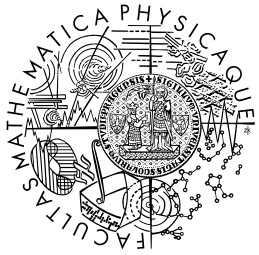

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

William Tatarko

CSP over oriented trees

Department of Algebra

Supervisor of the bachelor thesis: RNDr. Jakub Bulín, Ph.D.

Study programme: Mathematics

Study branch: General Mathematics

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I would like to thank my supervisor RNDr. Jakub Bulín, Ph.D. I highly appreciate his guidance, helpful advices, patience and wise criticism.

Title: CSP over oriented trees

Author: William Tatarko

Department: Department of Algebra

Supervisor: RNDr. Jakub Bulín, Ph.D., Department of Algebra

Abstract: In this thesis we present an oriented tree with only 26 vertices, whose CSP is NP-complete. This serves as a counterexample for a conjecture that any oriented tree with this property has at least 39 vertices. The work itself is divided into three chapters. In the first one, basic definitions and tools of this topic are introduced. Then, tractability of trees of special shapes is shown. Finally, NP-completeness of a certain oriented tree is proven.

Keywords: CSP oriented trees NP-completeness

Contents

Introduction

Constraint satisfaction problem (or CSP for short) is a decision problem, whether or not it is possible to assign values to the given variables in such a way that all the given constraints are satisfied. Since 1970s it has been a subject of study for many mathematicians and theoretical computer scientists. There are two main reasons for its popularity. Firstly, its mathematical structure is rich enough to provide an environment for an in-depth examination [1]. Despite this, it is still very general, and therefore it may represent a wide range of real-life problems. For instance, job scheduling, where the complete process is composed of several tasks, some of which must finish before another can start. E.g. complex class scheduling at an university can be solved similarly. Other examples include choosing frequencies for a mobile phone network, construction of DNA strings with certain properties or laying out components on a circuit board.

In 1999, Feder and Vardi published an article [2], in which they introduced the famous dichotomy conjecture. It states that CSP of every relational structure is either in **P** or it is **NP**-complete (assuming $\mathbf{P} \neq \mathbf{NP}$). The conjecture had been motivated by the presence of proofs for special cases. Namely, Schaefer's dichotomy theorem for Boolean CSPs in 1978 [3] and dichotomy for graph coloring in 1990 [4]. In 2001 the dichotomy for CSP was proven for oriented cycles [5]. Later (in 2008), it was also proven for all smooth digraphs (i.e. digraphs whose each vertex has an incoming and an outgoing edge) [6]. This was the motivation to examine oriented trees, because (in some sense) they are as different from the smooth digraphs as possible. However, at the beginning (in 2009), the dichotomy was proven only for a particular class of oriented trees called special triads [7]. Finally (in 2017), two proofs of the dichotomy conjecture for a general relational structure were found. One by Dmitriy Zhuk [8] and the other by Andrei Bulatov [9].

An **NP**-complete oriented tree might be used e.g. as a counterexample. Ideally, there should be a small representative of **NP**-complete oriented trees, with which mathematicians could work conveniently. The first known **NP**-complete oriented tree, which was discovered in 1991, had 287 vertices [10]. Shortly after that, it has been simplified to an **NP**-complete oriented tree with 81 vertices [11]. Later (in 1996), an **NP**-complete oriented tree (special triad) with only 45 vertices was found [12]. Finally, in the article [7], which proved the dichotomy for special triads, a new **NP**-complete oriented tree was discovered as a byproduct. It has 39 vertices, which makes it (to our knowledge) the smallest known **NP**-complete oriented tree (and the smallest **NP**-complete special triad [7]). In this work we present an **NP**-complete oriented tree (triad), which has only 26 vertices.

It is noteworthy that the majority of the recent progress (including [7, 8, 9]) in the area of CSP has been done using so-called algebraic approach. More precisely, based on the Galois correspondence between relations and operations, which has been discovered independently in [13, 14], it has been proven that CSP depends only on polymorphisms [15]. For instance, the main theorem of [7] has been

proven by working with polymorphisms. In this work we decided to adapt this technique.

1. Fundamentals

Firstly, we must introduce basic definitions, notation, mathematical tools etc. Therefore, a reader who is familiar with the fundamentals of computational complexity and general mathematics (e.g. digraphs) may skip this chapter.

1.1 Computational complexity

We would like to talk about the computational difficulty of certain problems. In order to do so, we are going to (informally speaking[1]) represent every object as a sequence of ones and zeros. Therefore it is useful to define a set of all such object representations.

Definition 1. *Let $U = \{0,1\}^* = \bigcup_{n=0}^{\infty}\{0,1\}^n$, i.e. a set of all finite sequences of ones and zeros. Then U is called the universe.*

We are going to be mainly concerned about the question whether or not some object has some property. In other words, we can define a subset of the universe U such that it consists of representations of objects with the specific property (and nothing more) and then simply ask whether a given object has its representation in the subset.

Definition 2. *Let $D \subseteq U$, where U is the universe. Then a decision problem (given by) D is the problem to decide whether or not a given $x \in U$ satisfies $x \in D$.*

An algorithm is a function, which takes an object as an input (e.g. a statement or two numbers) and transforms it into an output object (e.g. a boolean value or a number). Since the universe U contains representations of all objects, it is possible to think of an algorithm as a function from U to U, which may be computed in n steps[2].

We would like to find an algorithm that could solve a certain decision problem, i.e. an algorithm, whose output would be yes or no. Our main problem is going to be the question whether there exists such an algorithm, which is also fast enough. Generally, an algorithm is said to be fast if it works in polynomial time in the size of the input.

Definition 3. *Let $f : U \to U$ be an algorithm. If there exist $a, b \in \mathbb{N}$ such that $f(x)$ can be from x computed in $|x|^a + b$ steps, where $|x|$ denotes the size (e.g. the length of the representation) of the input x, then f is said to be computable in polynomial time. If f is computable in polynomial time, then we also say that f is in class $\mathbf{P}$ (denoted by $f \in \mathbf{P}$) or that f is tractable.*

[1] Since we are going to be working mainly with high-level mathematical objects, we are going to be a little bit informal.

[2] For the purpose of this work it will be sufficient to understand the step intuitively. Note that in order to define it formally, we would need to introduce Turing machine.

Since we are going to be often talking about decision problems, it is convenient to be able to talk about a decision problem without mentioning the algorithm, which solves it (i.e. the algorithm that decides whether the given object belongs to the specified subset of the universe). Hence, we will say that a decision problem $D \subseteq U$ is solvable (or decidable) in polynomial time (denoted by $D \in \mathbf{P}$) or that D is tractable, if there exists an algorithm such that it is computable in polynomial time and it determines whether or not an input x belongs to D.

The goal of this work is to find an object such that the decision problem described by its properties is not going to be in class **P**. Therefore, we need to define a bigger class of problems, which will also contain harder problems. More precisely, we would like to define a class of decision problems such that for every decision problem D from that class (but no other) and every object there exists a witness (i.e. solution or proof), using which we can at least (in polynomial time) verify whether the object belongs to D.

Definition 4. *Let us define* **NP** *as a class of all decision problems D, for which there exists an algorithm computable in polynomial time (verifier) $w : U \to U$ and $a, b \in \mathbb{N}$ with the following properties. For every $x \in D$ there exists (a witness) $y \in U$ such that $|y| \leq |x|^a + b$ and $w(x, y) = 1$. For all $x \notin D$ and $y \in U$ holds $w(x, y) = 0$.*

It would be useful to define a way how to compare the difficulty of different decision problems. If there exists a fast (i.e. computable in polynomial time) algorithm that can transform one decision problem into another, then the first one could be obviously solved by firstly transforming it into the second one and then solving it. Hence, the first problem is clearly at most as hard as the second one.

Definition 5. *Let $A, B \subseteq U$, where U is the universe, be decision problems. If there exists an algorithm f such that it is solvable in polynomial time and $x \in A \iff f(x) \in B$, then we say that a decision problem A is (poly-time) reducible to B (denoted by $A \leq B$). If $A \leq B$ and $A \geq B$ hold true, then the decision problems are said to be (poly-time) equivalent (denoted by $A \equiv B$).*

Now we are able to compare the difficulty of two problems. In order to compare the difficulty of one problem to the whole set of problems, we need to define the following.

Definition 6. *Let $D \subseteq U$ be a decision problem and $\mathcal{C} \subseteq \mathcal{P}(U)$, where $\mathcal{P}(\cdot)$ denotes the power set, a class of decision problems. If for every $C \in \mathcal{C}$ holds $C \leq D$, then D is said to be $\mathcal{C}$-hard.*

Class **NP** may contain harder problems than **P**.[3] But clearly $\mathbf{P} \subseteq \mathbf{NP}$, and thus being in **NP** does not necessarily mean being hard. It would be convenient to define a class of truly hard problems.

Definition 7. *Let $D \in \mathbf{NP}$. If D is also* **NP***-hard, then it is called* **NP***-complete.*

[3]Assuming that $\mathbf{P} \neq \mathbf{NP}$. With this assumption it has been also shown that there exist problems, which are in **NP**, but are neither in **P** nor in **NP**-complete [16].

1.2 Digraphs

The main object of our study are going to be oriented trees, i.e. a type of digraphs. Therefore, let us firstly define what a digraph is.

Definition 8. *Let G be a (finite) set of vertices and let $E \subseteq G^2$ be a set of edges. Then a tuple $\mathbb{G} = (G, E)$ is called a digraph. We will denote $(a, b) \in E$ by $a \to b$ or $a \overset{\mathbb{G}}{\to} b$.*

In order to be able to define the kind of digraph that we are most interested in, we firstly need to define a path and a cycle.

Definition 9. *Let $\mathbb{G} = (G, E)$ be a digraph and let $\{v_0, \ldots, v_n\} \subseteq G$, where $n \geq 0$, be a set of (unique) vertices such that $(v_{i-1} \to v_i) \vee (v_{i-1} \leftarrow v_i)$ for $\forall i \in \{1, \ldots, n\}$. Then a sequence $(v_0, \ldots, v_n)$ is called a path (of length n). If additionally $(v_0 \to v_n) \vee (v_0 \leftarrow v_n)$, we say that it is a cycle.*

Now we are ready to define a tree, i.e. the type of a digraph, which is going to be the main object of our study.

Definition 10. *A (connected) digraph is called an oriented tree, if it has no cycles.*

In order to be able to define the decision problem, whose computational complexity we want to examine, we need to define a homomorphism.

Definition 11. *Let $\mathbb{G} = (G, E_G)$ and $\mathbb{H} = (H, E_H)$ be digraphs and let $f : G \to H$ be a mapping. If for all $a, b \in G$ holds that $a \overset{\mathbb{G}}{\to} b$ implies $f(a) \overset{\mathbb{H}}{\to} f(b)$, then is f called a homomorphism.*

There are several important types of homomorphisms. Let us define them.

Definition 12. *A homomorphism $f : \mathbb{G} \to \mathbb{H}$ is called an endomorphism if $G = H$. An endomorphism $f : \mathbb{G} \to \mathbb{G}$ is called an automorphism, if it is bijective.*

Later we will see, that for CSP it is not necessary to consider the whole digraph, but only its (special) part, which is called a core.

Definition 13. *Let $\mathbb{G} = (G, E_G)$ be a digraph. Then a digraph $\mathbb{C} = (C, E_C)$, where $C \subseteq G$ and $E_C = \{(a, b) \in E_G; a \in C, b \in C\}$, is called a core of $\mathbb{G}$ if each endomorphism on C is an automorphism.*

As we have already mentioned, a core can be used to examine properties of the whole tree. Therefore it is crucial to know that every digraph has (up to isomorphism) exactly one core. Hence, it will actually be the core. Proof of the following lemma can be found in [17].

Lemma 1. *Let $\mathbb{G}$ be a digraph. Then $\mathbb{G}$ has (up to isomorphism) exactly one core.*

Furthermore, it can be easily shown that the core of an oriented tree is also an oriented tree.

An oriented tree consists of vertices and arrows (i.e. oriented edges). We can draw the tree in such a way that all arrows will be oriented upwards. Then some vertices will be higher than others. Inspired by this we can define the following mapping.

Definition 14. *Let $\mathbb{T} = (T, E)$ be an oriented tree. Let us define a mapping $\mathrm{lvl} : T \to \mathbb{N} \cup \{0\}$ such that for all tuples $(a, b) \in E$ holds $f(b) = f(a) + 1$ and such that there exists a vertex $x \in T$ such that $f(x) = 0$. Value $\mathrm{lvl}(x)$ is called the level of a vertex $x \in T$. Moreover, the number $\max_{x \in T} \mathrm{lvl}(x)$ is called the height of $\mathbb{T}$.*

It is easy to see that mapping lvl is uniquely determined. Later we will show that based on the height it can be sometimes easily decided whether or not an oriented tree is tractable.

There exists a special (in some sense simple) type of oriented trees. Before we are able to define it, we must introduce a new property of vertices.

Definition 15. *Let $\mathbb{T} = (T, E)$ be a digraph. Let us define a mapping $\deg : T \to \mathbb{N} \cup \{0\}$ by $\deg(x) = y$, where $x \in T$, if y is the count of vertices $z \in T$ satisfying $(x \to z) \vee (x \leftarrow z)$. Number $\deg(x)$ is called the degree of a vertex $x \in T$.*

Finally, we are ready to define the mentioned type of oriented trees.

Definition 16. *Let $\mathbb{T} = (T, E)$ be an oriented tree. If there exists a vertex (called the root) $r \in T$ such that $\deg(r) = n$, where $n \geq 3$, and for $\forall x \in T \setminus \{r\}$: $\deg(x) \in \{1, 2\}$, then $\mathbb{T}$ is called a polyad. By removing r from $\mathbb{T}$ we will obtain several components. Each component together with the root is called a branch. The number of the components (that is equal to n) will be called the degree of a polyad. A polyad of degree three is called a triad.*

1.3 Constraint satisfaction problem

Now we are getting to the main topic of this work, which is the computational complexity of a certain decision problem called a constraint satisfaction problem (or CSP for short). It is possible to define it for a general relational structure, however for the purposes of this work, it will be sufficient to define it only for digraphs.

Definition 17. *Let $\mathbb{G} = (G, E_G)$ be a digraph. Then $\mathrm{CSP}(\mathbb{G})$ is a decision problem, whether or not there exists a homomorphism $f : \mathbb{H} \to \mathbb{G}$, where $\mathbb{H} = (H, E_H)$ is a given digraph.*

As it has been already mentioned, if we want to investigate, which tree has its CSP in **P**, it is sufficient to examine only its core.

Lemma 2. *Let $\mathbb{G}$ be a digraph and $\mathbb{C}$ its core. Then* CSP($\mathbb{G}$) *and* CSP($\mathbb{C}$) *are poly-time equivalent.*

Proof. If $\mathbb{G} = \mathbb{C}$, then the statement holds. Otherwise, there exists an endomorphism, which is not an automorphism, and thus it is not surjective. Hence, we have found a homomorphism from $\mathbb{G}$ to a smaller digraph $\mathbb{G}_1$. We can repeat this process until we find such $i \in \mathbb{N}$ that $\mathbb{G}_i = \mathbb{C}$ (up to isomorphism). By composing the appropriate homomorphisms we get a homomorphism from $\mathbb{G}$ to $\mathbb{C}$.

Clearly, there exists a homomorphism from $\mathbb{C}$ to $\mathbb{G}$ given by the inclusion $C \subset G$. Therefore there exist homomorphisms from $\mathbb{G}$ to $\mathbb{C}$ and from $\mathbb{C}$ to $\mathbb{G}$. This implies that for every digraph $\mathbb{A}$ there exists a homomorphism from $\mathbb{A}$ to $\mathbb{G}$ if and only if there exists a homomorphism from $\mathbb{A}$ to $\mathbb{C}$. Hence CSP($\mathbb{G}$) = CSP($\mathbb{C}$), from which the lemma follows. □

To make expressions a bit easier, we will allow ourselves to talk about the computational complexity of some CSP without explicitly mentioning CSP. We will say that $\mathbb{G}$ is in **P** or that $\mathbb{G}$ is tractable, if CSP($\mathbb{G}$) $\in$ **P**. Similarly, we will say that $\mathbb{G}$ is **NP**-complete or that $\mathbb{G}$ is intractable, if CSP($\mathbb{G}$) is **NP**-complete.

1.4 Polymorphisms

It can be shown that the computational complexity of CSP depends only on the existence of special functions, which are in some sense compatible with the oriented tree, whose CSP we are solving [15, 18]. Using this method we will not be forced to work directly with algorithms or computational complexity in general. It will be possible to transfer the problem completely into the language of universal algebra. Let us begin by defining the mentioned special operation.

Definition 18. *Let $\mathbb{G} = (G, E)$ be a digraph. An n-ary function $f : G^n \to G$ is called a polymorphism, if $\bigwedge_{i=1}^{n}(v_i \to u_i)$ implies $f(v_1, \ldots, v_n) \to f(u_1, \ldots, u_n)$, where $v_i, u_i \in G$ for all $i \in \{1, \ldots, n\}$. We also say that f is compatible with $\mathbb{G}$ or that $\mathbb{G}$ admits f.*

A basic, but very important, property of a function is so-called idempotency.

Definition 19. *A function $f : G^n \to G$ is said to be idempotent, if $f(x, \ldots, x) = x$ for all $x \in G$.*

There are many types of polymorphisms. However, the most convenient kind of polymorphism for us is so-called weak near-unanimity.

Definition 20. *Let $\mathbb{G} = (G, E)$ be a digraph and let $w : G^n \to G$, where $n \geq 2$, be an idempotent polymorphism. If for all $x, y \in G$ holds $w(x, \ldots, x, y) = w(x, \ldots, x, y, x) = \cdots = w(y, x, \ldots, x)$, then w is called a weak near-unanimity polymorphism of arity n (shortly WNU or n-WNU).*

It has been proven that a digraph is intractable if it does not admit any weak near-unanimity polymorphism (of any arity) [18, 19]. Therefore, in order to prove the tractability of a digraph, it is sufficient to show the existence of a weak near-unanimity of some arity, since the dichotomy conjecture has already been proven in [8, 9].

Before the proof of the dichotomy, the tractability could be shown by the existence of various polymorphisms such as semilattices, or, for instance, Mal'cev polymorphism [15, 20, 21]. Also the intractability can be shown differently. For example, it is possible to prove it by demonstrating the nonexistence of compatible Taylor or Siggers polymorphism [18, 22].

Generally, there had been many ways how to prove tractability (or intractability) of a digraph and after the proof of the dichotomy conjecture, there are even more. However, in this work we will only need to use weak near-unanimity polymorphisms. The theorem below follows from [20] and will be very important in the main proof of this work (see later).

Theorem 3. *Let $\mathbb{G}$ be a digraph, which is a core. Then $\mathbb{G}$ is tractable if and only if $\mathbb{G}$ admits weak near-unanimity polymorphism of some arity.*

Note that the theorem above could be rewritten using a general digraph (which is not necessarily a core). However, then we would have to consider a weak near-unanimity polymorphism, which is not idempotent. This can be easily shown by using the endomorphism to the core, however for the purpose of this thesis Theorem 3 we be sufficient.

2. Tractable trees

In this chapter we are going to show how in some (special) cases we can easily decide whether an oriented tree is tractable. In other words, we are going to develop methods that can be (often) used to show that an oriented tree is tractable. In the next chapter we will find a small oriented tree, on which these methods cannot be used and whose intractability will be later proven.

2.1 Trees of small height

In this section we are going to prove that every oriented tree of height smaller than four is tractable. Before we will start, let us add one more definition.

Definition 21. *Let $\mathbb{T} = (T, E)$ be an oriented tree and $a \in T$ be a vertex. Let* $\mathrm{dist} : T \to \mathbb{N} \cup \{0\}$ *be a mapping such that* $\mathrm{dist}(a, a) = 0$ *and for all* $x \in T \setminus \{a\} : \mathrm{dist}(a, x) = n + 1$ *if there exist vertices* $v_1, \ldots, v_n$ *such that the sequence* $(a, v_1, \ldots, v_n, x)$ *is a path. Value* $\mathrm{dist}(a, x)$ *will be called the distance from a to x (in $\mathbb{T}$).*

We aim to prove Theorem 6, which states that every oriented tree of height at most three is tractable. Because of the following lemma it will be sufficient to demonstrate that every oriented tree of height at most three has a path as its core.

Lemma 4. *Every digraph whose core is a path is tractable.*

Proof. From Lemma 2 it follows that it is sufficient to prove that the core $\mathbb{C} = (C, E)$ is tractable. If $\mathbb{C}$ is a single vertex, then its CSP can be reduced to a question whether the input digraph contains an (oriented) edge, which is clearly in **P**. Let us then suppose that the core $\mathbb{C}$ has two vertices of degree one (because it is a path of the length at least one). Let us arbitrary choose one of them and denote it a. Let us define a binary operation $f : C \times C \to C$ such that $f(x, y) = x$ if $\mathrm{dist}(a, x) \leq \mathrm{dist}(a, y)$ (which means that x lies on the path from a to y) and $f(x, y) = y$ otherwise.

We are going to show that f is a 2-WNU polymorphism, i.e. (by the definition) a commutative idempotent polymorphism compatible with $\mathbb{C}$. Obviously, $f(x, x) = x$, since $\mathrm{dist}(a, x) \leq \mathrm{dist}(a, x)$, and thus f is idempotent. The mapping f is also (clearly from its definition) commutative.

Now we are going to prove that f is a polymorphism admitted by $\mathbb{C}$. Let us suppose that $f(p, q) = p$, $p \to r$ and $q \to s$. If $p = q$, then $p \to s$, and thus f is a polymorphism regardless of the value of $f(r, s)$ (because $f(r, s) \in \{r, s\}$).

Let us then suppose that $p \neq q$. Therefore, $\mathrm{dist}(a, p) < \mathrm{dist}(a, q)$. Seeking a contradiction, we suppose that $f(r, s) = s$. Similarly to $p \neq q$ we can also assume that $r \neq s$, and thus $\mathrm{dist}(a, r) > \mathrm{dist}(a, s)$.

We know that $\mathrm{dist}(a, r) = \mathrm{dist}(a, p) \pm 1$ and $\mathrm{dist}(a, s) = \mathrm{dist}(a, q) \pm 1$. Because $\mathrm{dist}(a, p) < \mathrm{dist}(a, q)$ and $\mathrm{dist}(a, r) > \mathrm{dist}(a, s)$, the only option is $\mathrm{dist}(a, r) = \mathrm{dist}(a, p) + 1$, $\mathrm{dist}(a, s) = \mathrm{dist}(a, q) - 1$ and $\mathrm{dist}(a, q) = \mathrm{dist}(a, p) + 1$. This is a little bit more clear from Figure 2.1, where r' (s') denotes the other

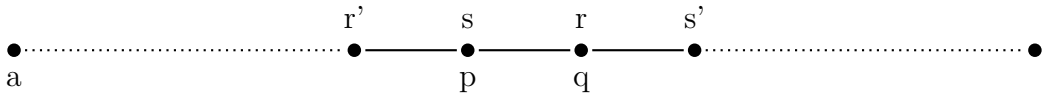

Figure 2.1: Helper picture of an oriented path (with highlighted significant vertices) for the proof of Lemma 4.

choice for r (s), which has been rejected (in order to satisfy the inequality $\mathrm{dist}(a, r) > \mathrm{dist}(a, s)$).

But this implies that $r = q$ and $s = p$, and therefore $(p \to q) \wedge (p \leftarrow q)$, which is a contradiction, because $\mathbb{C}$ is an oriented path. Hence, $\mathbb{C}$ admits a binary commutative idempotent polymorphism (i.e. 2-WNU) f, and therefore it is tractable (it follows from Theorem 3). Finally, tractability of the core $\mathbb{C}$ implies tractability of the whole digraph (it follows from Lemma 2). □

As promised, now we are going to show that the core of an arbitrary tree of height smaller than four is a path, and thus that the tree is tractable (following Lemma 4). Let us begin by proving it for trees whose height is smaller than three.

Lemma 5. *Every oriented tree of height smaller than three is tractable.*

Proof. Let $\mathbb{T}$ be an oriented tree of height two. Obviously, $\mathbb{T}$ contains vertices a, b and c such that $a \to b$ and $b \to c$, because otherwise it would not be possible to reach level two. Let us define a mapping $f : T \to T$ by

$$f(x) = \begin{cases} a & \text{if } \mathrm{lvl}(x) = 0, \\ b & \text{if } \mathrm{lvl}(x) = 1, \\ c & \text{if } \mathrm{lvl}(x) = 2. \end{cases}$$

Now we are going to prove that f is a homomorphism. If $x \to y$, then $\mathrm{lvl}(x) = 0 \wedge \mathrm{lvl}(y) = 1$ or $\mathrm{lvl}(x) = 1 \wedge \mathrm{lvl}(y) = 2$. In both cases $f(x) \to f(y)$ by the definition of f. Hence, f is a homomorphism.

Clearly $f(\mathbb{T})$ (i.e. a, b and c with the appropriate edges) is the core of $\mathbb{T}$. Because the core of $\mathbb{T}$ is an oriented path, its CSP is tractable (following Lemma 4) and so is CSP($\mathbb{T}$) (following Lemma 2).

If $\mathbb{T}$ is an oriented tree of height one, the statement can be proven analogically (using a core $a' \to b'$ and a simplified function f'). If the height of $\mathbb{T}$ is zero (i.e. $\mathbb{T}$ is a single vertex), then its CSP can be reduced to a question whether the input digraph contains an (oriented) edge, which is clearly in **P**. □

All lemmata for the main theorem of this section are prepared. However, it would be convenient to introduce one more definition, before we are going to prove the theorem. It specifies the shape of a path, i.e. when it goes down and when up (if we use the convention that oriented edges are arrows pointing upwards).

Definition 22. *Let us define a path of type $d_1 d_2 \dots d_n$ (or shortly $d_1 d_2 \dots d_n$-path), where $d_i \in \{0, 1\}$, as a path $(v_0, \dots, v_n)$ such that $v_{i-1} \to v_i$ if $d_i = 1$ and $v_{i-1} \leftarrow v_i$ if $d_i = 0$ for $\forall i \in \{1, \dots, n\}$. We also allow ourselves to write $(d_1 d_2 \dots d_n)^k$ instead of $(d_1 d_2 \dots d_n) \dots (d_1 d_2 \dots d_n)$.*

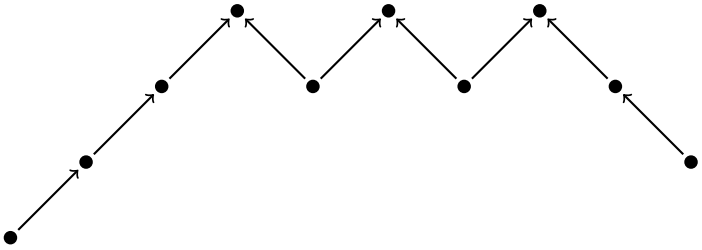

Figure 2.2: This is an example of 111010100-path (i.e. $111(01)^200$-path or $1^3(01)^20^2$-path). Note that we assume that the path starts on the left.

See Figure 2.2 for an illustration of Definition 22. We have prepared everything, which is needed for the main theorem of this section.

Theorem 6. *Every oriented tree of height at most three is tractable.*

Proof. Because of Lemma 5, we only need to prove that oriented trees of the height (exactly) three are tractable. Let $\mathbb{T} = (T, E)$ be an oriented tree of the height three. Let $\mathbb{C} = (a, c_0, d_0, c_1, d_1, \ldots, c_n, d_n, b)$ be a path (i.e. a subtree) of minimal length such that $\mathrm{lvl}(a) = 0$ and $\mathrm{lvl}(b) = 3$. Note that there might be more such paths (we can arbitrary choose one).

Let t be the type of the path $\mathbb{C}$ (starting from a). Firstly, we are going to show that $t = 11(01)^n1$ for some $n \in \mathbb{N} \cup \{0\}$. Because we start at a vertex of level zero, t must start with 1. If t did start with 10, then the first two vertices could be removed from $\mathbb{C}$, which would be a contradiction with the minimality of the length of $\mathbb{C}$. Therefore t starts with 11. Clearly t cannot start with $11(01)^m00$ for any $m \in \mathbb{N} \cup \{0\}$, because otherwise we could remove the first $2m+4$ vertices from $\mathbb{C}$. Therefore t must be $11(01)^n1$, since e.g. $11(01)^n101$ would be again a contradiction with the minimality of the length of $\mathbb{C}$ (we could remove the last two vertices). Since $t = 11(01)^n1$, we know that $\mathrm{lvl}(c_i) = 1$ and $\mathrm{lvl}(d_i) = 2$ for all $i \in \{0, \ldots, n\}$.

Now we are going to define a function $f : T \to C$ by stepwisely mapping each vertex of tree $\mathbb{T}$. The algorithm is defined in Algorithm 1 and illustrated in Figure 2.3. Note that we decided to define f algorithmically in order to increase the readability.

Now we are going to show that in each step of Algorithm 1, the definition of a homomorphism is not violated, and thus that f is a homomorphic function. In the rest of the proof, by "vertex x is connected" we will mean that for $x \in T$ there exists an already mapped vertex $y \in M$ such that either $x \to y$ or $x \leftarrow y$.

Clearly, no vertex mapped on line 7 or line 8 is connected, and thus the definition of a homomorphism is not violated.

Vertices that are being processed on line 10 are connected only to vertices of level zero, which have been mapped to a on line 7. Because $a \to c_0$, the definition of a homomorphism is not violated.

Vertices that are being processed on line 12 may be connected to vertices of level one or three. In the first case, the definition of a homomorphism is not violated, since $c_0 \to d_0$. If x is connected to a vertex y of level three, than there must be vertices $x_0, x_1 \in T$ such that $x_0 \to x_1 \to x \to y$. From the minimality of the length of $\mathbb{C}$ it follows that $n = 0$, and thus $f(x) \to f(y)$, since $f(x) = d_0$, $f(y) = b$ and $d_0 \to b$.

Vertices that are being processed on line 18 may be connected to vertices of the level zero or two. The first case cannot actually occur, because such x had to be already mapped on line 10. Let us assume that x is connected to a vertex y of the level two. Vertex y must have been mapped to d_{i_L}, because it had to be mapped to d_j for some $j \in \{1, \ldots, i_L\}$ and if $j < i_L$, then x would have to be already mapped (in some previous iteration). From the definition of $\mathbb{C}$ and the fact that $f(x) = c_i$ and $f(y) = d_{i_L}$ it follows that $f(x) \to f(y)$ (because $i_L \in \{i-1, i\}$).

Vertices that are being processed on line 20 may be connected to vertices of the level three or one. If x is connected to a vertex y of level three, then there exists an oriented path $(a', c'_0, d'_0, \ldots, c'_i, x, y)$ such that $\mathrm{lvl}(a') = 0$. From the minimality of the length of $\mathbb{C}$ and from the fact that $i \leq n$ follows that $i = n$, and thus $f(x) = d_n$. Hence $f(x) \to f(y)$, because y was mapped to b on line 8 and from the definition of $\mathbb{C}$ it follows that $d_n \to b$. Let us now assume that x is connected to a vertex y of level one. Then $f(y) = c_i$. Therefore, from $f(x) = d_i$ and the definition of $\mathbb{C}$ it follows that $f(y) \to f(x)$.

At this point, the unmapped vertices are only of the level one or two such that between them and some vertex of the level zero must be a vertex of the level three. Therefore all vertices that are being processed on line 22 and line 23 can be connected only to vertices of level three or vertices mapped in this step (i.e. on line 22 or on line 23). However, each vertex x that could be connected to a vertex of level three y (i.e. $\mathrm{lvl}(x) = 2$) is being mapped to d_n, and thus $f(x) \to f(y)$ (because $f(x) = d_n$, $f(y) = b$ and $d_n \to b$). Finally, if a vertex x is connected to a vertex y that has been mapped in this step, then the definition of a homomorphism is also not violated, because either $f(x) = d_n$, $f(y) = c_n$, or $f(x) = c_n$, $f(y) = d_n$.

Hence, we have found a homomorphic function from T to $C \subseteq T$. In order to prove that $\mathbb{C}$ is actually the core of $\mathbb{T}$, we ought to show that $\mathbb{C}$ cannot be further reduced. However, that is clearly not possible, because the core must contain a vertex of the level zero and a vertex of the level three and $\mathbb{C}$ is the smallest subtree which satisfies this condition.

□

2.2 Drawable trees

In this section we are going to show how to be (sometimes) sure that an oriented tree is tractable literally at the first glance. Let us begin by defining a property that will allow us to do so.

Definition 23. *We will say that an oriented tree is drawable if it can be drawn on an Euclidean plane with (straight) horizontal parallel lines in such a way that*

- *vertices of the same level are all drawn on the same line,*
- *vertices of higher level are higher on the plane,*
- *all edges are (straight) line segments and no edges cross,*
- *no vertices are drawn at the same position.*

Algorithm 1 Algorithm used for the definition of the homomorphism f from the proof of Theorem 6. Note that indices i_L and i_U are present only for the purpose of commentary.

$i \leftarrow 0$ ▷ Iteration index.
$i_L \leftarrow 0$ ▷ Last iteration index.
$i_U \leftarrow 0$ ▷ "Unbound" iteration index.
$M \leftarrow \{\}$ ▷ Set of all mapped vertices.
$P \leftarrow \{x \in T; \exists \text{ path } (x, y_1 \ldots, y_n) \forall i \in \{1, \ldots, n\} \mathrm{lvl}(y_i) \neq 3 \wedge \mathrm{lvl}(y_n) = 0\}$

for all $x \in T \setminus M$ **do**
    if $\mathrm{lvl}(x) = 0$ **then** $f(x) := a$ **and** add x to M
    else if $\mathrm{lvl}(x) = 3$ **then** $f(x) := b$ **and** add x to M

for all $x \in T \setminus M$ **do**
    if $\mathrm{lvl}(x) = 1$ **and** $\exists y \in M : y \to x$ **then** $f(x) := c_0$ **and** add x to M

for all $x \in T \setminus M$ **do**
    if $\mathrm{lvl}(x) = 2$ **and** $\exists y \in M : y \to x$ **then** $f(x) := d_0$ **and** add x to M

while $P \cap M \neq P$ **do**
    $i_L \leftarrow i$
    $i_U \leftarrow i_U + 1$
    if $i < n$ **then** $i \leftarrow i + 1$

    for all $x \in T \setminus M$ **do**
        if $\mathrm{lvl}(x) = 1$ **and** $\exists y \in M : y \leftarrow x$ **then** $f(x) := c_i$ **and** add x to M

    for all $x \in T \setminus M$ **do**
        if $\mathrm{lvl}(x) = 2$ **and** $\exists y \in M : y \to x$ **then** $f(x) := d_i$ **and** add x to M

for all $x \in T \setminus M$ **do**
    if $\mathrm{lvl}(x) = 2$ **then** $f(x) := d_n$ **and** add x to M
    else if $\mathrm{lvl}(x) = 1$ **then** $f(x) := c_n$ **and** add x to M

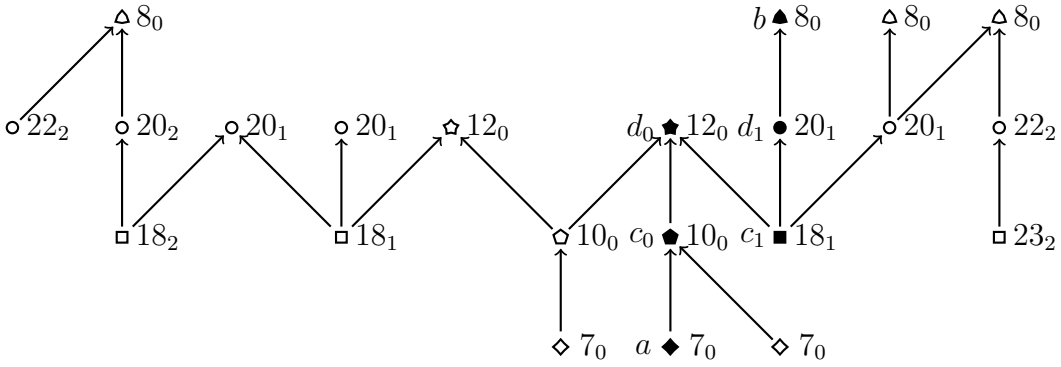

Figure 2.3: This is a visualization of Algorithm 1 from the proof of Theorem 6. Vertices that are represented by full shapes and denoted (on the left) by letters (corresponding to the proof) are part of the core. Other vertices are denoted by empty shapes and the shape itself symbolizes to which vertex (of the core) is each vertex mapped. The number (on the right) of each vertex represents on which line of Algorithm 1 is the vertex mapped and its subscript corresponds to i_U from Algorithm 1.

Note that the definition above is a little bit informal, however the meaning should be clear from Figures 2.4a and 2.4b. We will show how drawability can be used to define a (2-WNU) polymorphism, which implies the tractability of the oriented tree.

(a) A drawable oriented tree. (b) An undrawable oriented tree.

Figure 2.4: Simple examples of oriented trees.

Theorem 7. *Every drawable oriented tree is tractable.*

Proof. Let $\mathbb{T} = (T, E)$ be a drawable oriented tree. Without loss of generality, we can suppose that $\mathbb{T}$ is a core, because if the whole tree is drawable, then its core is also drawable and if we prove that the core is tractable, then the whole tree must be tractable as well.

We will show how to define a binary idempotent commutative polymorphism (i.e. 2-WNU) compatible with $\mathbb{T}$, which will imply the tractability of $\mathbb{T}$ (Theorem 3).

Let us define functions $h : T \to \mathbb{R}$ and $v : T \to \mathbb{R}$ such that $h(x)$ will be the horizontal and $v(x)$ the vertical position of $x \in T$ on the plane (from the Definition 23). Now we will define a function, which will informally speaking select the lower vertex (and if both vertices have the same vertical position, it will select the leftmost vertex). Let us define a binary operation $f : T \times T \to T$ by

$$f(x,y) = \begin{cases} x & \text{if } v(x) < v(y), \\ y & \text{if } v(x) > v(y), \\ g(x,y) & \text{if } v(x) = v(y), \end{cases}$$

where $g : T \times T \to T$ is a function such that

$$g(x,y) = \begin{cases} x & \text{if } h(x) \leq h(y), \\ y & \text{if } h(x) > h(y). \end{cases}$$

The operation f is idempotent, because $f(x,x) = g(x,x) = x$. Clearly, f is also commutative, because the definition is always symmetric. Now we only need to prove that f is a polymorphism compatible with $\mathbb{T}$.

Let $x, y \in T$ be such that $x \neq y$. We know that $f(x,y) \in \{x,y\}$. Let us firstly assume that $v(x) \neq v(y)$. Without loss of generality, let us assume that $v(x) < v(y)$, and thus $f(x,y) = x$. If $x \to z$ and $y \to w$ (or $x \leftarrow z$ and $y \leftarrow w$), then $v(z) < v(w)$ (because v corresponds to the level), and therefore $f(z,w) = z$. This meets the definition of a polymorphism.

Let us now suppose that $v(x) = v(y)$. Without loss of generality, let us assume that $h(x) < h(y)$. Let $z, w \in T$ such that $x \to z$ and $y \to w$ (or $x \leftarrow z$ and $y \leftarrow w$). Because $\mathbb{T}$ is drawable, no edges are crossed, and therefore either $z = w$, or $h(z) < h(w)$. Hence, $f(z,w) = g(z,w) = z$, which, again, meets the conditions of a polymorphism. Therefore f is 2-WNU compatible with $\mathbb{T}$, and thus $\mathbb{T}$ is tractable (Theorem 3). □

3. Intractable trees

This chapter is divided into two sections. In the first one, we are going to use the knowledge about tractable trees from the previous chapter (e.g. drawability) in order to find an **NP**-complete candidate, i.e. an oriented tree, whose tractability cannot be easily proven with the tools, which have been already introduced. In the second section, we are going to prove that the **NP**-complete candidate is truly **NP**-complete.

3.1 Definition

Our aim is to find an **NP**-complete oriented tree as small as possible. From this it follows that we only need to consider cores, because every **NP**-complete oriented tree has an **NP**-complete core of smaller (or equal) size.

We know that every oriented tree, whose core is a path is tractable (Lemma 4). Therefore, we are looking for a more complicated oriented tree (and a core at the same time) than a path. Again, because we are looking for the smallest one, we decided to start searching among the simplest possible oriented trees (which are not paths), i.e. triads.

The previous smallest known **NP**-complete oriented tree was the one discovered in [7] and visualized in Figure 3.1. It has 39 vertices and it is a so-called special triad, i.e. a triad whose root is of the level zero and whose each branch ends with a vertex of the level zero and contains exactly one vertex of the level equal to the height of the triad and except the end no vertex of the level zero. Therefore, it also makes sense to start with triads, because if we were able to show that no smaller **NP**-complete triad exists, we would at least prove that the one from [7] is the smallest among triads.

In Theorem 6 we have proven that every oriented tree of the height at most three is tractable. Therefore, we are looking for a triad, which is of the height at least four. Again, we are trying to start investigating as simple oriented tree as possible. Hence, we decided to firstly check such triads that only one branch reaches level four.

We have also shown that every drawable tree is tractable (Theorem 7). In summary, we are looking for a triad, which is of the height four, but only one branch reaches level four, which is a core and which is not drawable. Let us start searching among triads, whose root is of the level zero.

Now we are going to take the advantage of the easy visualization of drawability. If one of the three branches does not reach level zero again (i.e. besides the root), it can be drawn in the following way. The other two branches will be drawn on sides (one on the left and the other on the right) in such a way that the horizontal distance from the root will be an increasing function of the distance dist from the root. Then the branch, which does not reach level zero again, can be drawn in the middle between the two other branches similarly to Figure 2.4a. Hence, every branch (of an **NP**-complete candidate) must reach level zero again.

Let us begin by finding the smallest branch of the height four, which has the potential to be a part of the sought triad. It cannot contain a 1111-path, because otherwise the whole tree would collapse, i.e. there would exist a homomorphism

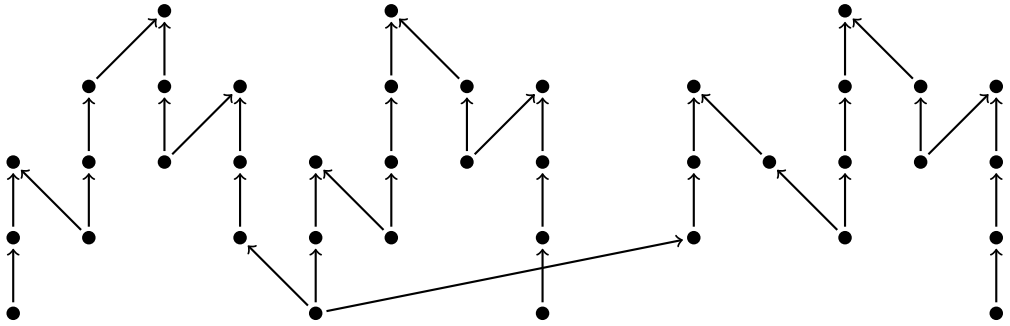

Figure 3.1: This is the previous smallest known **NP**-complete oriented tree found in [7]. Note that in the originally published version of [7] was a mistake (the oriented tree had 33 vertices, not 39).

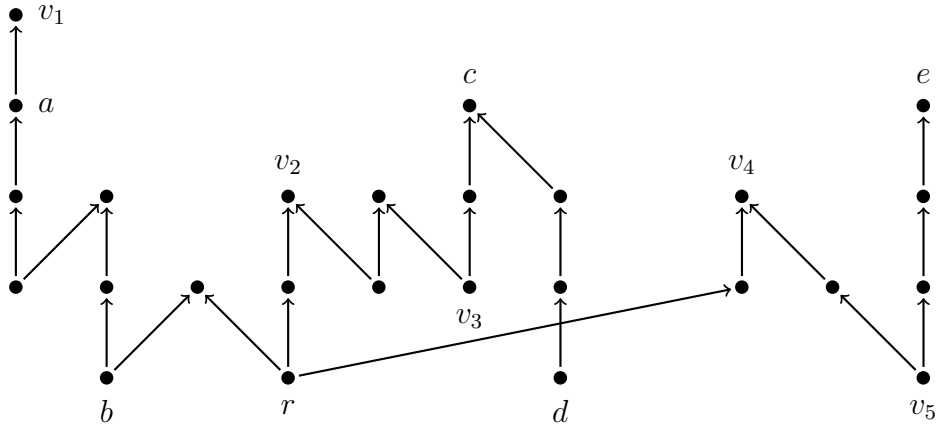

Figure 3.2: This is the new smallest known **NP**-complete oriented tree $\mathbb{T}_{\mathbf{NP}}$, i.e. the main result of this work. Note that vertices to which we will often refer are denoted by letters without subscript.

to the 1111-path (similarly to the proof of Lemma 5), and thus the core would actually be only the 1111-path. Let us firstly try **NP**-complete candidates with a branch of the height four that contains a 110111-path. It must reach level zero. Let us think in a greedy way and try to minimize the number of tree vertices by considering the smallest path, which contains a 110111-path and which starts at level zero, then reaches another vertex of the level zero and finally reaches a vertex of the level four, i.e. a path of type 10110111.

It is a priori not clear how to continue building the **NP**-complete candidate. We were trying to find the smallest triad with the branch described above, on which we could use a similar trick (proving its **NP**-completeness) to the one used on a special triad in [7]. By trial and error we have examined several triads. Always, after proving that it is tractable, we slightly changed the shape of the oriented tree in such a way that the same argument (proving its tractability) could not be used anymore. Using this method we have obtained the oriented tree drawn in Figure 3.2, which we will denote $\mathbb{T}_{\mathbf{NP}}$.

3.2 Verification

We are going to prove that the oriented tree $\mathbb{T}_{\mathbf{NP}}$ is intractable. Firstly, we are going to show that the triad $\mathbb{T}_{\mathbf{NP}}$ is a core.

Lemma 8. *The oriented tree $\mathbb{T}_{\mathbf{NP}}$ is a core.*

Proof. Clearly, the 110111-path mentioned above must be a part of the core, because it has height 4, and thus it cannot be mapped anywhere else. If we map r to b, then v_5 must be mapped to b as well, but then it is not possible to map e anywhere. Therefore, the whole path from v_1 to r has to be in the core. Again, if v_5 is mapped to r, it becomes impossible to map e. Hence, the whole the path from v_1 to v_5 must be in the core. From this follows that also the path from v_1 to e must be a part of the core.

Vertex v_2 can be mapped only to v_4 or itself. If it is mapped to v_4, then it will not be possible to map c. Hence, v_2 must be mapped to itself. If v_3 is not mapped to itself, then there will be no vertex, to which we can map c. Therefore v_3 must be mapped to itself, from which it follows that c also has to be mapped to itself. Finally, it is clear that also d cannot be mapped to a different vertex. Hence, every endomorphism is an automorphism, and thus $\mathbb{T}_{\mathbf{NP}}$ is a core. □

Because $\mathbb{T}_{\mathbf{NP}}$ is a core, its CSP is **NP**-complete if and only if there does not exist any (idempotent) WNU (of any arity). We are going to prove this nonexistence. Let us start by developing new notation.

Definition 24. *Let $\mathbb{P} = (p_1, \ldots, p_n)$ be a path. Then we will define a function $\mathbb{P} : T_{\mathbf{NP}} \to \mathcal{P}(T_{\mathbf{NP}})$ (induced from the path $\mathbb{P}$) such that for $x \in T_{\mathbf{NP}}$ the set $\mathbb{P}(x)$ will be a set of all vertices $y \in T_{\mathbf{NP}}$ such that there exists a homomorphism $h : \mathbb{P} \to \mathbb{T}_{\mathbf{NP}}$ satisfying $h(p_1) = x$ and $h(p_n) = y$.*

The goal is to prove that no (idempotent) WNU w compatible with $\mathbb{T}_{\mathbf{NP}}$ can be defined. We will achieve that by demonstrating $w(e, a, \ldots, a) \neq w(a, \ldots, a, e)$. Therefore it will be necessary to make deductions from statements of the form $w(x_1, \ldots, x_n) = y$. From the definition of a polymorphism it follows that if $\bigwedge_{i=1}^{n}(x_i \to x_i')$ and $w(x_1', \ldots, x_n') = y'$, then $y \to y'$. If additionally $\bigwedge_{i=1}^{n}(x_i' \to x_i'')$ and $w(x_1'', \ldots, x_n'') = y''$, then $y \to y' \to y''$. It is easy to see that this procedure can be generalized. We can use an arbitrary path, not just 11-path as in this case. More formally, if $w(x_1, \ldots, x_n) = y$ and if there exists a path $\mathbb{P} = (p_1, \ldots, p_m)$ and homomorphisms $h_i : \mathbb{P} \to \mathbb{T}_{\mathbf{NP}}$ such that $h_i(p_1) = x_i$ and $h_i(p_m) = x_i'$, then $w(x_1', \ldots, x_n') \in \mathbb{P}(y)$. Henceforward, we will be saying "$w(x_1, \ldots, x_n) = y$ implies $w(x_1', \ldots, x_n') \in \mathbb{P}(y) = \{y_1', \ldots, y_m'\}$", from which it is clear, which path was used to obtain the result. If the path $\mathbb{P}$ was selected in such a way that $\mathbb{P}(y) = \{y'\}$, then obviously $w(x_1', \ldots, x_n') = y'$.

In Table 3.1 we have listed all paths, on which the following proofs will be based. In Attachment A.1 is Table A.1, which contains all sets of form $\mathbb{P}_i(x)$ that are going to be implicitly used in the rest of this chapter. There is also Algorithm 2 that can be used to generate these values, although it is clearly possible to obtain the values without the help of a computer. Now we are prepared to prove the following lemmata. Let us start by computing the value of $w(e, a, \ldots, a)$.

Path name	Path type
$\mathbb{P}_1$	10110011011
$\mathbb{P}_2$	00100101101011
$\mathbb{P}_3$	00100110011011011
$\mathbb{P}_4$	101101011
$\mathbb{P}_5$	00100
$\mathbb{P}_6$	1011001101011
$\mathbb{P}_7$	00100101100111
$\mathbb{P}_8$	0010100101100110 11011
$\mathbb{P}_9$	0010100110011011
$\mathbb{P}_{10}$	00011001011011
$\mathbb{P}_{11}$	00101001011011

Table 3.1: List of paths, which will be used in proofs in this section.

Lemma 9. *For every weak near-unanimity polymorphism w compatible with $\mathbb{T}_{\mathbf{NP}}$ must hold $w(e, a, \ldots, a) = a$*

Proof. Since w is idempotent, the equality $w(r, \ldots, r) = r$ must hold. From this (and the reasoning above) it follows that $w(e, a, \ldots, a) \in \mathbb{P}_1(r) = \{a, e\}$. Seeking a contradiction, let us suppose that $w(e, a, \ldots, a) = e$.

The equality $w(e, a, \ldots, a) = e$ implies $w(e, c, \ldots, c) \in \mathbb{P}_2(e) = \{e\}$. From $w(e, a, \ldots, a) = e$ we also obtain that $w(c, a, \ldots, a) \in \mathbb{P}_3(e) = \{c, e\}$. From $w(r, \ldots, r) = r$ follows $w(c, a, \ldots, a) \in \mathbb{P}_4(r) = \{a, c\}$. Hence $w(a, \ldots, a, c) = w(c, a, \ldots, a) = c$, which implies $w(b, \ldots, b, d) \in \mathbb{P}_5(c) = \{d\}$. Finally, from this it follows that $w(e, c, \ldots, c) \in \mathbb{P}_6(d) = \{c\}$. This is a contradiction, and thus $w(e, a, \ldots, a) = a$. □

Lemma 10. *For every weak near-unanimity polymorphism w compatible with $\mathbb{T}_{\mathbf{NP}}$ must hold $w(a, e, \ldots, e) = e$.*

Proof. Since w is idempotent, equality $w(r, \ldots, r) = r$ must hold. From this it follows that $w(a, e, \ldots, e) \in \mathbb{P}_1(r) = \{a, e\}$. Seeking a contradiction, let us suppose that $w(a, e, \ldots, e) = a$.

From $w(a, e, \ldots, e) = a$ follows $w(a, c, \ldots, c) \in \mathbb{P}_3(a) = \{a\}$. Therefore $w(e, c, \ldots, c) \in \mathbb{P}_7(a) = \{e\}$, which implies $w(c, \ldots, c, e) = e$. From this it follows that $w(a, e, \ldots, e) \in \mathbb{P}_8(e) = \{e\}$. This is a contradiction, and thus $w(a, e, \ldots, e) = e$. □

Lemma 11. *Let w be a weak near-unanimity polymorphism compatible with $\mathbb{T}_{\mathbf{NP}}$. If $w(\underbrace{a, \ldots, a}_{k}, e, e, \ldots, e) = e$, then $w(\underbrace{a, \ldots, a}_{k}, c, e, \ldots, e) = e$.*

Proof. From $w(r, \ldots, r, r, r, \ldots, r) = r$ it follows that $w(a, \ldots, a, c, e, \ldots, e) \in \mathbb{P}_6(r) = \{a, c, e\}$. We are going to show that the two other values lead to contradictions.

Seeking a contradiction, let us suppose that $w(a, \ldots, a, c, e, \ldots, e) = a$. Therefore $w(a, \ldots, a, e, e, \ldots, e) \in \mathbb{P}_9(a) = \{a\}$. This is a contradiction.

Seeking a contradiction, let us now suppose that $w(a, \ldots, a, c, e, \ldots, e) = c$. This implies $w(e, \ldots, e, c, e, \ldots, e) \in \mathbb{P}_7(c) = \{c\}$, and thus $w(e, \ldots, e, c) = c$.

Therefore $w(a,\dots,a,c) \in \mathbb{P}_{10}(c) = \{c\}$. Hence $w(a,\dots,a,e) \in \mathbb{P}_9(c) = \{c,e\}$. Finally, from $w(r,\dots,r) = r$ it follows that $w(a,\dots,a,e) \in \mathbb{P}_1(r) = \{a,e\}$. Hence $w(e,a,\dots,a) = e$. This is a contradiction with Lemma 9. □

Lemma 12. *Let w be a weak near-unanimity polymorphism compatible with $\mathbb{T}_{\mathbf{NP}}$. If $w(\underbrace{a,\dots,a}_{k},c,e,\dots,e) = e$, then $w(\underbrace{a,\dots,a}_{k},a,e,\dots,e) = e$.*

Proof. From $w(a,\dots,a,c,e,\dots,e) = e$ it follows that $w(a,\dots,a,a,e,\dots,e) \in \mathbb{P}_{11}(e) = \{e\}$. □

Theorem 13. *Constraint satisfaction problem of the oriented tree $\mathbb{T}_{\mathbf{NP}}$ is* **NP***-complete.*

Proof. It is sufficient to prove that there does not exist any weak near-unanimity operation w (of any arity) compatible with $\mathbb{T}_{\mathbf{NP}}$.

From Lemma 10 it follows that $w(a,e,e,\dots,e) = e$. Hence, from Lemma 11 we can obtain $w(a,c,e,\dots,e) = e$. Therefore $w(a,a,e,\dots,e) = e$ (from Lemma 12). By repeated application of Lemmata 11 and 12, we can obtain $w(a,\dots,a,e) = e$. Therefore $w(e,a,\dots,a) = e$, since w is a weak near-unanimity operation. This is a contradiction with Lemma 9. Hence, no (idempotent) weak near-unanimity polymorphism compatible with $\mathbb{T}_{\mathbf{NP}}$ can be defined, from which it follows that $\mathrm{CSP}(\mathbb{T}_{\mathbf{NP}})$ is **NP**-complete. □

Conclusion

We have found an oriented tree with 26 vertices, whose CSP is **NP**-complete. This serves as a counterexample to the conjecture from [7], which states that their **NP**-complete oriented tree with 39 vertices is the smallest one. However, it is still not clear, whether our oriented tree is actually the smallest one. We have nevertheless managed to lower the minimal count of vertices from 39 to 26. Furthermore, we have introduced tools (such as the drawability), which might be used for further research.

Bibliography

[1] Libor Barto, Andrei Krokhin, and Ross Willard. *Polymorphisms, and How to Use Them*, volume 7 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017.

[2] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104 (electronic), 1999.

[3] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[4] Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory Ser. B*, 48(1):92–110, February 1990.

[5] T. Feder. Classification of homomorphisms to oriented cycles and of k-partite satisfiability. *SIAM Journal on Discrete Mathematics*, 14(4):471–480, 2001.

[6] Libor Barto, Marcin Kozik, and Todd Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of Bang-Jensen and Hell). *SIAM Journal on Computing*, 38(5):1782–1802, September 2008.

[7] Libor Barto, Marcin Kozik, Miklós Maróti, and Todd Niven. CSP dichotomy for special triads. *Proceedings of the American Mathematical Society*, 137(9):2921–2934, 2009.

[8] D. Zhuk. A proof of CSP dichotomy conjecture. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, volume 00, pages 331–342, Oct. 2017.

[9] A. A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, Oct 2017.

[10] Wolfgang Gutjahr, Emo Welzl, and Gerhard Woeginger. Polynomial graph-colorings. *Discrete Appl. Math.*, 35(1):29–45, November 1991.

[11] W. Gutjahr. *Graph colourings*. Ph.D. thesis, Free University Berlin, 1991.

[12] P. Hell, J. Nešetřil, and X. Zhu. Complexity of tree homomorphisms. *Discrete Applied Mathematics*, 70(1):23–36, 1996.

[13] David Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27:95–100, 1968.

[14] V. G. Bodnarčuk, L. A. Kalužnin, V. N. Kotov, and B. A. Romov. Galois theory for post algebras. i, II. *Otdelenie Matematiki, Mekhaniki i Kibernetiki Akademii Nauk Ukrainskoi SSR. Kibernetika*, 3:no. 5, 1–9, 1969.

[15] Peter Jeavons, David Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, July 1997.

[16] Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, January 1975.

[17] P. Hell and J. Nešetřil. The core of a graph. *Discrete Math*, 109:117–126, 1992.

[18] Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

[19] Miklós Maróti and Ralph McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3-4):463–489, 2008.

[20] L. Barto and M. Kozik. Constraint satisfaction problems of bounded width. In *50th Annual IEEE Symposium on Foundations of Computer Science, 2009. FOCS '09*, pages 595–603, oct 2009.

[21] Andrei A. Bulatov. Mal'tsev constraints are tractable. *Electronic Colloquium on Computational Complexity (ECCC)*, 2002.

[22] Mark H. Siggers. A strong Mal'cev condition for locally finite varieties omitting the unary type. *Algebra universalis*, 64(1):15–20, Oct 2010.

A. Attachments

A.1 Details about the paths from Section 3.2

In this attachment we are going to provide details about the oriented paths, which have been used in Section 3.2 to prove **NP**-completeness of $\mathbb{T}_{\mathbf{NP}}$. In Table A.1 can be found all important images (which have been used in Section 3.2 implicitly) of functions induced from the paths from 3.1. Algorithm 2 then provides an easy way, how to generate the values in Table A.1.

Path	$\mathbb{P}_*(a)$	$\mathbb{P}_*(b)$	$\mathbb{P}_*(c)$	$\mathbb{P}_*(d)$	$\mathbb{P}_*(e)$	$\mathbb{P}_*(r)$
$\mathbb{P}_1$	$\emptyset$	$\{a, e\}$	$\emptyset$	$\{c\}$	$\emptyset$	$\{a, e\}$
$\mathbb{P}_2$	$\{a, c\}$	$\emptyset$	$\{c\}$	$\emptyset$	$\{e\}$	$\emptyset$
$\mathbb{P}_3$	$\{a\}$	$\emptyset$	$\{c\}$	$\emptyset$	$\{c, e\}$	$\emptyset$
$\mathbb{P}_4$	$\emptyset$	$\{a, c\}$	$\emptyset$	$\{c\}$	$\emptyset$	$\{a, c\}$
$\mathbb{P}_5$	$\{b\}$	$\emptyset$	$\{d\}$	$\emptyset$	$\{v_5\}$	$\emptyset$
$\mathbb{P}_6$	$\emptyset$	$\{a, c, e\}$	$\emptyset$	$\{c\}$	$\emptyset$	$\{a, c, e\}$
$\mathbb{P}_7$	$\{e\}$	$\emptyset$	$\{c\}$	$\emptyset$	$\{e\}$	$\emptyset$
$\mathbb{P}_8$	$\{a, e\}$	$\emptyset$	$\{a, c, e\}$	$\emptyset$	$\{e\}$	$\emptyset$
$\mathbb{P}_9$	$\{a\}$	$\emptyset$	$\{c, e\}$	$\emptyset$	$\{e\}$	$\emptyset$
$\mathbb{P}_{10}$	$\emptyset$	$\emptyset$	$\{c\}$	$\emptyset$	$\{a, e\}$	$\emptyset$
$\mathbb{P}_{11}$	$\{a\}$	$\emptyset$	$\{a, c\}$	$\emptyset$	$\{e\}$	$\emptyset$

Table A.1: List of important images of functions induced from the paths from Table 3.1. Note that e.g. in row $\mathbb{P}_9$ and column $\mathbb{P}_*(c)$ is the set $\mathbb{P}_9(c)$.

Algorithm 2 Given a vertex *start_vertex*, path type *path* (i.e. an array of ones and zeros) and the array of all edges (i.e. pairs of vertices) (of $\mathbb{T}_{\mathbf{NP}}$) *edges*, the following function returns the set $\mathbb{P}(start\_vertex)$, where $\mathbb{P}$ is a path of the type *path*.

```
1: procedure GetEndVertices(start_vertex, path, edges)
2:     end_vertices ← {start_vertex}
3:     for all path_edge ∈ path do
4:         new_end_vertices ← ∅
5:         for all old_end_vertex ∈ end_vertices do
6:             for all edge ∈ edges do
7:                 if path_edge = 1 and old_end_vertex = edge[0] then
8:                     new_end_vertices ← new_end_vertices ∪ {edge[1]}
9:                 else if path_edge = 0 and old_end_vertex = edge[1] then
10:                    new_end_vertices ← new_end_vertices ∪ {edge[0]}
11:        end_vertices ← new_end_vertices
       return end_vertices
```