Binary Commutative Polymorphisms of Core Triads

Michael Wernthaler

February 19, 2021

Contents

1 TODO Abstract

It has been known for a while that for a given digraph 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. **TODO** We describe an algorithm with various optimizations, that checks the existence of Siggers polymorphisms for triads up to a certain number/size of vertices/armlength?.

2 TODO Introduction

Let $H = (V, E)$ be a finite digraph. The *H-colouring problem* (also called the *constraint satisfaction problem for H*) is the problem of deciding for a given finite digraph 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.

In this paper we confirm the conjecture made by Jakub Bulin, that the triad $01001111, 0110000, 101000$ has no Siggers polymorphism and that all smaller triads do have a Siggers polymorphism. Moreover, we will present a new-found triad of the same size that doesn't have a Siggers polymorphism either. Lastly, we provide a proof of the non-existence of a binary-commutative polymorphism for our new triad, that isn't based on running a computer program, but understandable to humans.

2.1 TODO Organization of the paper

3 TODO Preliminaries

3.1 Graphs

A *directed graph* (also *digraph*) is a pair $G = (V, E)$ of disjunctive sets, where $E \subseteq V^2$. We call the elements of $V = V(G)$ *vertices* of G and $E = E(H)$ its *edges*. The graph G is called *finite* or *infinite* depending on whether $V(G)$ is finite or infinite. However, since this paper deals exclusively with finite digraphs, we will omit this and consider our graphs to be finite. Two vertices $x, y \in G$ are adjacent in G and are called neighbors of each other if $(x, y) \in E(G)$.

A *path* p (from u_1 to u_k in G) is a sequence $(u_1, \ldots, u_k)$ of vertices of G such that u_i is adjacent to u_{i+1} for $1 \leq i < n$. For the rest of the paper we consider *the path* from u_1 to u_k to be a shortest sequence that meets the requirements above.

If $P = x_0 ... x_{k-1}$ is a path and $k \geq 3$, then the graph $C := P+$ a circle.
$x_{k-1}x_0$

We also often denote a circle briefly by its (cyclic) sequence of vertices, so in the above example $C = x_0 ... x_{k-1} x_0$. The length of a circle is again the number of its channels. and we denote the circle of length k by C^k.

3.2 Trees

3.3 Triads

A *triad* is an orientation of a tree, which has a single vertex of degree 3 (also called *root*) and otherwise only vertices of degree 2 and 1.

3.4 Homomorphisms

Let $H = (V, E)$ be another graph. A *homomorphism* from G to H is a mapping $h : V(G) \to V(H)$ such that $(h(u), h(v)) \in E(H)$ whenever $(u, v) \in E(G)$. If such a homomorphism exists between G and H we say that G *homomorphically maps* to H, and write $G \to H$.

G is called *isomorphic* to H, written $G \simeq H$, if there exists a bijection $\phi : V(G) \to V(H)$ with $(x, y) \in E(G) \Leftrightarrow \phi(x)\phi(y) \in E(H)$ for all $x, y \in V$. We call such a mapping ϕ an *isomorphism*. We usually do not distinguish between isomorphic graphs, thus we write $G = H$ instead of $G \simeq H$.

An *endomorphism* of a graph H is a homomorphism from H to H. An *automorphism* of a graph H is an isomorphism from H to H. A finite graph H is called a *core* if every endomorphism of H is an automorphism. A subgraph G of H is called *a core of* H if H is homomorphically equivalent to G and G is a core.

3.5 Polymorphism

4 TODO The Arc-consistency Procedure

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 TODO AC

- The arc-consistency procedure is one of the most studied algorithms for solving constraint satisfaction problems. It found its first mentions in [51, 52] and is often referred to as the *consistency check procedure*.
- Let H be a finite digraph, and let G be an instance of $CSP(H)$.
- The idea is to maintain a list $L(v)$ of vertices from $V(H)$ for each vertex $v \in G$.
- Each element in the list of x represents a candidate for an image of x under a homomorphism from G to H.
- The procedure successively removes vertices from the lists by performing consistency checks.

- This done by following only two rules two rules:
  - remove u from $L(x)$, if there is no $v \in L(y)$ with $(u, v) \in E(H)$
  - remove v from $L(y)$, if there is no $u \in L(x)$ with $(u, v) \in E(H)$
- If eventually we can not remove any vertex from any list, the graph G together with the resulting lists is called *arc consistent*.
- Overall, there are three possible outcomes
  1. One list is empty A homomorphism from G to H does not exist
  2. Each list contains a single vertex Homomorphism from G to H
  3. Some list contain more than one vertex -> may or may not be a solution in this case, arc consistency isn't enough to solve the problem: we need to perform search

4.2 AC pruning search

There are cases in which AC does not solve CSP, then we need to perform search.

Initially we run AC_H on the input graph G. If the empty list is derived, we reject, otherwise we pick some vertex $x \in V(G)$ and set $L(x)$ to v for some $v \in L(x)$. Next, we run AC again and proceed recursively with the resulting lists. However, if AC_H now derives the empty list, we backtrack and remove v from $L(x)$. Finally, if AC didn't derive the emtpy list, after setting singleton lists for every vertex $x \in V(G)$, we get a homomorphism from G to H.

The algorithm described above defines a recursive tree-search procedure, that can give us exponential time savings, even though it's overall runtime is still exponential. We prefer DFS over BFS, since the search-tree is always finite and has no cycles.

4.3 Singleton AC

Singleton arc consistency is stronger than arc consistency. A domain value $a \in L(x)$ in a CSP instance I is *singleton arc consistent*, if the instance obtained from I by removing all domain values $b \in L(x)$ with $a \neq b$ can be made arc consistent without emptying any domain. A CSP instance is *singleton arc consistent* (SAC) if every domain value is singleton arc consistent.

4.4 TODO Node consistency

4.5 TODO Benchmarks

5 TODO Cores

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

Let v and w be two vertices of $\mathbb{T}$. 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 $f(f(u))$. We claim 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 cycle 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 [1].

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 $End(\mathbb{T}) = \{id\}$.

TODO Define "AC derives id" as AC derives $L(v) = v$ for every vertex v

5.2 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 (criterion? **TODO**), based upon which we can decide whether to discard triads at an earlier stage in the generation process (**TODO**). We need the following definitions as prerequisites:

TODO since our algorithm builds up triads from individual arms.

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 T by adding arms to it. In this case we say that T 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 **TODO**) for (making a statement **TODO**) about the homomorphic equivalence of 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.

Hence 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 in θ.

Let θ be a partial triad and let T be a triad derived from θ. The table below shows the relation of ACR and AC (when/being **TODO**) applied to θ.

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

Hence we can formulate the following lemma.

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

Finally, we can answer question 1.

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

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 {p_1, p_2} in armlist do
    if ACR_{p1p2}(p_1p_2) derived L(v) ≠ v for some vertex v then
        Drop the pair and cache the two indices
foreach triad 𝕋 = {p_1, p_2, p_3} 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
```

5.3 Optimizations

5.3.1 Complements

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

Proof: Let $H = (V, E)$ be a digraph and let $\bar{H} = (V, \bar{E})$ be the digraph 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 $\bar{H}$.

By lemma 3 we can now reduce the number of triads to look at by half. (**TODO**)Of each pair of triads, that form a complement of each other, our algorithm excludes the one, whose first edge of the first arm is an forward edge. As an example, consider the triads $1000, 11, 0$ and $0111, 00, 1$. We exclude the latter one, since its first arm starts with 0 (Is this example really needed?).

5.3.2 Allocation

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

Code excerpt:

```
// Estimates number of cores with armlength len for Vector allocation
fn num_cores_length(len: u32) -> u32 {
    (0.005 * (9 as u32).pow(len) as f32) as u32
}
```

6 TODO 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

6.1 Algorithm

There is a polynomial-time algorithm to decide whether a given digraph H has a siggers polymorphism.

The pseudo-code of the procedure can be found in Figure 3. Given H, we construct a new graph G as follows. We start from the second power H^2, and then contract all vertices of the form (u, v) and (v, u). Let G be the resulting graph. Note that there exists a homomorphism from G to H if and only if H has a binary-commutative polymorphism. To decide whether G has a homomorphism to H, we run AC_H on G. If AC_H rejects, then we can be sure that there is no homomorphism from G to H, and hence H has no binary-commutative polymorphism. Otherwise, we use the same idea as in the proof of **TODO** pick $x \in V(G)$ and remove all but one vertex u from $L(x)$. Then we continue with the execution of AC_H. If AC_H derives the empty list, we try the same with another vertex v from $L(x)$. If we obtain failure for all vertices in $L(x)$, then clearly there is no homomorphism from G to H, and we reject. Otherwise, if AC_H does not derive the empty list after removing all vertices but u from $L(x)$, we continue with another vertex $y \in V(G)$, setting $L(y)$ to $\{u\}$ for some $u \in L(y)$. We repeat this procedure

Algorithm 3: Algorithm for finding polymorphisms

Input: a finite digraph H
Output: is there a binary-commutative polymorphism for H

```
G ⟵ H^2
forall u, v ∈ V(H) do
    contract the vertices (u, v), (v, u) in G
if PC_H(G) derives the empty list then
    reject
foreach x ∈ V(G) do
    Found = False
    foreach u ∈ L(x) do
        foreach y ∈ V(G) do
            Let L'(y) be a copy of L(y)
        L'(u) ⟵ {u}
        Run PC_H(G) with the lists L'
        if PC_H(G) does not derive the empty list then
            foreach y ∈ V(G) do
                L(y) ⟵ L'(y)
            Found = True
    if Found = False then
        reject
accept
```

until eventually all lists are singleton sets $\{u\}$; the map that sends x to u is a homomorphism from G to H. In this case we accept. If there exists $x \in V(G)$ such that AC_H detects an empty list for all $u \in L(x)$ then the adaptation of AC_H for the precoloured CSP would have given an incorrect answer for the previously selected variable: AC_H did not detect the empty list even though the input was unsatisfiable. Hence, H cannot have a binary-commutative polymorphism. It is easy to see that the procedure described above has polynomial running time.

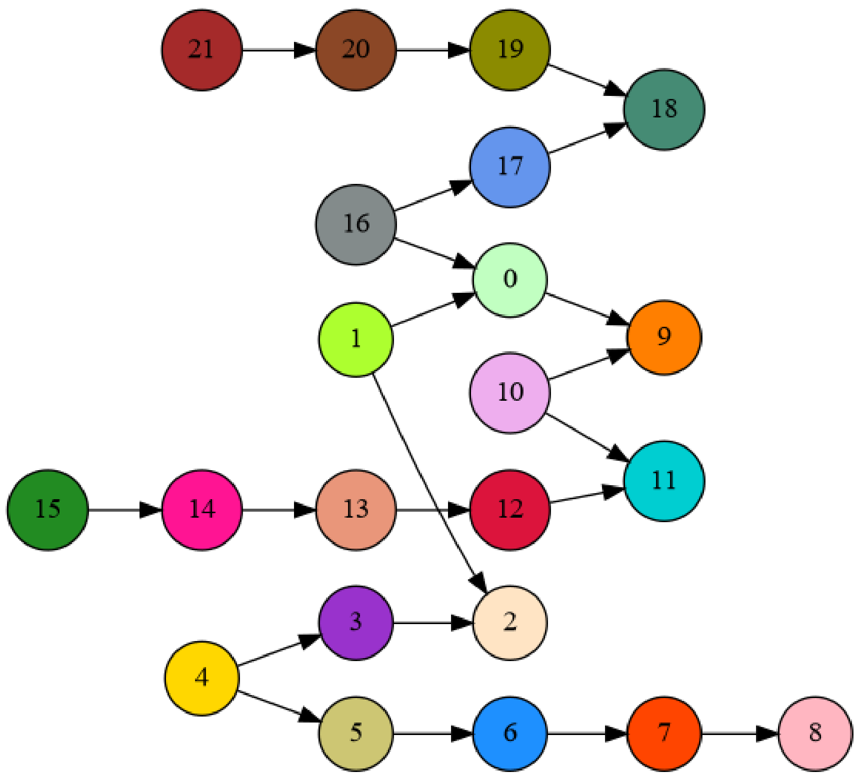

Figure 1: 01001111,0110000,101000

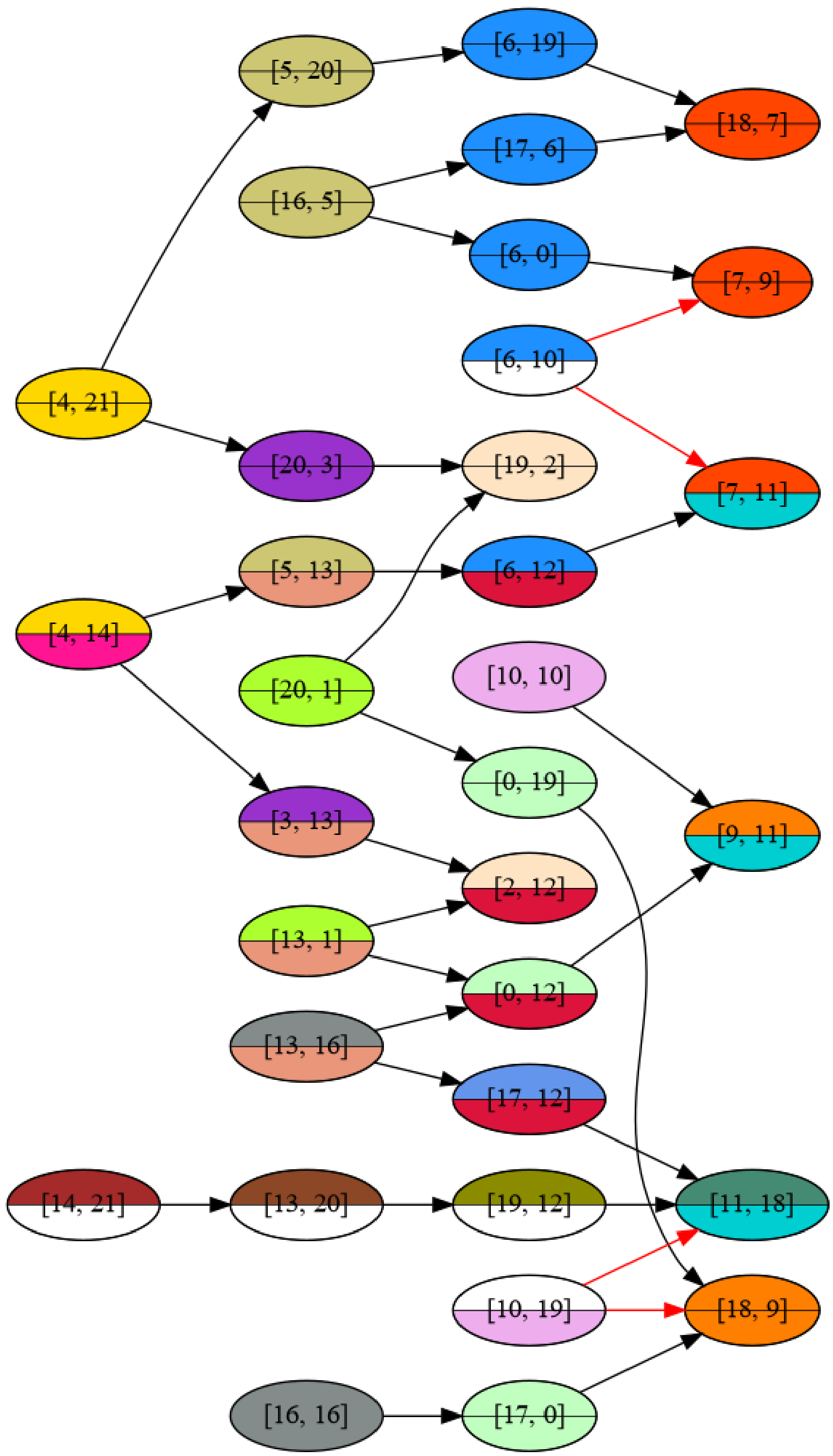

Figure 2: Reduced component of powergraph

6.2 Results

6.3 Proof

Lemma 4. Let $\mathbb{T}$ be a core tree and let h be a homomorphism from $\mathbb{T}^k$ to $\mathbb{T}$. Then $h(v, v, ..., v) = v$.

Proof: Let $\mathbb{T}$ be a core tree. By definiton of the productgraph, there is an edge between two vertices $(v, v, ..., v)$ and $(w, w, ..., w)$ of $\mathbb{T}^k$, if and only if $(v, w) \in \mathbb{T}$. Now let h be a homomorphism from $\mathbb{T}^k$ to $\mathbb{T}$. Let's assume that $h(v, v, ..., v) \neq v$ for at least one $v \in \mathbb{T}$. We then construct a non-injective endomorphism h' of $\mathbb{T}$ by defining $h'(v) = h(v, v, ..., v)$. By lemma 1 we know that such a homomorphism h' cannot exist. Hence our assumption was wrong, so h must map $(v, v, ..., v)$ to v.

Decreasing and increasing levels? We start at $[10, 10]$, which by lemma 4 must be mapped to 10. This enforces only two possible mappings on $[9, 11]$, either 9 or 11. In fact, either of the two choices will eventually lead to a contradiction, so we continue by making a case distinction. The first case, $[9, 11] \mapsto 9$, is represented by the top half colour of each vertex, whereas the second case, $[9, 11] \mapsto 11$, is represented by the bottom half colour.

Case 1. In this case we map $[9, 11]$ to 9. It's easy to see that mapping $[0, 12]$ to 10 isn't possible, since 10 has no incoming edge, which means we must map $[0, 12]$ to 0. For our further traversal we may choose either of the two adjacent vertices $[13, 1]$ or $[13, 16]$, of which we choose $[13, 1]$. Looking ahead we notice that to get to $[4, 14]$, we have to decrease by two levels. Therefore we traverse $[13, 1]$, $[2, 12]$, $[3, 13]$ and $[4, 14]$ by mapping them to 1, 2, 3 and 4, respectively. Continuing from $[4, 14]$, we see that have to be three forward edges, which forces $[7, 11]$ to be mapped to 7. Now the mapping for each of the following vertices is straightforward until we arrive at $[4, 21] \mapsto 4$. We notice the path from $[4, 21]$ to $[16, 16]$ has the exact same orientation as the path from 4 to 16. Therefore we must map the former to the latter, since we know that $[16, 16] \mapsto 16$. In particular, we map $[18, 9]$ to 9. As a consequence $[11, 18]$ must be mapped to 11, as the latter is the only vertex, from where we can traverse three consecutive backward edges. It follows that $[13, 6] \mapsto 13$ and we finally arrive at a contradiction, since there is no edge between 13 and 0.

Case 2. This is the case, in which we map $[9, 11]$ to 11. Again, $[0, 12] \mapsto 10$ isn't possible, since 10 has no incoming edge, therefore we map $[0, 12]$ to 12. For our further traversal we may choose either of the two adjacent vertices $[13, 1]$ and $[13, 16]$. Traversing in direction of $[13, 1]$ is straightforward, and we stop at $[7, 11] \mapsto 11$. Now we go back to $[0, 12]$ and start traversing in the other direction. It's easy to see that $[11, 18] \mapsto 11$. Next, we notice that the path from $[11, 18]$ to $[16, 16]$ has the exact same orientation as the path from 11 to 16. That leaves us no choice, but to map the former to the latter, since we know that $[16, 16] \mapsto 16$. In particular, we map $[18, 9]$ to 9. It follows, that $[20, 1] \mapsto 1$, since otherwise we wouldn't be able to decrease by two levels to reach $[4, 21]$. After mapping $[4, 21]$ to 4 the following mappings are straightforward until we map $[7, 9]$ to 7. Having mapped $[7, 11]$ to 11 earlier we see that for $[6, 10]$ there is no mapping s.t. 9 and 11 are adjacent.

7 TODO Acknowledgements

8 Notes

8.1 Deprecated

8.1.1 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.

8.2 Questions

- Search for finding a binary-commutative polymorphism, well known procedure? Theorem?
- Write "digraph" everywhere instead of "graph"?

8.3 Todo

8.3.1 Program

1. **TODO** Pull data from taurus
2. **TODO** Use with$_{\text{capacity}}$ for vectors
3. **TODO** Replace Range with explicit boundaries
4. **TODO** Add verbose flag
5. **TODO** Parallelize pruning-search
6. **TODO** Add a –conservative flag $f(v_1, ..., v_n) \in \{v_1, ..., v_n\}$
7. **TODO** Singleton-AC not equal to Pruning Search

8.3.2 Thesis

1. **TODO** Use academic-phrases for abstract
2. **TODO** Introduction
   - ☐ Explain organization of paper
3. **TODO** Measure time of algorithm for various triads
4. **DONE** Abstract
   - ☒ Digraphs
   - ☒ Algorithm with various optimizations
5. **TODO** Switch to V(T) notation
6. **TODO** Write down proof section 6

7. **DONE** Check siggers polymorphisms of both triads

   jakub 1:01:01
   michael 0:28:00

8.3.3 Style

- For instance, . . .
- We are now in position to. . .
- Folglich:
  - thus
  - consequently
  - hence
  - accordingly
  - therefore
  - then
  - by implication
  - as a consequence
  - as a result
- Exhibiting
- Notion
- Yield <-> give
- Seemingly
- Frequently
- Fix indentation
- Suppose/assume?
- Then we -> We then
- "does not derive the empty list" !

References

[1] M. Bodirsky. *Graph Homomorphsims and Universal Algebra*. 2020.