'}(\zeta(y')) &= g_{yy'}(f_{y'}(\zeta'(y'))) = f_y(g'_{yy'}(\zeta'(y')) \\ &= f_y(\zeta'(y)) = \zeta(y) \end{aligned}$$

as desired.

Now pick a constraint $C \in \mathcal{C}$ and let us show that ζ satisfies C. If C does not bind a variable of T, then

$C \in \mathcal{C}'$ and ζ and ζ' agree on the variables bound by C, hence ζ satisfies C. Otherwise, without loss of generality, $C = (z_1, z_2, R_C)$ with $z_1 \in V(T)$ and there is the corresponding constraint $C' = (z_1, z_2, R'_C)$ in $\mathcal{C}'$ as defined above. Since ζ' satisfies C', we have $(\zeta'(z_1), \zeta'(z_2)) \in R'_C$. By the definition of R'_C, this is equivalent to $(f_{z_1}(\zeta'(z_1)), \zeta'(z_1)) \in R_C$. Since $\zeta(z_1) = f_{z_1}(\zeta'(z_1))$ (as $z_1 \in V(T)$) and $\zeta(z_2) = \zeta'(z_2)$, this is equivalent to $(\zeta(z_1), \zeta(z_2)) \in R_C$. Hence, ζ satisfies the constraint C. This finishes the proof of the claim. ∎

Claim 3. *Let ζ be a satisfying assignment to $\mathcal{I}$ that additionally satisfies $\zeta(y_1) = f'(\zeta(y_2))$. Define an assignment ζ' to $\mathcal{I}'$ as follows. For every $y \in V(F) \setminus V(T)$, set $\zeta'(y) = \zeta(y)$. For every $y \in V(T)$, set $\zeta'(y) = g'_{y_2 \to y}(\zeta(y_2))$. Then ζ' is a satisfying assignment to $\mathcal{I}'$.*

Proof. To see that ζ' is indeed an assignment, it is immediate from the definition of $\mathcal{I}'$ that for every tree A of F' and $y \in V(A)$ we have $\zeta'(y) \in D_A$. To see that ζ' is a satisfying assignment, by definition, for every $e = yy' \in E(F')$ we have $\zeta'(y') = g'_e(\zeta'(y))$. Also, obviously ζ' satisfies all constraints of $\mathcal{C}'$ that come unmodified from a constraint of $\mathcal{C}$ that does not bind a variable of $V(T)$. It remains to show that the remaining constraints are satisfied.

Consider a constraint $C' = (z_1, z_2, R'_C) \in \mathcal{C}'$ that comes from a constraint $C = (z_1, z_2, R_C) \in \mathcal{C}$ binding a variable of $V(T)$. Without loss of generality, $z_1 \in V(T)$ and $z_2 \notin V(T)$. By composing (4.1) over all edges on the path from z_1 to y_1 in T we obtain that

$$g_{y_1 \to z_1} \circ f'' = f_{z_1} \circ g'_{y_1 \to z_1}.$$

By composing the above with g° on the right and using $f'' = f' \circ g^\circ$ (hence $f'' \circ g^\circ = f'$) and $g^\circ = g'_{y_1 y_2}$, we obtain that

$$g_{y_1 \to z_1} \circ f' = f_{z_1} \circ g'_{y_2 \to z_1}. \tag{4.2}$$

By the definition of R'_C, we have that $(\zeta'(z_1), \zeta'(z_2)) \in R'_C$ is equivalent to

$$(f_{z_1}(\zeta'(z_1)), \zeta'(z_2)) \in R_C.$$

By the definition of ζ', this is equivalent to

$$(f_{z_1} \circ g'_{y_2 \to z_1}(\zeta(y_2)), \zeta(z_2)) \in R_C.$$

By (4.2), this is equivalent to

$$(g_{y_1 \to z_1} \circ f'(\zeta(y_2)), \zeta(z_2)) \in R_C.$$

Since $f'(\zeta(y_2) = \zeta(y_1)$, this is equivalent to

$$(g_{y_1 \to z_1}(\zeta(y_1)), \zeta(z_2)) \in R_C.$$

By the definition of $g_{y_1 \to z_1}$, this is in turn equivalent to

$$(\zeta(z_1), \zeta(z_2)) \in R_C,$$

which follows as ζ satisfies C. This finishes the proof of the claim. ∎

Claims 2 and 3 show the correctness of the third branching step, concluding the proof of Lemma 4.1 and of Theorem 4.1.

5 Conclusions

We would like to conclude with a number of possible future research directions.

For the Optimal Discretization problem, the natural direction is to try to improve our running time bound $2^{\mathcal{O}(k^2 \log k)} n^{\mathcal{O}(1)}$. Improving the parametric factor to $2^{o(k^2)}$ seems very challenging, as it would require a significant paradigm shift: in our algorithm, even the most natural branching step — guessing the content of every cell of the solution — yields $2^{\Omega(k^2)}$ subcases. Note that a polynomial kernel for the problem would be an even stronger result, the trivial $n^{k+\mathcal{O}(1)}$-time algorithm for the problem, pipelined with the supposed kernel, gives a $2^{\mathcal{O}(k \log k)} + n^{\mathcal{O}(1)}$-time algorithm for Optimal Discretization. A different, but perhaps more accessible, direction would be to analyze and optimize the factor $n^{\mathcal{O}(1)}$ in the running time bound.

In a larger perspective, the key to our tractability result is the tractability of the Forest CSP problem. How large is this isle of tractability in the CSP world? That is, consider the family of binary CSPs with the number of variables as a parameter. Without any restriction on the allowed constraints, this problem is exactly Multicolored Clique, the most popular starting point of W[1]-hardness reductions. Restricting the allowed constraints to Forest CSP makes the problem fixed-parameter tractable (but still NP-hard, as any permutation can be encoded as a composition of sufficiently many segment reversions), while reducing the allowed constraints to only conjunctions of clauses of the form $(x < a) \vee (y > b)$ makes the problem polynomial-time solvable. Can we further relax the restrictions on the constraints in Forest CSP while maintaining fixed-parameter tractability and where is the boundary of tractability of the resulting class of problems?

Acknowledgments. Marcin Pilipczuk would like to thank Brian Lavallee and Blair Sullivan for introducing him to the problem and sharing some initial thoughts during his visit at NCSU in February 2018.

We acknowledge the comfortable and inspiring atmosphere of Dagstuhl Seminar 19271, where the details of the algorithm for Forest CSP have been discussed.

References

[1] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys*, 30(4):412–458, 1998.

[2] A. Agrawal, K. V. K. Knudsen, D. Lokshtanov, S. Saurabh, and M. Zehavi. The parameterized complexity of guarding almost convex polygons. In *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164 of *LIPIcs*, pages 3:1–3:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[3] P. Ashok, F. V. Fomin, S. Kolay, S. Saurabh, and M. Zehavi. Exact algorithms for terrain guarding. *ACM Transactions on Algorithms*, 14(2):25:1–25:20, 2018.

[4] A. Banik, F. Panolan, V. Raman, V. Sahlot, and S. Saurabh. Parameterized complexity of geometric covering problems having conflicts. *Algorithmica*, 82(1):1–19, 2020.

[5] L. Barto, A. A. Krokhin, and R. Willard. Polymorphisms, and how to use them. In A. A. Krokhin and S. Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[6] É. Bonnet, P. Giannopoulos, and M. Lampis. On the parameterized complexity of red-blue points separation. In D. Lokshtanov and N. Nishimura, editors, *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *LIPIcs*, pages 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[7] É. Bonnet and T. Miltzow. Parameterized hardness of art gallery problems. *ACM Transactions on Algorithms*, 16(4):42:1–42:23, 2020.

[8] K. Bringmann, L. Kozma, S. Moran, and N. S. Narayanaswamy. Hitting set for hypergraphs of low vc-dimension. In P. Sankowski and C. D. Zaroliagis, editors, *Proceedings of the 24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[9] G. Călinescu, A. Dumitrescu, H. J. Karloff, and P. Wan. Separating points by axis-parallel lines. *International Journal Of Computational Geometry & Applications*, 15(6):575–590, 2005.

[10] C. Carbonnel and M. C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, 2016.

[11] B. S. Chlebus and S. H. Nguyen. On finding optimal discretizations for two attributes. In L. Polkowski and A. Skowron, editors, *Proceedings of the First International Conference on Rough Sets and Current Trends in Computing (RSCTC 1998)*, volume 1424 of *Lecture Notes in Computer Science*, pages 537–544. Springer, 1998.

[12] M. Dom, M. R. Fellows, F. A. Rosamond, and S. Sikdar. The parameterized complexity of stabbing rectangles. *Algorithmica*, 62(1-2):564–594, 2012.

[13] G. Even, R. Levi, D. Rawitz, B. Schieber, S. Shahar, and M. Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Transactions on Algorithms*, 4(3):34:1–34:17, 2008.

[14] V. Froese. *Fine-Grained Complexity Analysis of Some Combinatorial Data Science Problems*. PhD thesis, Technische Universität Berlin, 2018.

[15] S. Garcia, J. Luengo, J. A. Sáez, V. López, and F. Herrera. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, 2013.

[16] P. Giannopoulos, C. Knauer, G. Rote, and D. Werner. Fixed-parameter tractability and lower bounds for stabbing problems. *Computational Geometry*, 46(7):839–860, 2013.

[17] P. Giannopoulos, C. Knauer, and S. Whitesides. Parameterized complexity of geometric problems. *Computer Journal*, 51(3):372–384, 2008.

[18] R. Hassin and N. Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29–42, 1991.

[19] P. Heggernes, D. Kratsch, D. Lokshtanov, V. Raman, and S. Saurabh. Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing via iterative localization. *Information and Computation*, 231:109–116, 2013.

[20] S. Kratsch, T. Masarík, I. Muzi, M. Pilipczuk, and M. Sorge. Optimal discretization is fixed-parameter tractable. *arXiv:2003.02475 [cs.DS]*, 2020.

[21] S. Langerman and P. Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

[22] H. Liu, F. Hussain, C. L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423, 2002.

[23] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, fourth edition, 2011.

[24] Y. Yang, G. I. Webb, and X. Wu. Discretization methods. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 101–116. Springer, 2010.