Binary Commutative Polymorphisms of Core Triads

Michael Wernthaler

February 10, 2021

Contents

1 Abstract

It has been known for a while that for a given graph H the complexity of CSP(H) also known as the H-colouring problem only depends on the set of polymorphisms of H. It follows from the results of Bulatov [4] and of Zhuk [6] from 2017 that H-colouring problem is in P if H has a so-called (4-ary) Siggers polymorphism. In this paper we focus on the case where H is a so-called *triad*, i.e., an orientation of a tree which has a single vertex of degree 3 and otherwise only vertices of degree 2 and 1. (...) We describe an efficient algorithm that checks the existence of Siggers polymorphisms for triads up to a certain number/size of vertices/armlength?.

2 Introduction

Let $H = (V, E)$ be a finite directed graph. The *H-colouring problem* (also called the *constraint satisfaction problem for H*) is the problem of deciding for a given finite graph G whether there exists a homomorphism from G to H. Note that if $H = K_k$, the clique with k vertices, then the H-/colouring problem equals the famous k-colouring problem, which is NP-hard for $k \geq 3$ and which can be solved in polynomial time if $k \leq 2$.

It has been known for a while that the complexity of the H-colouring problem only depends on the set of *polymorphisms* of H. It follows from results of Bulatov [4] and of Zhuk [6] from 2017 that the H-colouring problem is in P of H has a so-called *(4-ary) Siggers* polymorphism, i.e., an operation $s : V^4 \longrightarrow V$ which satisfies for all $a, e, r \in V$

$$s(a, r, e, a) = s(r, a, r, e)$$

Before these results, the complexity of CSP(H) was open even if H is an orientation of a tree. It is not obvious at all how an orientation of a tree looks like if it has a Siggers polymorphism. In fact, this question is already open if H is a *triad*, i.e., an orientation of a tree which has a single vertex of degree 3 and otherwise only vertices of degree 2 and 1. Jakob Bulin claims that the following triad with 22 vertices has no Siggers polymorphism.

$$01001111, 0110000, 101000$$

Here, 0 stands for forward edge, 1 stands for backward edge, and the three words stand for the three paths that leave the vertex of degree 3 of the triad. He also claims that all smaller triads do have a Siggers polymorphism, and conjectures that an orientation of a tree has a Siggers polymorphism if and only if it has a binary polymorphism f satisfying $f(u, y) = f(y, x)$ for all $x, y \in V$. Jakub Bulin conjectures that in this case the Path-Consistency algorithm can solve the H-colouring problem.

3 TODO Lemma

To show that our proposed algorithm runs correctly we first have to prove the following lemma.

Lemma 1. Let $\mathbb{T}$ be a finite tree. The following are equivalent

1. $\mathbb{T}$ is a core
2. $End(\mathbb{T}) = \{id\}$
3. $AC_{\mathbb{T}}(\mathbb{T})$ terminates with $L(v) = v$ for all vertices v of $\mathbb{T}$

Proof: 1. $\Rightarrow$ 2.: Note, that every unique shortest path from v to w maps to the unique shortest path from $f(v)$ to $f(w)$, since f is an automorphism of a tree. It follows, that there must be a leaf u on which f is not the identity, because otherwise $f = id$.

We consider p to be the unique path from u to $f(u)$, which maps to the unique path p' from $f(u)$ to $f(f(u))$. Our claim is that there has to be a vertex v on p for which $f(v) = v$. To show this we take the orbit of u and the paths in between.

In the simple case we suppose that $f(f(u)) = u$. This implies $f(u_i) = u_{l-i}$ for $i \in \{0, 1, ..., l\}$. Since no double-edges are allowed, we conclude that $l = 2m$, which gives us $f(u_m) = u_m$.

For the general case, we consider the orbit of u to be of size $n \geq 3$. Because of $f(u_0) = u_l$ there is a greatest $m \leq l$ such that $f(u_i) = u_{l-i}$, for every $i \in \{0, 1, ..., m\}$ from which follows that there must be a cyclic path from u_m to $f^n(u_m) = u_m$ of length $n(l - 2m)$. Since $\mathbb{T}$ is a tree, we require that $n(l - 2m) = 0$. The latter equation can only be satisfied for $l = 2m$, and again we get $f(u_m) = u_m$.

Now let $\mathbb{T} = v(T_1, T_2, .., T_k)$, where T_i are the components of $T - v$. We know that $T_a \to T_b$ for at least one pair T_a, T_b, where T_a contains u and T_b contains $f(u)$. We then construct an endomorphism h of $\mathbb{T}$ by taking f on T_a. For every other component we define h as id. It's easy to see that h is non-injective, since f maps T_a ot T_b.

However, this means that $\mathbb{T}$ can't be a core, which means our assumption was wrong and $End(\mathbb{T})$ cannot contain such a f, but only id.

2. $\Rightarrow$ 1: If $End(\mathbb{T}) = id$, then the only homomorphism $h : \mathbb{T} \to \mathbb{T}$ is id, which is an automorphism. Hence $\mathbb{T}$ must be a core.

2. $\Rightarrow$ 3: Suppose that $End(\mathbb{T}) = \{id\}$. To prove that $AC_{\mathbb{T}}(\mathbb{T})$ terminates with $L(v) = \{v\}$ for all vertices v of $\mathbb{T}$ we use a modified version of the prove of implication $4 \Rightarrow 2$ of Theorem 2.7 in the script of Graph-Homomorphisms *?*.

Let v' be an arbitrary vertex in $\mathbb{T}$. By choosing a vertex u from the list of each node v we can construct a sequence $f_0, ..., f_n$ for $n = |V(\mathbb{T})|$, where f_i is a homomorphism from the subgraph of $\mathbb{T}$ induced by the vertices at distance at most i to v', and f_{i+1} is an extension of f_i for all $1 \leq i \leq n$. We start by defining f_0 to map v' to an arbitrary vertex $u' \in L(v')$.

Suppose inductively, that we have already defined f_i. Let w be a vertex at distance $i+1$ from v' in $\mathbb{T}$. Since $\mathbb{T}$ is an orientation of a tree, there is a unique $w' \in V(\mathbb{T})$ of distance i from v' in $\mathbb{T}$ such that $(w, w') \in E(\mathbb{T})$ or $(w', w) \in E(\mathbb{T})$. Note that $x = f_i(w')$ is already defined. In case that $(w', w) \in E(\mathbb{T})$, there must be a vertex y in $L(w)$ such that $(x, y) \in E(\mathbb{T})$, since otherwise the arc-consistency procedure would have removed x from $L(w')$. We then set $f_{i+1}(w) = y$. In case that $(w, w') \in E(\mathbb{T})$ we can proceed analogously. By construction, the mapping f_n is an endomorphism of $\mathbb{T}$.

Knowing that id is the only endomorphism of $\mathbb{T}$ we get $v' = f_n(v') = u'$. Since v' and u' both were chosen arbitrarily, it follows that $L(v) = \{v\}$, for every vertex $v \in \mathbb{T}$.

3. $\Rightarrow$ 2: It's obvious, that always $\{id\} \subseteq End(\mathbb{T})$. Since $AC_{\mathbb{T}}(\mathbb{T})$ derived $L(v) = v$ for all vertices v of $\mathbb{T}$ we know there can't be another homomorphism h for which $h(v) \neq v$, hence $End(\mathbb{T}) = \{id\}$.

4 TODO Arc-Consistency Procedure

Implement the arc-consistency procedure such that your algorithm runs in linear time in the size of the input.

Algorithm 1: $AC_{\mathbb{T}}$ ($\mathbb{T}$ is a triad)
1 Input: digraph $\mathbb{G}$, initial lists $L : G \mapsto P(T)$ Output: Is there a homomorphism $h : \mathbb{G} \mapsto \mathbb{T}$ such that $h(v) \in L(v)$ for all $v \in G$

4.1 Notes

- Can we optimize AC for paths?
- Done by implementing AC-3 for graphs

4.2 Benchmarks

5 TODO Core Triads

5.1 Algorithm

Let n be the maximal arm length. The number of possible paths is $p = \sum_{i=1}^{n} 2^i$ and there are p^3 triads. To reduce the number of cases to look at we consider only triads that are cores, i.e., not homomorphically equivalent to smaller triads. Thus, we pose the following question.

Question 1. When is a triad homomorphically equivalent to a smaller triad?

A method to answer this question has already been presented in Lemma 1. We simply run $AC_{\mathbb{T}}(\mathbb{T})$ and see, if it derives $L(v) = \{v\}$ for every vertex v. If this is the case, then we know that $\mathbb{T}$ is a core.

However, this approach is inefficient. As an example, consider the case, in which a triad θ has two identical arms. We can easily see, that θ is not a core without the need to apply the costly AC-procedure. Below, we will

formulate a lemma, based upon which we can decide whether to discard triads at an earlier stage in the generation process (). We need the following definitions as prerequisites:

Definition 5.1. A *partial triad* is a triad of the form $(p_1 p_2 p_3)$, where $p_i = \varepsilon$ for at least one $i \in \{1, 2, 3\}$.

Each partial triad θ can be completed to form a triad τ by adding arms to it. In this case we say that τ was derived from θ. Note, that adding arms to a partial triad puts further restrictions on its root node. Therefore running $AC_\theta(\theta)$ on a partial triad θ, (is insufficient) for (making a statement) about a triad derived from θ. E.g. consider the arm 100, on which AC doesn't derive only *id*. Yet, $100, 11, 00$ is still a core.

Thus we define $ACR_{\mathbb{T}}$ as a modification of $AC_{\mathbb{T}}$ that initially colours the root r with $L(r) = \{r\}$.

Definition 5.2. A *rooted core* (RC) names a partial triad θ for which $ACR_\theta(\theta)$ did derive $L(v) = \{v\}$ for every vertex v.

Now let θ be a partial triad, and τ be a triad derived from θ.

The following table summarizes

$AC_\theta(\theta) \to id$	no statement
$AC_\theta(\theta) \nrightarrow id$	no statement
$ACR_\theta(\theta) \to id$	no statement
$ACR_\theta(\theta) \nrightarrow id$	τ **cannot** be a core

Algorithm 2: Algorithm for finding core triads

Input: An unsigned integer m

Output: A list of all core triads whose arms each have a length $\leq m$

```
// Finding a list of RCAs
armlist ⟵ [ ];
foreach arm p with length(p) ≤ m do
    if ACR_p(p) didn't derive L(v) ≠ v for any vertex v then
        put p in armlist

// Assembling the RCAs to core triads
triadlist ⟵ [ ];
foreach {p1, p2} in armlist do
    if ACR_{p1p2}(p1p2) derived L(v) ≠ v for some vertex v then
        Drop the pair and cache the two indices;

foreach triad 𝕋 = {p1, p2, p3} do
    if 𝕋 contains a cached index pair then
        Drop 𝕋 and continue;
    if AC_𝕋(𝕋) didn't derive L(v) ≠ v for some vertex v then
        Put 𝕋 in triadlist;

return triadlist
```

Finally, this gives us the following lemma.

Lemma 2. Every partial triad that is not a RC cannot be completed to form a core triad.

Algorithm 2 displays the pseudo-code of the entire core triad generation.

5.2 Optimizations

Lemma 3. Let $H = (V, E)$ be a graph and let f be a polymorphism of H. ()Then f is also a polymorphism of $\bar{H}$.

Proof: Let $H = (V, E)$ be a graph and let $\bar{H} = (V, \bar{E})$ be the graph where $\bar{E}(\bar{H}) = \{(y, x) \mid (x, y) \in E(H)\}$. A mapping $f : V(H)^k \to V(H)$ is a polymorphism of H, if and only if $(f(u_1, ..., u_k), f(v_1, ..., v_k)) \in E(H)$ whenever $(u_1, v_1), ..., (u_k, v_k)$ are arcs in $E(H)$. Now let f be a polymorphism of H. Since $(f(v_1, ..., v_k), f(u_1, ..., u_k)) \in E(\bar{H})$ whenever $(v_1, u_1), ..., (v_k, u_k)$ are arcs in $\bar{E}(\bar{H})$, f is also a polymorphism of $\bar{H}$.

We have to generate only have of all triads e.g. ~~("01","0","11")~~ because ("10","1","00")

5.2.1 Numbers of triads

armlength	4	5	6	7	8
number of triads	27	265	2667	22547	189681
		9,81	10,06	8,5	8,4

6 TODO Commutative Polymorphisms

Write an algorithm that enumerates all core triads that do not have a commutative polymorphism up to a fixed path-length. For every triad $\mathbb{T}$ there is a unique homomorphism

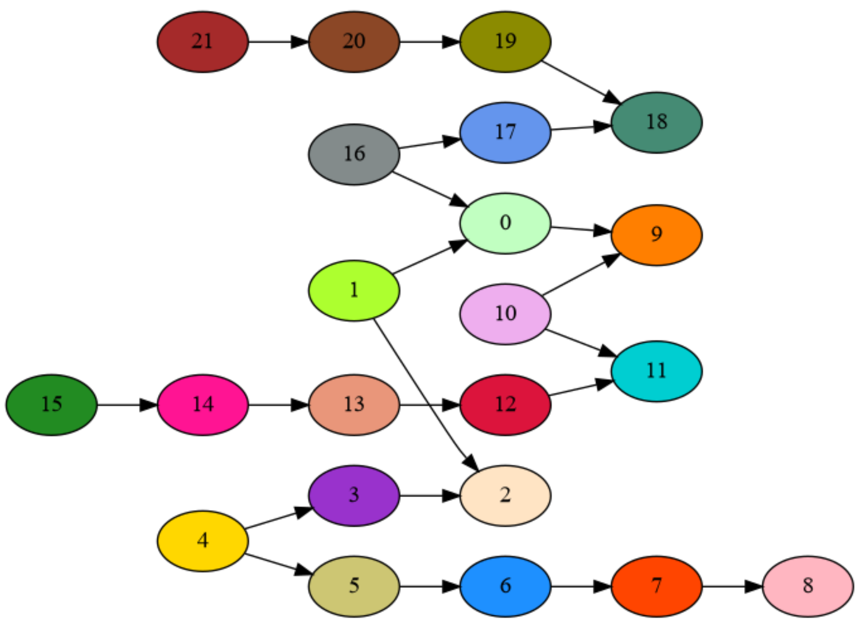

Figure 1: 01001111,0110000,101000

6.1 Notes

- Singleton-arc-consistency receives the following graph as its input:
  - Calculate the productgraph of $\mathbb{T}$ with itself

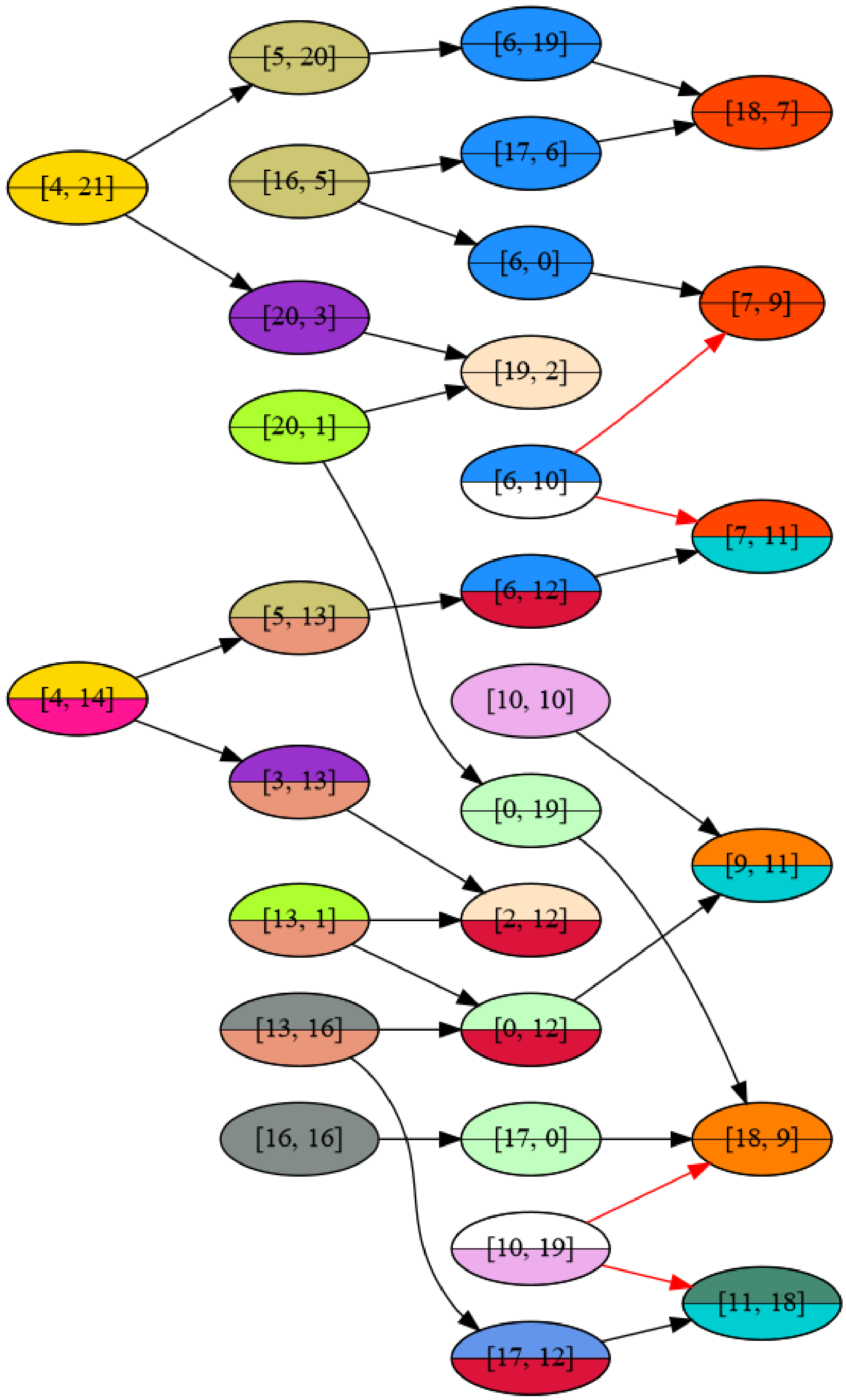

Figure 2: Reduced component of powergraph

– Merge every pair of vertices (x, y) and (y, x) to one vertex

7 Notes

7.1 Deprecated

7.1.1 Task 1

- ☒ "3. $\implies$ 1." If $AC_{\mathbb{T}}(\mathbb{T})$ terminates with $L(v) = v$ for all vertices v of $\mathbb{T}$, we know that, if there was a homomorphism $h : \mathbb{T} \to \mathbb{T}$, h would map each vertex v to itself. We see that h is obviously an automorphism, hence $\mathbb{T}$ must be a core.

$1 \Rightarrow 2$: Let $\mathbb{T}$ be a core. We assume there is another homomorphism $f \in End(\mathbb{T})$ with $f \neq id$.

Note, that every unique shortest path from v to w maps to the unique shortest path from $f(v)$ to $f(w)$, since f is an automorphism of a tree. It follows, that there must be a leaf u on which f is not the identity, because otherwise $f = id$.

Then we take the orbit of u and the paths in between, which induces a subtree $\mathbb{T}'$. Note that each vertex $v \in \mathbb{T}'$ lies on a path from $x \in Orb(u)$ to $y \in Orb(u)$ that f maps to the path from $f(x) \in Orb(u)$ to $f(y) \in Orb(u)$,... liegt auf! TODO we know $f(\mathbb{T}') \subseteq \mathbb{T}'$. Explizit: f is automorphismus von T'

... we know that every automorphism of a tree fixes either a vertex or an edge. Since we don't allow double-edges there must be a an inner **?** vertex $v \in \mathbb{T}'$ for which $f(v) = v$.

Now let $\mathbb{T} = v(T_1, T_2, .., T_k)$, where T_i are the components of $T - v$. We know that $T_a \to T_b$ for at least one pair T_a, T_b, where T_a contains u and T_b contains $f(u)$. We then construct an endomorphism h of $\mathbb{T}$ by taking f on T_a. For every other component we define h as id. It's easy to see, that h is non-injective.

However, this means that $\mathbb{T}$ can't be a core, which means our assumption was wrong and $End(\mathbb{T})$ cannot contain such a f, but only id.

7.2 Todo

7.2.1 TODO Use with$_{\text{capacity}}$ for vectors

7.2.2 TODO Replace Range with explicit boundaries

7.2.3 TODO Task 1 -

- 1. => 2.

☐ Construct homomorphism first, then argue that it's non-injective
☒ v(T1, T2, ..)? $\mathrm{T_i}$ sind Komponenten von $T - v$

7.2.4 TODO Switch to V(T) notation

7.2.5 TODO Add verbose flag

7.2.6 TODO Parallelize pruning-search

7.2.7 TODO Add a –conservative flag

$f(v_1, ..., v_n) \in \{v_1, ..., v_n\}$

7.3 Questions

- Are empty arms allowed?
- If conjecture is true, then singleton-arc-consistency can be used to check commutative polymorphisms in the same way like path-consistency can be used to check majority polymorphisms

7.4 Definitions

- Define path as shortest
- Define "AC derives id" as AC derives L(v) = v for every vertex v
- Define root as vertex that has degree 3

7.4.1 Polymorphism

7.4.2 Path

7.4.3 Triad

A *triad*, i.e., an orientation of a tree which has a single vertex of degree 3 and otherwise only vertices of degree 2 and 1.

7.4.4 AC

7.4.5 AC pruning search