2018-06-07

Bounded Width Dichotomies in Constraint Satisfaction Problems

Liprandi, Maximiliano

UNIVERSITY OF CALGARY

Bounded Width Dichotomies in Constraint Satisfaction Problems

by

Maximiliano Liprandi

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN MATHEMATICS AND STATISTICS

CALGARY, ALBERTA

JUNE, 2018

Abstract

In this thesis we examine the connection between structures with bounded width, polymorphisms and pebble games. There has been extensive work on trying to prove the dichotomy conjecture for finite structures, namely, the Constraint Satisfaction Problem (CSP) of a finite structure is either in P or NP-complete. We are interested in finding classes in which a stronger dichotomy exists; namely, where every structure has either bounded width or a hard CSP. We call this kind of dichotomy a bounded width dichotomy. We will investigate properties of polymorphisms of structures with bounded width, and use this to look at classes of directed graphs with a bounded width dichotomy.

Preface

This dissertation is submitted for the degree of Doctor of Philosophy at the University of Calgary. It contains original work done under the supervision of Drs. Claude Laflamme and Robert Woodrow.

Acknowledgements

I would like to thank my supervisors, Claude Laflamme and Robert Woodrow, for their invaluable help and guidance, their patience and dedication, and their constant support and encouragement throughout my program.

I would also like to thank Richard Guy for his careful and dedicated help with my writing, and his never-ending support. Thanks to Gary MacGillivray for providing useful insight and suggestions for an earlier version of this thesis. Thanks to Jacobus Swarts for helping me find new ideas and lines of work. Thanks to Diane Fenton for her proofreading, her ideas, and her unconditional support. Thanks to Mark Girard for creating an excellent template and all his help with LaTeX.

Table of Contents

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol or abbreviation	Definition	
$\neg$	The negation operator.	11
$\vee$	The OR operator.	11
$\wedge$	The AND operator.	11
Δ, Γ	A relational structure.	3
$\Delta \models \varphi$	The structure Δ models the formula φ.	11
$\langle\Delta\rangle_{pp}$	The expansion of Δ by its pp-definable relations.	22
Π	A decision problem.	16
π	A Datalog program.	17
τ	A signature.	3
$\chi(G)$	The chromatic number of G.	4
$\mathbb{A}$	An algebra.	15
C	The directed graph $\vec{C}_3 \oplus \vec{C}_3$.	53
$\vec{C}_n$	A directed cycle on n vertices.	6
CSP(Δ)	The constraint satisfaction problem of Δ.	15
dom(h)	The domain of a function h.	43
$f^{(k)}(\bar{r_1}, ..., \bar{r_n})$	The k-tuple $(f(r_{1,1}, ..., r_{n,1}), ..., f(r_{1,k}, ..., r_{n,k}))$.	14
FALSE	The empty relation of arity 0.	3
$G = (V; E)$	A (directed) graph with vertices V and edges E.	3
$G_1 \oplus G_2$	The linear sum of G_1 and G_2.	8
$\mathcal{G}$	The class of graphs.	3
$\mathcal{H}$	A winning strategy for Duplicator.	43
I_v, J_v	Families of arcs of a circle.	30
Inv(A)	The relations that are preserved by A.	22
K_n	The complete graph on n vertices.	4
$\overline{K}_n$	The empty graph on n vertices.	4
$\vec{L}_n$	The directed graph $v_1 \oplus ... \oplus v_n$.	6
M_n	The Mycielski graph of order n.	9
$max\{x_1, ..., x_n\}$	The maximum element of $\{x_1, ..., x_n\}$.	23
$min\{x_1, ..., x_n\}$	The minimum element of $\{x_1, ..., x_n\}$.	23
N, S	Distinguished points on a circle.	30

Symbol or abbreviation	Definition	
$[n]$	The set $\{1, 2, ..., n\}$.	4
$N^+(v)$	The set of vertices with an edge from v.	57
$N^-(v)$	The set of vertices with an edge to v.	57
$N(u)$	The set of neighbours of a vertex u.	42
$N_a(\bar{x})$	The number of components of $\bar{x}$ equal to a.	54
$\mathrm{Pol}(\Delta)$	The set of polymorphisms of Δ.	14
pp	Positive primitive.	15
R	A relation symbol.	2
R^Δ	The relation in Δ associated with R.	4
R_n	The directed graph $\vec{C}_n$ with an edge reversed.	78
S_n	The symmetric group over $[n]$.	72
T_4	The smallest tournament that contains $\vec{C}_4$.	7
Tractable	Of a problem that is in P.	16
TRUE	The non-empty relation of arity 0.	3
$u \to v$	An edge of a directed graph. i.e. $(u, v) \in E$.	5
uv	An edge of a graph. i.e. $(u, v), (v, u) \in E$.	4
WNU	Weak near-unanimity.	24
$\bar{x}$	A k-tuple $(x_1, x_2, ..., x_k)$.	15
x^-	The vertex x with $x^- \to x$ in $\vec{C}_3$.	54
x^+	The vertex x with $x \to x^+$ in $\vec{C}_3$.	54

Chapter 1

Introduction

Constraint satisfaction problems (CSPs) can model a wide range of decision problems, and occur frequently in theoretical computer science. CSPs with finite templates have been extensively studied. In 1978, Schaefer [39] showed that every CSP of a Boolean (two-element) template is either tractable or NP-complete. Hell and Nešetřil showed in [22] that this "dichotomy" property is also true for simple graphs. One of the central problems in the study of the complexity of CSPs with finite templates is the Dichotomy Conjecture posed by Feder and Vardi in their seminal paper [18] in 1999, which states that every finite relational structure has either a tractable CSP or an NP-complete one. An algebraic approach has proved helpful in the study of this conjecture. An equivalent dichotomy conjecture was stated in [12] that ties the dichotomy to the existence of a certain kind of polymorphism in the structure.

We will examine the complexity of CSPs of finite structures by focusing on structures with bounded width, and the conditions that can be found to determine whether a structure has bounded width. We will use algebraic tools, as well as certain pebble games, which have a strong connection to the notion of bounded width. We will look for classes of directed graphs where a bounded width dichotomy exists. For this we will study known classes with a traditional "P versus NP-complete" dichotomy, and determine whether a bounded width

dichotomy also exists.

In Chapter 1, we describe the definitions and concepts that we will use in this thesis, as well as give illustrative examples to help the readers to familiarize themselves with these concepts. In Chapter 2, we will examine the connection between the complexity of CSPs and the existence of certain polymorphisms. We will give some useful results for determining whether a structure has bounded width by finding special polymorphisms. In Chapter 3, we will examine pebble games and their connection to Datalog and structures with bounded width. In Chapter 4, we will look at classes of directed graphs that exhibit a bounded width dichotomy. We will prove that the class of tournaments, and more generally the class of connected locally semi-complete graphs, have a bounded width dichotomy. We will also show that transitive directed graphs have bounded width, and give different proofs utilizing Datalog and pebble games. We will give an analysis of directed graphs of at most four vertices that do not admit a binary, conservative WNU polymorphism, of which two have bounded width, and analyze a generalization of these two directed graphs. We found that some of the work we had done for tournaments had already been done in [3]. However, we employ different methods in our proofs and refine some of the results in [3]. We will discuss open questions and further research in Chapter 5.

1.1 Relational structures

In this thesis we will examine the CSP of certain relational structures. We introduce the basic notions associated with relational structures and provide simple examples for these concepts.

Definition 1.1. Given a set D, a **relation R of arity k on D** is a subset of D^k. We write $R(x_1, ..., x_k)$ for the atomic sentence $(x_1, ..., x_k) \in R$.

There are two relations of arity zero (since R^0 has only one element) which can be thought

of as boolean constants, and will often be referred to as **FALSE** (for the empty relation) and **TRUE** (for the non-empty relation).

Definition 1.2. A **signature** τ is a set $\tau = \{R_i\}, i \in I$ of relation symbols R_i, each with an associated arity k_i.

Definition 1.3. A **τ-structure** $\Delta = (D; \{R_i\}_{i \in I})$ is a set D (its domain) together with relations R_i of arity k_i on D.

Example 1.4. The structure $NAE = (\{0,1\}; R)$ with $R = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\}$ is a τ-structure with signature $\tau = \{R\}$ and domain $\{0,1\}$. The relation R has arity 3. The name of the structure stands for "Not All Equal", since its relation R contains all triples where not every component is the same. Structures with 2-element domains are usually referred to as **Boolean**.

Example 1.5. We can think of the class $\mathcal{G}$ of graphs as a class of τ-structures, where the signature $\tau = \{E\}$ consists of a single binary relation symbol E that represents the edge relation of graphs. A structure from $\mathcal{G}$ is a graph $G = (V; E)$ with domain V, which corresponds to the vertices of G, and with a binary, symmetric relation E that corresponds to the edges of G. For a more detailed description of graphs see Section 1.2.

Unless noted, we will not distinguish a structure $\Delta = (D; \{R_i\}_{i \in I})$ from its domain D. We will therefore use $d \in \Delta$ to refer to an element $d \in D$.

In this thesis we will study relational structures $\Delta = (D; \{R_i\}_{i \in I})$ where D is a finite set. Unless otherwise stated, we will assume that relational structures have finite domains.

Definition 1.6. Given a τ-structure $\Delta = (D; \{R_i\}_{i \in I})$, a τ-structure $\Delta' = (D'; \{R_i'\}_{i \in I})$ is a **substructure of** Δ if $D' \subseteq D$ and $R_i' \subseteq R_i$ for all i. We say $\Delta' = (D'; \{R_i'\}_{i \in I})$ is an **induced substructure of** Δ if, in addition, $R_i' = R_i \cap (D')^{k_i}$.

Example 1.7. Let $D = \{a,b,c\}, D' = \{a,b\}, R = \{(a,b)(b,c)(c,a)\}, R' = \emptyset, R'' = \{(a,b)\}$. Then $\Delta' = (D'; R')$ is a substructure (but not an induced substructure) of Δ, while $\Delta'' = (D', R'')$ is an induced substructure (and therefore a substructure) of Δ.

We will not usually distinguish between a relation symbol R and its associated set in a given structure. If we are considering two τ-structures Δ and Γ, we will write R^{Δ}, R^{Γ} for the associated relations in each structure.

1.2 Graph theory

We define graphs and directed graphs as relational structures in the following way:

Definition 1.8. A (simple) **graph** is a structure $(V; E)$ where E is a symmetric, irreflexive binary relation. If $(v_1, v_2) \in E$, we say v_1 is connected to v_2 (and vice-versa), and we write $v_1 v_2 \in E$.

When working with graphs, since their edge relation is symmetric, we will sometimes write only $(v_1, v_2) \in E$ to represent $(v_1, v_2), (v_2, v_1) \in E$. It is assumed that in a graph $(v_1, v_2) \in E$ implies $(v_2, v_1) \in E$.

Definition 1.9. Given a graph G, the **chromatic number of** G, denoted by $\chi(G)$, is the least number of colours needed to colour the vertices of G so that no adjacent vertices share the same colour. Any finite graph can be coloured by assigning a different color for each vertex. Thus for $G = (V; E)$, we have that $\chi(G) \leq |V|$.

Example 1.10. Given $n \geq 1$, the complete graph on n vertices, denoted by K_n, is given by $K_n = (V; E)$, where $V = [n]$ and $E = V^2 \setminus \{(v, v) | v \in V\}$. The empty graph on n vertices, denoted by $\overline{K_n}$, is given by $\overline{K_n} = (V; E')$, where $V = [n]$ and E' is a binary relation with $E' = \emptyset$. The graph K_n has chromatic number n, while the graph $\overline{K_n}$ has chromatic number 1. The graphs K_2, K_3 and K_4 are shown in Figure 1.1.

Example 1.11. A graph $G = (V; E)$ is **bipartite** if $V = A \cup B$ for some A and B such that there is no edge between any two elements of A or any two elements of B. The graph K_2 is bipartite, while K_n is not, for $n \geq 3$. The graph $\overline{K_n}$ is bipartite for any n.

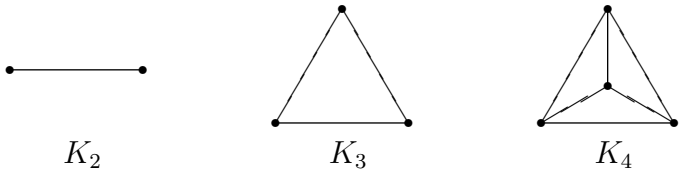


Figure 1.1: The complete graphs K_2, K_3, K_4

Definition 1.12. Given a graph $G = (V; E)$, a **path** is a sequence of vertices $v_1, ..., v_n \in G$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq 1 < n$. We say that $v_1, ..., v_n$ is a path from v_1 to v_n, with **length** $n - 1$.

Definition 1.13. A graph $G = (V; E)$ is **connected** if, for every pair of vertices $u, v \in V$, there is a path $u, x_1, x_1x_2, ..., x_n, v$ connecting them.

Definition 1.14. A (simple) **directed graph** is a structure $(V; E)$ where E is an asymmetric, irreflexive binary relation. If $(v_1, v_2) \in E$, then we say that v_1 is **connected to** v_2, or that v_2 is **connected from** v_1, and we write $v_1 \to v_2$.

We will mostly focus on simple graphs. Unless specified, we will assume graphs and directed graphs to be simple. Note that a non-simple graph may contain loops, so its edge relation is not irreflexive, and a non-simple directed graph may contain loops and pairs of edges $u \to v, v \to u$, so its edge relation is not irreflexive or asymmetric.

Definition 1.15. Given a directed graph $G = (V; E)$, a **directed path** is a sequence of vertices $v_1, \cdots, v_n \in G$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq 1 < n$. We say that $v_1, \cdots, v_n$ is a directed path from v_1 to v_n, with **length** $n - 1$.

Definition 1.16. Let G be a directed graph. A vertex $v \in G$ is a **source** if there is no vertex $u \in G$ such that $u \to v$. Similarly, v is a **sink** if there is no vertex $u \in G$ such that $v \to u$.

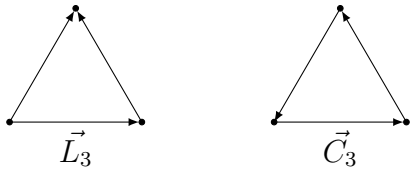


Figure 1.2: The directed graphs $\vec{L_3}$ and $\vec{C_3}$.

Example 1.17. The directed graph $\vec{L_n}$ has vertices $u_1, \cdots, u_n$, and $u_i \to u_j$ if and only if $i < j$. This graph represents a strict linear order on n elements. The vertex u_1 is a source and the vertex u_n is a sink. The directed graph $\vec{C_n}$ has vertices $v_1, \cdots, v_n$ and edges $v_1 \to v_2, \cdots, v_{n-1} \to v_n, v_n \to v_1$. We call $\vec{C_n}$ the directed cycle on n vertices. The directed cycle $\vec{C_n}$ has no sources or sinks. The directed graphs $\vec{L_3}$ and $\vec{C_3}$ are shown in Figure 1.2.

Example 1.18. A tournament T on n vertices is a complete simple directed graph; that is, $T = (V; E)$ where $V = [n]$ and for all $v_1, v_2 \in V$ with $v_1 \neq v_2$, exactly one of $(v_1, v_2), (v_2, v_1)$ is in E.

Definition 1.19. A directed graph is **weakly connected** if its associated graph (the graph obtained by transforming directed edges into edges) is connected.

Definition 1.20. A directed graph G is **strongly connected** if for every pair of vertices $u, v \in G$, there is a path $u \to x_1 \to \cdots \to x_n \to v$ from u to v and a path $v \to y_1 \to \cdots \to y_m \to u$ from v to u, where x_i, y_i are vertices of G. Note that m or n could be 0.

Every strongly connected graph is also weakly connected, but the converse is not always true.

Example 1.21. The graph $\vec{L_3}$ (Figure 1.2) is weakly connected, but not strongly connected. The graph $\vec{C_3}$ (Figure 1.2) is strongly connected (and thus weakly connected).

Definition 1.22. A directed graph G with n vertices is **pancyclic** if it contains a directed cycle of length k for every $3 \leq k \leq n$.

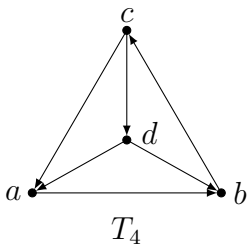


Figure 1.3: The tournament T_4

Example 1.23. The tournament T_4 shown in 1.3 has a 3-cycle $a \to b \to c \to a$ and a 4-cycle $a \to b \to c \to d \to a$. It is therefore pancyclic. Note that this implies that T_4 is also strongly connected.

We provide a theorem that will be used in Chapter 4, that shows that if a tournament contains a cycle of length n, then it also contains cycles of length $3 \leq k \leq n$.

Theorem 1.24. *[21] If a tournament T is strongly connected, then it is pancyclic.*

Proof. Assume that T has $n \geq 3$ vertices. We proceed by induction. Assume that T is strongly connected. Then it is not transitive, and so it has a cycle of length 3. Now assume that it has a cycle $C = v_1 \to v_2 \to \cdots \to v_k \to v_1$ of length $k < n$. We will prove it has a cycle of length $k + 1$. There are two cases: either there exists a vertex $u \in T$ not in C that is connected to a point in C and connected from a point in C, or there is no such vertex u.

Case 1. Suppose that there is a vertex $u \in T$ not in C that is connected from v_i and to v_j. Let v_ℓ be the first vertex in C, starting from v_i, that is connected from u. Then $v_1 \to \cdots \to v_{\ell-1} \to u \to v_\ell \cdots \to v_k \to v_1$ is a cycle of length $k + 1$.

Case 2. If no such u exists, then we can partition all vertices in T not in C into two subsets U_1 and U_2 such that every vertex in U_1 is connected to every vertex in C, and every vertex in U_2 is connected from every vertex in C. Since there is no u as in case 1 (and since $k < n$), one of these sets must be non-empty, and so both of them must be non-empty, since T is strongly connected. Furthermore, there must exist $u_1 \in U_1$ and $u_2 \in U_2$ such that u_2 is connected to u_1. Then $u_1 \to v_1 \to \cdots \to v_{k-1} \to u_2 \to u_1$ is a cycle of length $k + 1$. □

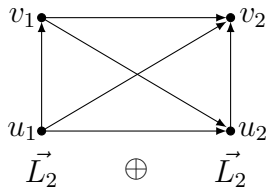


Figure 1.4: The directed graph $\vec{L_2} \oplus \vec{L_2}$.

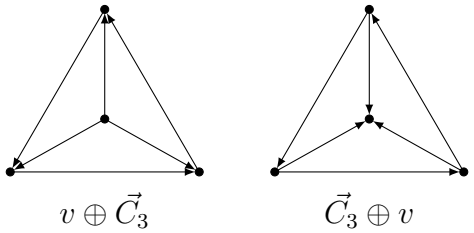


Figure 1.5: The directed graphs $v \oplus \vec{C_3}$ and $\vec{C_3} \oplus v$.

Definition 1.25. Let $G_1 = (V_1; E_1), G_2 = (V_2; E_2)$ be disjoint directed graphs. The **linear sum of** G_1 **and** G_2, denoted by $G_1 \oplus G_2$, is the directed graph $H = (V; E)$, where $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup \{(v_1, v_2) | v_1 \in V_1, v_2 \in V_2\}$.

For a given vertex v, we will denote $v \oplus G_1 = H \oplus G_1$, $G_1 \oplus v = G_1 \oplus H$, where $H = (\{v\}; E)$ and $E = \emptyset$.

Example 1.26. Given the directed graph $\vec{L_2} = (\{u, v\}; \{(u, v)\})$, the linear sum $\vec{L_2} \oplus \vec{L_2}$ is the directed graph

$$\vec{L_2} \oplus \vec{L_2} = (\{u_1, u_2, v_1, v_2\}; \{(u_1, v_1), (u_2, v_2), (u_1, u_2), (u_1, v_2), (v_1, u_2), (v_1, v_2)\})$$

as shown in Figure 1.4. By renaming the vertices u_1, v_1, u_2, v_2 into u_1, u_2, u_3, u_4, in that order, we can see that $\vec{L_2} \oplus \vec{L_2}$ coincides with the directed graph $\vec{L_4}$.

Note that linear sum is not commutative.

Example 1.27. Let v be a vertex and $\vec{C_3}$ the directed cycle on 3 vertices. Then the graphs $v \oplus \vec{C_3}$ and $\vec{C_3} \oplus v$ (see Figure 1.5) are not copies of each other, since the first has a vertex with 3 outgoing edges, while the second does not.

1.2.1 The Mycielski graphs

The Mycielski graphs, denoted by M_n where $n \geq 2$, is a family of graphs which are constructed recursively and have the properties that they are triangle-free (i.e. it does not contain the graph K_3 as an induced subgraph), and increasing in chromatic number with n. We will use these graphs in Chapter 3 to show certain properties of pebble games (see Section 3.2).

We construct the family as follows. The starting graph is $M_2 = K_2$, a single edge. We obtain M_{n+1} from M_n in the following way:

Let $u_1, \cdots, u_r$ be the vertices of M_n. Then M_{n+1} will have $m = 2r+1$ vertices, $v_1, \cdots, v_r$, $w_1, \cdots, w_r, z$. The edges of M_{n+1} will be $v_i v_j, v_i w_j, w_i v_j$ for each edge $u_i u_j$ of M_n, as well as $z w_i$ for $i = 1, \cdots, r$. The graphs M_2, M_3 and M_4 are shown in Figure 1.6. The graph M_5 has 23 vertices and 71 edges. The Online Encyclopedia of Integer Sequences contains the following sequences related to the Mycielski graphs:

- A266550 Independence number, i.e., number of vertices. [40]
- A122695 Number of edges. [17]
- A137890 Number of (directed) Hamiltonian paths. [41]
- A143247 Number of (directed) Hamiltonian circuits. [42]
- A234625 Number of undirected cycles. [43]
- A193148 Number of spanning trees. [44]
- A287432 Number of connected dominating sets. [45]

Proposition 1.28. *[36] The Mycielski graph M_n is triangle-free and has chromatic number n.*

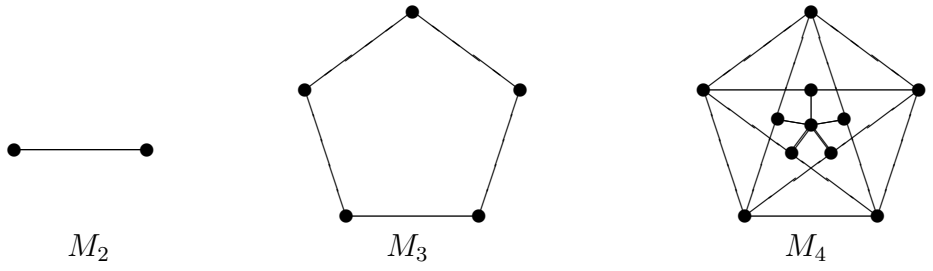


Figure 1.6: The Mycielski graphs M_2,M_3,M_4

Proof. We proceed by induction. Clearly $M_2 = K_2$ is triangle-free and has chromatic number 2. Let $k \geq 2$ and suppose M_k is triangle-free and has chromatic number k. Let $v_1, \cdots, v_r$,$w_1, \cdots, w_r, z$ be the vertices of M_{k+1} as in the construction above. Since by construction $w_1, \cdots, w_r$ is an independent set, M_{k+1} does not have any triangles with two or more vertices in $\{w_1, \cdots, w_r\}$. Therefore M_{k+1} does not have a triangle with z as one of its vertices. We only need to verify that M_{k+1} does not have triangles of the form $v_i v_j w_\ell$ for some $i, j, \ell \in [r]$. If these vertices formed a triangle, then by construction M_k would have the edges $u_i u_j, u_i u_\ell, u_j u_\ell$, which is a contradiction. Therefore M_{k+1} is triangle-free. We are left with proving that M_{k+1} has chromatic number $k+1$. Notice that $M_{k+1} \setminus \{z\}$ can be coloured with k colours, by taking a k-colouring of M_k and colouring v_i, w_i with the same colour as u_i. But any colouring of $M_{k+1} \setminus \{z\}$ requires at least k colours for the set $\{w_1, \cdots, w_r\}$, since w_i and v_i have the same set of neighbours, and the subgraph induced by $\{v_1, \cdots, v_r\}$ is isomorphic to M_k. Therefore $M_{k+1} \setminus \{z\}$ has chromatic number k, and since z is adjacent to $w_1, \cdots, w_r$, it requires a new colour. Therefore M_{k+1} has chromatic number $k+1$.

□

1.3 Logic

We refer the reader to [24] for an introduction to logic and model theory. We will give informal explanations for the basic concepts that we will use.

Definition 1.29. An **atomic formula φ over τ**, called an atomic τ-formula, is an expression of the form $R(x_1, \cdots, x_k)$, where $R \in \tau$ is of arity k. The atomic formula φ expresses $(x_1, \cdots, x_k) \in R$.

Definition 1.30. The **set of formulas over τ** is the smallest set containing all the atomic τ-formulas such that if φ, ψ are formulas and x is a variable, then

1. $\neg\varphi$ is a formula,
2. $\varphi \vee \psi$ and $\varphi \wedge \psi$ are formulas,
3. $\exists x\ \varphi$ and $\forall x\ \varphi$ are formulas.

Example 1.31. Given a graph $G = (V; E)$, where $\tau = \{E\}$, an atomic formula is of the form $E(x_1, x_2)$ and represents the statement "there is an edge between x_1 and x_2". An example of a τ-formula is

$$\varphi(x_1, x_2) = \exists x(E(x_1, x) \wedge E(x, x_2)),$$

which represents the statement "there is a path of length 2 between x_1 and x_2".

Definition 1.32. Given a τ-structure Δ, a τ-formula $\varphi(x_1, \cdots, x_n)$, and elements $t_1, \cdots, t_n \in D$, we say Δ **models** $\varphi(t_1, \cdots, t_n)$ (and write $\Delta \models \varphi(t_1, \cdots, t_n)$) if $\varphi(t_1, \cdots, t_n)$ is true in Δ.

Example 1.33. Let $G = (V; E)$ where $V = \{u, v, w\}$ and E contains the edges uv, vw but not uw. Let $\varphi(x_1, x_2)$ be the formula from Example 1.31. Then G does not model $\varphi(u, v)$, since there is no path of length 2 between u and v.

1.4 Homomorphisms

Definition 1.34. Given τ-structures Δ and Γ, a function $\varphi : \Delta \to \Gamma$ is a **homomorphism** if it preserves the relations in τ; that is, for all $R \in \tau$ (of, say, arity k), if $(d_1, \cdots, d_k) \in R^\Delta$ then $(\varphi(d_1), \cdots, \varphi(d_k)) \in R^\Gamma$.

Example 1.35. Consider $\Delta = K_{3,3}$, the complete bipartite graph on 3 and 3 vertices, with its independent sets of vertices labelled $\{a_1, a_2, a_3\}, \{b_1, b_2, b_3\}$ respectively, and $\Gamma = K_2$, a single edge, with vertices labelled x, y. Then the function $\varphi : \Delta \to \Gamma$, with $\varphi(a_i) = x, \varphi(b_j) = y$ for all $i, j \in [3]$, is a homomorphism.

Example 1.36. Let G be a graph and K_n be the complete graph on n vertices. A homomorphism from G to K_n will produce a colouring of G with n colours by using a different colour for the pre-image of each vertex of K_n. Additionally, every colouring of G with n vertices gives a homomorphism from G to K_n by identifying each colour with a vertex of K_n and mapping every vertex of G coloured with that colour to the associated vertex of K_n. We see that there is a homomorphism from G to K_n if and only if G can be coloured with n colours. This gives us a way to define the chromatic number of a graph using homomorphisms: the graph G has chromatic number n if there is a homomorphism from G to K_n, but there is no homomorphism from G to K_{n-1}.

Definition 1.37. An **isomorphism** is a bijective homomorphism φ such that φ^{-1} is also a homomorphism. A homomorphism $\varphi : \Delta \to \Delta$ is called an **endomorphism**. An **automorphism** is an endomorphism that is an isomorphism.

Let Δ be a relational structure and $f : \Delta \to \Delta$ a bijective endomorphism of Δ. Since Δ is finite, f must be a permutation of the elements of Δ and so $f^n = f \circ \cdots \circ f$ will be the identity operation for some n. Therefore f is an isomorphism.

Example 1.38. Let K_n be the complete graph with vertices $u_1, \cdots, u_n$. Any permutation $\sigma : [n] \to [n]$ gives an isomorphism f_σ of K_n by $f_\sigma(u_i) = u_{\sigma(i)}$. Let $\bar{K_n}$ be the empty graph

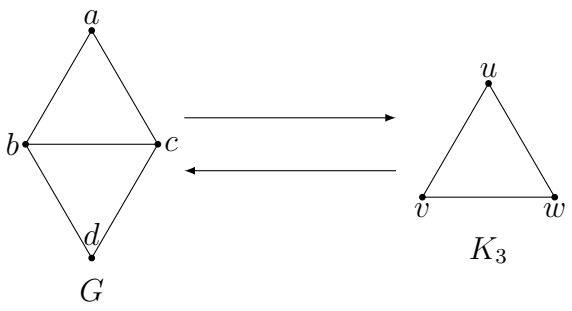


Figure 1.7: A graph G with core K_3.

on the vertices $v_1, \cdots, v_n$. The homomorphism $g : \bar{K_n} \to \bar{K_n}$ given by $g(v_i) = v_1$ is an endomorphism, but not an automorphism (since g is not bijective).

Definition 1.39. Two structures Δ and Γ are **homomorphically equivalent** if there exist homomorphisms $\varphi_1 : \Delta \to \Gamma$ and $\varphi_2 : \Gamma \to \Delta$.

Definition 1.40. A finite structure Δ' is a **core** if every endomorphism $\varphi : \Delta' \to \Delta'$ is an automorphism.

Example 1.41. The complete graph K_n is a core. To see this, note that any endomorphism $\varphi : K_n \to K_n$ must be injective, since there is an edge between every pair of vertices of K_n. Any injective endomorphism of a finite structure is also surjective, so it is an automorphism.

Definition 1.42. Given a finite τ-structure Δ, a τ-structure Δ' **is a core of Δ** if Δ' is a core and Δ' is homomorphically equivalent to Δ.

Example 1.43. Consider the graph G illustrated in Figure 1.7. Note that there is a homomorphism from G to K_3 obtained by mapping the vertices a, d to the same vertex in K_3, and there is a natural inclusion from K_3 to G. Therefore G and K_3 are homomorphically equivalent. Since K_3 is a core, K_3 is a core of G.

We can verify that every finite τ-structure Δ has a core. If every endomorphism of Δ is an automorphism, then Δ is a core. Otherwise, there is an endomorphism $f : \Delta \to \Delta$ that is not surjective. Let Δ_1 be the induced substructure of Δ whose domain is the image of

f. Then Δ_1 is homomorphically equivalent to Δ. If Δ_1 is not a core, then we repeat this process until we find a core Δ_n (this will always happen since Δ is finite and one-element structures are cores.) Since Δ_n is also homomorphically equivalent to Δ, Δ_n is the core of Δ. Note that the core of Δ is unique up to isomorphism.

1.5 Polymorphisms and algebra

Given an operation $f : D^n \to D$ and k-tuples $\bar{r}_1, \bar{r}_2, \cdots, \bar{r}_n \in D^k$, where $\bar{r}_i = (r_{i,1}, \cdots, r_{i,k})$, we denote by $f^{(k)}(\bar{r}_1, \cdots, \bar{r}_n)$ the k-tuple

$$(f(r_{1,1}, r_{2,1}, \cdots, r_{n,1}), \cdots, f(r_{1,k}, r_{2,k}, .., r_{n,k})).$$

Definition 1.44. An operation $f : \Delta^n \to \Delta$ is a **polymorphism** if, for all $R \in \Delta$ (of arity k), the following is true: if $\bar{r}_1, \cdots, \bar{r}_n \in R$, then $f^{(k)}(\bar{r}_1, \cdots, \bar{r}_n) \in R$.

Example 1.45. Let $\Delta = \vec{L}_m$ ($\vec{L_m} = (\{v_1, \cdots, v_m\}; \{(v_i, v_j) | i < j\})$, see Example 1.17) and let $f : \Delta^n \to \Delta$ be defined by $f(x_1, \cdots, x_n) = u_d$ where $d = \min\{i \mid u_i \in \{x_1, \cdots, x_n\}\}$. In other words, $f(x_1, \cdots, x_n)$ is the minimum of $\{x_1, \cdots, x_n\}$ with respect to the ordering $u_1 < \cdots < u_m$. Then f is a polymorphism. To see this, suppose we have tuples $r_1 = (x_1, y_1), \cdots ., r_n = (x_n, y_n) \in E$; i.e. $x_1 \to y_1, \cdots, x_n \to y_n$. We need to prove that $f^{(2)}(r_1, \cdots, r_n) \in E$; i.e. $f(x_1, \cdots, x_n) \to f(y_1, \cdots, y_n)$. If $f(x_1, \cdots, x_n) = u_d = x_i$ for some i and $f(y_1, \cdots, y_n) = u_e = y_j$ for some j, then $d \leq d'$ where $u_{d'} = x_j$, and $d' < e$, since $x_j \to y_j$. Therefore $d < e$ and so $u_d \to u_e$, as needed.

Polymorphisms can be composed in the following way: if $f : D^n \to D$, and $g_1, \cdots, g_n : D^k \to D$ are polymorphisms of Δ, then $f(g_1, \cdots, g_k) : D^k \to D$ is given by

$$f(g_1, \cdots, g_k)(\bar{x}) = f(g_1(\bar{x}), \cdots, g_n(\bar{x})).$$

We denote the class of polymorphisms of Δ by $\mathrm{Pol}(\Delta)$.

Definition 1.46. An operation $\pi : D^n \to D$ is a **projection** if there exists $i \in [n]$ such that

$$\pi(x_1, \cdots, x_n) = x_i \text{ for all } x_1, \cdots, x_n \in D.$$

The set of polymorphisms of Δ contains all projections of Δ and is closed under composition. We call a set with these properties a **clone**.

Definition 1.47. An **algebra** is an ordered pair $\mathbb{A} = (A; F)$, where A is a non-empty set and F is a family of finitary operations on A. The set A is called the **universe** of $\mathbb{A}$.

Definition 1.48. A first-order formula ψ over τ is **primitive positive** if it is of the form

$$\psi(\bar{x}) = (\exists \bar{y})(\phi_1(\bar{x}, \bar{y}) \wedge \cdots \wedge \phi_l(\bar{x}, \bar{y})),$$

where the formulas ϕ_i are atomic; that is, they are of the form "$R(z_1, \cdots, z_k)$" for some $R \in \tau$, or of the form "$z_1 = z_2$". We call ψ a **pp-formula**.

1.6 Constraint Satisfaction Problems

Definition 1.49. Given a $\tau-$structure Δ, the **Constraint Satisfaction Problem** (or **CSP**) associated with Δ is the following decision problem: given a finite $\tau-$structure $\mathcal{S}$, is there a homomorphism $f : \mathcal{S} \to \Delta$? The structure Δ is called the **template** of the CSP, while $\mathcal{S}$ is an **instance** of it.

Example 1.50. Let $\Delta = K_2$, the graph consisting of two vertices joined by an edge. Then CSP(Δ) is the following decision problem: given a finite graph G, is there a homomorphism from G to Δ? Suppose there exists a homomorphism f. Then, the pre-image of each vertex of Δ under f must be an independent set, since otherwise the edge relation would not be preserved. Therefore, the vertices of G form two independent sets, with no restrictions

between vertices from different sets. We see that there is a homomorphism between G and Δ if and only if G is bipartite.

1.7 Computational complexity

When studying CSPs, we are interested in their complexity; that is, how fast we can answer the decision problem. Given an instance S of CSP(Δ), we think of its size as the number of elements of S plus the number of tuples in its relations.

We refer the reader to [1] and [38] for an introduction to computational complexity and complexity classes. We will give informal explanations for the complexity classes to which we will most commonly refer.

Definition 1.51. A decision problem Π of size n is **in P** if it can be solved by an algorithm that needs at most $Q(n)$ steps, for some polynomial Q. In this case we say that Π is **tractable**.

Example 1.52. The problem of deciding whether a graph $G = (V; E)$ is triangle-free is in P, since it is sufficient to check if every 3-element subset of V forms a triangle. This can be done within $Q(n)$ steps, where Q is a polynomial of degree 3 and n is the size of G.

Example 1.53. The problem CSP(K_2) described in Example 1.50 is tractable, since it is equivalent to deciding whether there is an odd cycle in a given graph G, which is well-known to be a tractable problem.

Definition 1.54. A decision problem of size n is **in NP** if any YES instance to it can be verified within $Q(n)$ steps, for some polynomial Q. In particular, decision problems in P are also in NP.

Example 1.55. The problem of deciding whether the vertices of a graph $G = (V; E)$ can be coloured using 3 colours is in NP, since to check whether a given colouring of G with 3 colours is valid it is sufficient to check that every edge of G has vertices of different colours.

This can be done within $Q(n)$ steps, where Q is a polynomial of degree 2 and n is the size of G.

Definition 1.56. A decision problem Π is **NP-complete** when it is in NP and it is at least as hard as any other problem in NP. That is, given a problem Σ in NP, there is a polynomial-time reduction from Π to Σ.

Example 1.57. The problem of deciding whether the vertex of a graph $G = (V; E)$ can be coloured using n colours is NP-complete for every $n > 2$. Note that this problem is equivalent to CSP(K_n).

1.8 Datalog and bounded width

Datalog is the language of logic programs without function symbols. A datalog program is a finite set of rules of the form

$$R_0(x_1^0, \cdots, x_\ell^0) : -R_1(x_1^1, \cdots, x_{k_1}^1), \cdots, R_r(x_1^r, \cdots, x_{k_r}^r).$$

The relation tuple $R_0(x_1^0, \cdots, x_\ell^0)$ is the head of the rule, while the tuples $R_1(x_1^1, \cdots, x_{k_1}^1)$, $\cdots$, $R_r(x_1^r, \cdots, x_{k_r}^r)$ make the body of the rule. The relation symbols that only appear in the body of the rules are called **EDB**s (extensional databases), and the rest of the symbols are called **IDB**s (intentional databases).

Let $\Delta = (D; \mathcal{R})$ and π be a datalog program with IDBs $P_1, \cdots, P_m$ and EDBs $R_1, \cdots, Rn \in \mathcal{R}$. An evaluation of π on Δ creates a new structure Δ' in steps $i = 0, 1, \cdots$ according to the rules of the program, in the following way: on step 0, we start with a structure $\Delta^{(0)}$ with domain $D^{(0)} = D$ and relations $R_j^{(0)} = R_i, 1 \leq j \leq n$ and $P_k^{(0)} = \emptyset, 1 \leq k \leq m$. The domain and EDBs will not change during the evaluation of the program (i.e. $D^{(i)} = D$ and $R_j^{(i)} = R_j$ for all i, j). On step $i > 0$, a structure Δ_i is created from Δ_{i-1} by adding tuples to each $P_k^{(i-1)}$ in the following way. For each rule of π

$$R_0(x_1^0, \cdots, x_\ell^0) : -R_1(x_1^1, \cdots, x_{k_1}^1), \cdots, R_r(x_1^r, \cdots, x_{k_r}^r),$$

if $R_1^{(i-1)}(y_1^1, \cdots, y_{k_1}^1), \cdots, R_r^{(i-1)}(y_1^r, \cdots, y_{k_r}^r)$ hold in Δ_{i-1}, then we add the tuple $(y_1^0, \cdots, y_\ell^0)$ to $R_0^{(i)}$, where $y_p^0 = y_q^j$ if and only if $x_p^0 = y_q^j$. The step ends when no new tuples can be added this way for any rule. The evaluation ends when $\Delta^{(j)} = \Delta^{(j-1)}$, in which case $\Delta' = \Delta^{(j)}$.

In Datalog programs associated with CSP problems, we have a distinguished 0-arity relation FALSE.

Definition 1.58. A Datalog program π **solves** CSP(Δ) if FALSE is derived for an instance $\mathcal{S}$ if and only if there is no homomorphism from $\mathcal{S}$ to Δ.

Example 1.59. An example of a Datalog program π is the following:

$$\begin{aligned} oddpath(x, y) &: -E(x, y) \\ twopath(x, y) &: -E(x, z), E(z, y) \\ oddpath(x, y) &: -oddpath(x, z), twopath(z, y) \\ FALSE &: -oddpath(x, x) \end{aligned}$$

This program takes as input a graph G with a set of edges E and decides whether an odd cycle exists in the following way: the first rule instructs that every edge from G is an odd path, the second rule instructs that adding two adjacent edges from G forms a 2-path, and the third rule instructs that odd paths are obtained by combining a (previously obtained) odd path with an adjacent 2-path. Finally, FALSE is derived if an odd cycle exists.

Definition 1.60. A datalog program has **width** (l, k) if every IDB is at most l-ary, and if every rule contains at most k different variables. A τ-structure Δ has **width** (ℓ, k) if there exists an (ℓ, k)-Datalog program that solves CSP(Δ). We say Δ has **width** ℓ if it has width (ℓ, k) for some k. We say Δ has **bounded width** if it has width ℓ for some finite ℓ.

Note that every Datalog program has a finite number of variables in its rules, so it has width (l, k) for some l, k. Therefore a structure Δ has bounded width if there is a Datalog program that solves CSP(Δ).

Example 1.61. Consider the program π from Example 1.59:

$$
\begin{aligned}
oddpath(x, y) &: -E(x, y) \\
twopath(x, y) &: -E(x, z), E(z, y) \\
oddpath(x, y) &: -oddpath(x, z), twopath(z, y) \\
FALSE &: -oddpath(x, x)
\end{aligned}
$$

This Datalog program has width (2,3) since every IDB has arity at most 2, and there are at most 3 different variables per rule. It derives FALSE precisely when an odd cycle exists. This program solves CSP(K_2) (where K_2 is a graph with two vertices and an edge connecting them). This is because for every instance S of CSP(K_2), there is an homomorphism from S to K_2 if and only if S is bipartite, which is the case if and only if S does not have an odd cycle in it. Therefore π derives FALSE on S if and only if there is no homomorphism from S to K_2. Since π has width (2,3), we have that K_2 has width (2,3) and so has bounded width.

Chapter 2

Constraint Satisfaction Problems

We are interested in studying the computational complexity of CSPs. One of the central problems in the study of the complexity of CSPs with finite templates is the Dichotomy Conjecture posed by Feder and Vardi in their seminal paper [18] in 1999:

Conjecture 2.1 (Dichotomy Conjecture)**.** *Given a finite relational structure Δ, CSP(Δ) is either in P or NP-complete.*

Note that for any finite structure Δ, CSP(Δ) is in NP, so if $P = NP$ then the conjecture is true. On the other hand, it was shown in [30] that if P$\neq$NP, then there are infinitely many complexity classes between P and NP. The dichotomy conjecture has been proved for special classes of structures, but remains unanswered in the general case. In this thesis, we will focus on a different class of tractable problems: those with bounded width. In this section, we look at algebraic properties of structures that affect the complexity of their CSP, and in particular which of these properties can be used to determine whether a structure has bounded width.

2.1 Positive primitive-definable relations

Let Δ be a τ-structure and $\psi(\bar{x}) = (\exists \bar{y})(\phi_1(\bar{x}, \bar{y}) \wedge ... \wedge \phi_r(\bar{x}, \bar{y}))$ be a pp-formula over τ, where $\bar{x} = (x_1, ..., x_k)$ and $\bar{y} = (y_1, ..., y_\ell)$. Then ψ defines a k-ary relation R_ψ on Δ, where $R_\psi = \{\bar{x} \in D^k | \Delta \models \psi(\bar{x})\}$. We say that R_ψ is **pp-definable from** Δ.

Note that in general adding relations to a structure will make its CSP at least as hard. If $\Delta = (D; \mathcal{R})$ and $\Delta' = (D; \mathcal{R}')$, with $\mathcal{R} \subseteq \mathcal{R}'$, then every instance S of CSP(Δ) can be viewed as an instance of CSP(Δ') by interpreting $R = \emptyset$ for $R \in \mathcal{R}' \setminus \mathcal{R}$, and so CSP($\Delta$) can be reduced to CSP($\Delta'$). However, adding pp-definable relations to a structure does not increase the complexity of its CSP, as the following theorem shows.

Theorem 2.2 (Folklore)**.** *Let ψ be a pp-formula and R_ψ be defined by ψ on $\Delta = \{D; R_1, ..., R_n\}$, and let $\Delta' = \{D; R_1, ..., R_n, R_\psi\}$. Then $CSP(\Delta')$ can be reduced to $CSP(\Delta)$ in polynomial time.*

Proof. Suppose $\psi(\bar{x}) = (\exists \bar{y})(\phi_1(\bar{x}, \bar{y}) \wedge ... \wedge \phi_r(\bar{x}, \bar{y}))$, where $\bar{x} = (x_1, ..., x_k)$ and $\bar{y} = (y_1, ..., y_\ell)$.

Let $\mathcal{S}' = \{S'; R_1, ..., R_n, R_\psi\}$ be an instance of $CSP(\Delta')$. We construct an instance $\mathcal{S} = \{S; R_1, ..., R_n\}$ of $CSP(\Delta)$ in the following way:

the set S is obtained from S' by adding dummy variables $y_1, ..., y_\ell$ for each $\bar{x} \in R_\psi$; that is,

$$S = S' \bigcup_{\bar{x}_i \in R_\psi} \{y_{i,1}, ..., y_{i,\ell}\}.$$

Then, each ${R_i}^{\mathcal{S}}$ is obtained from ${R_i}^{\mathcal{S}'}$ by adding for each $\phi_i(\bar{x}, \bar{y})$ (which is of the form $R_j(z_1, ..., z_{k_j})$ for some j) the tuple $(z_1, ..., z_{k_j})$; that is,

$${R_i}^{\mathcal{S}} = {R_i}^{\mathcal{S}'} \cup \left\{(z_1, ..., z_{k_j}) \;\middle|\; \phi_i(\bar{x}, \bar{y}) = R_j(z_1, ..., z_{k_j}) \text{ for some } i, j\right\}.$$

Then there exists a homomorphism $f : \mathcal{S} \to \Delta$ if and only if there exists a homomorphism $f' : \mathcal{S}' \to \Delta$. To see this, suppose that $f : \mathcal{S} \to \Delta$ is a homomorphism. Let $f' : \mathcal{S}' \to \Delta$

be given by $f' = f|_{\mathcal{S}'}$. Then f' preserves every ${R_i}^{\mathcal{S}'}$, since ${R_i}^{\mathcal{S}'} = {R_i}^{\mathcal{S}} \cap S^{k_i}$. For every $\bar{x} \in R_\psi$, we have that $\psi(f^{(k)}(\bar{x}))$ is satisfied in Δ, since every $\phi_j(f^{(k)}(\bar{x}), f^{(\ell)}(\bar{y}))$ is satisfied. Therefore $\Delta \models \psi(f^{(k)}(\bar{x}))$ and so $f'(\bar{x}) = f(\bar{x}) \in {R_\psi}^{\Delta}$.

We are left with verifying that if there is a homomorphism $f' : \mathcal{S}' \to \Delta'$ then there is also a homomorphism $f : \mathcal{S} \to \Delta$. Create f as an extension of f' in the following way: for every dummy tuple $\bar{y} \in S^l$ defined from a tuple $\bar{x} \in R_\psi$, we have that $f'^{(k)}(\bar{x}) \in R_\psi$ (since f' preserves R_ψ), so there exists $\bar{y}' = (y_1', .., y_\ell')$ that witnesses $\Delta \models \psi(f^{(k)}(\bar{x}))$. Letting $f(y_i) = y_i'$ gives us the desired extension. □

2.2 Polymorphisms and algebras

Let $\langle\Delta\rangle_{pp}$ be the relational structure with same the domain as Δ containing all the relations that are pp-definable from Δ. Given a set A of (finitary) operations of Δ, let $\mathrm{Inv}(A)$ denote the relational structure with the same domain as Δ whose relations are precisely those preserved by all the operations in A. The following theorem shows that pp-definable relations are closely related to polymorphisms.

Theorem 2.3. *[19] Let Δ be a finite relational structure. Then*

$$\langle\Delta\rangle_{pp} = \mathrm{Inv}(\mathrm{Pol}(\Delta)).$$

The clone of polymorphisms of a structure seems to be the appropriate algebraic object to analyze when studying CSPs. The complexity of CSPs is determined by the structure of this clone, and many conclusions can be drawn when certain types of polymorphisms are present in this clone. For example, a **Mal'tsev operation** is a 3-ary operation $f : D^3 \to D$ satisfying

$$f(x, y, y) = f(y, y, x) = x \text{ for all } x, y \in D.$$

Theorem 2.4. *[11] Let Δ be a finite structure. If Pol(Δ) contains a Mal'tsev operation, then CSP(Δ) is tractable.*

Theorem 2.4 shows that the presence of a Mal'tsev polymorphism guarantees a tractable CSP, and provides a way to check that a given structure has a CSP in P.

Example 2.5. Let $\Delta = (\{0,1\}; R)$, where $R = \{(0,1),(1,0)\}$; that is, $(x,y) \in R$ if and only if $x = 1 - y$. Let $f : \Delta^3 \to \Delta$ be defined by $f(x,y,z) = x \oplus y \oplus z$ where $\oplus$ represents XOR addition (i.e. $0 \oplus 0 = 1 \oplus 1 = 0, 0 \oplus 1 = 1 \oplus 0 = 1$). Since $f(x,y,y) = x \oplus y \oplus y = y$ and $f(y,y,x) = y \oplus y \oplus x = x$, we have that f is a Mal'tsev operation. To see that f is a polymorphism, suppose that $(x_1,x_2),(y_1,y_2),(z_1,z_2) \in R$. Then $x_2 = 1-x_1, y_2 = 1-y_1, z_2 = 1 - z_1$, and so

$$f(x_2,y_2,z_2) = (1-x_1) \oplus (1-y_1) \oplus (1-z_1) = 1 - (x_1 \oplus y_1 \oplus z_1) = 1 - f(x_1,y_1,z_1).$$

Therefore $(f(x_1,y_1,z_1), f(x_2,y_2,z_2)) \in R$.

A similar property is known for majority operations. An operation $f : D^3 \to D$ is a **majority operation** if it satisfies

$$f(x,x,y) = f(x,y,x) = f(y,x,x) = x \text{ for all } x,y \in D.$$

Theorem 2.6. *[26] Let Δ be a finite structure. If Pol(Δ) contains a majority operation, then CSP(Δ) is tractable.*

Theorem 2.6 gives us a similar tool to determine that a given structure has a tractable CSP, by looking for a different arity 3 polymorphism.

Example 2.7. Let $\Delta = (\{0,1\}; R)$, where $R = \{(0,1),(1,0)\}$. Let $f : \Delta^3 \to \Delta$ be defined by $f(x_1,x_2,x_3) = \max_{i<j}\{\min\{x_i,x_j\}\}$. Then we see that $f(x,x,y) = \max\{x, \min\{x,y\}\} = x$. Similarly, $f(x,y,x) = f(x,x,y) = x$, therefore f is a majority operation. To see that f

is a polymorphism, suppose that $(x_1, x_2), (y_1, y_2), (z_1, z_2) \in R$. Since Δ only contains two elements, at least two of x_1, x_2, x_3 are equal, and at least two of y_1, y_2, y_3 are equal. Since f is a majority operation, we have that there exists $1 \leq i \leq 3$ such that $f(x_1, x_2, x_3) = x_i$ and $f(y_1, y_2, y_3) = y_i$. Since $(x_i, y_i) \in R$, we have that $(f(x_1, x_2, x_3), f(y_1, y_2, y_3)) \in R$.

Given an algebra $\mathbb{A} = (A; F)$ we say that F is tractable (NP-complete) if $(A; \mathrm{Inv}(F))$ has a tractable (NP-complete) CSP. Similarly, we say that $\mathbb{A}$ is tractable (NP-complete) if F is tractable (NP-complete).

The set $\mathrm{Pol}(\mathrm{Inv}(F))$ consists of all operations that can be obtained from compositions of operations in F and projections. If $f \in \mathrm{Pol}(\mathrm{Inv}(F))$, we say f is a **term operation** of $\mathbb{A}$.

Definition 2.8. An operation $f : D^n \to D$ is **idempotent** if

$$f(x, ..., x) = x \text{ for all } x \in D.$$

Definition 2.9. The **full idempotent reduct** of $\mathbb{A} = (A; F)$ is the algebra $\mathbb{A}_0 = (A; \mathrm{Term}_{id}(\mathbb{A}))$, where $\mathrm{Term}_{id}(\mathbb{A})$ consists of all idempotent term operations of $\mathbb{A}$.

Theorem 2.10. *[12] A finite surjective algebra $\mathbb{A}$ is tractable (or NP-complete) if and only if its full idempotent reduct $\mathbb{A}_0$ is tractable (or NP-complete).*

Theorem 2.10 shows us how it suffices to study the idempotent polymorphisms when analyzing the polymorphism clone of a structure. We are in particular interested in studying weak near-unanimity (WNU) operations.

Definition 2.11. Let $n > 1$. An operation $f : D^n \to D$ is a **weak near-unanimity operation** if it is idempotent and satisfies

$$f(x, ..., x, y) = f(x, ..., x, y, x) = ... = f(y, x, ..., x) \text{ for all } x, y \in D.$$

Note that a majority operation is a particular case of a WNU operation.

Studying the presence of WNU operations in the polymorphism clone of a structure Δ can help analyze the complexity of CSP(Δ), thanks to the following theorems.

Theorem 2.12. *[12] If a τ-structure Δ has no WNU operation, then CSP(Δ) is NP-complete.*

Theorem 2.13. *[5] A core Δ has bounded width if and only if it has WNU polymorphisms of all but finitely many arities.*

The algebraic dichotomy conjecture refines the dichotomy conjecture, tying it to the presence of a WNU operation in the polymorphism clone.

Conjecture 2.14 (Algebraic Dichotomy Conjecture [12])**.** *Let Δ be a finite core. If Δ has a WNU polymorphism, then CSP(Δ) is solvable in polynomial time. Otherwise, CSP(Δ) is NP-complete.*

While Conjecture 2.14 only refers to cores, this is sufficient to establish a dichotomy on every finite structure, since every finite structure Δ has a core Δ' with an equivalent CSP, in the sense that every instance of CSP(Δ) is also an instance of CSP(Δ') with the same answer.

Hence we have a tool for determining the complexity of a CSP by analyzing the presence of WNU operations: if no WNU operation exists, the CSP is NP-complete. If there are WNU operation of all but finitely many arities, then the CSP has bounded width. Proving the Algebraic Dichotomy Conjecture would give a way to study structures in between these two cases, since the existence of a WNU operation would imply that the CSP is in P.

Example 2.15. Let $\Delta = (\{0,1\}; \{(0,1),(1,0)\})$ (hence Δ is isomorphic to K_2). Then Δ does not have any binary WNU polymorphism, since for any binary polymorphism f,

$$(f(0,1), f(1,0)) \in \{(0,1),(1,0)\},$$

and so $f(0,1) \neq f(1,0)$. If f is a WNU operation, then it must satisfy $f(0,1) = f(1,0)$, which is impossible. However, for arity $n > 2$, the set of n-tuples $\{(0,...,0,1),....,(1,0,...,0)\}$ and $\{(1,...,1,0),...,(0,1,...,1)\}$ are disjoint, so there exists an idempotent polymorphism f that satisfies

$$f(x,...,x,y) = ... = f(y,x,...,x) \text{ for all } x,y \in \{0,1\}$$

Since Δ has WNU polymorphisms of all but one arity, we can conclude from Theorem 2.13 that Δ has bounded width.

We can also test for bounded width on structures with the following class of operations:

Definition 2.16. Let $n > 1$. An operation $f : D^n \to D$ is a **near-unanimity operation** if it satisfies

$$f(x,...,x,y) = f(x,...,x,y,x) = ... = f(y,x,...,x) = x \text{ for all } x,y \in D.$$

Note that near-unanimity (NU) polymorphisms are a type of WNU polymorphisms.

Theorem 2.17 (Folklore)**.** *Let $\Delta = (D; R_1, ..., R_n)$ be a finite structure that is a core, and let $f : \Delta^k \to \Delta$ be a NU polymorphism. Then Δ has bounded width.*

Proof. We will show that, given an m-ary NU polymorphism, we can create an $(m+1)$-ary NU polymorphism, thus guaranteeing that Δ has bounded width from Theorem 2.13.

Let $g : \Delta^m \to \Delta$ be a NU polymorphism. Let $g' : \Delta^{m+1} \to \Delta$ be defined by

$$g'(x_1,...,x_m,x_{m+1}) = g(x_1,...,x_m).$$

Then g' is a NU operation, since if at least m of $x_1,...,x_{m+1}$ are equal to x, then at least $m-1$ of $x_1,...,x_m$ are equal to x, and g is a NU operation. To see that g' preserves relations, let R be a relation of Δ with arity r, and let

$$(x_{1,1},...,x_{1,r}),...,(x_{m,1},...,x_{m,r}),(x_{m+1,1},...,x_{m+1,r}) \in R.$$

Then, since $(x_{1,1}, ..., x_{1,r}), ..., (x_{m,1}, ..., x_{m,r}) \in R$, we have that

$$(g(x_{1,1}, ..., x_{m,1}), ..., g(x_{1,r}, ..., x_{m,r})) \in R.$$

Therefore

$$(g'(x_{1,1}, ..., x_{m+1,1}), ..., g'(x_{1,r}, ..., x_{m+1,r})) \in R,$$

and the proof follows. □

2.2.1 Conservative Weak Near-Unanimity polymorphisms

We can refine Theorem 2.13 if we look for WNU polymorphisms that also preserve sets. We define this explicitly:

Definition 2.18. [9] Let Δ be a relational structure. A WNU polymorphism $f : \Delta^k \to \Delta$ is **conservative** if

$$f(x_1, ..., x_k) \in \{x_1, ..., x_k\} \text{ for every } x_1, ..., x_k \in \Delta.$$

By looking for conservative WNU polymorphisms, we can work with any structure (not necessarily a core). This is in part due to the following result:

Lemma 2.19. *Let Δ' be an induced substructure of Δ. If Δ has a conservative WNU polymorphism of arity k, then Δ' has a conservative WNU polymorphism of arity k as well.*

Proof. Let $f : \Delta^k \to \Delta$ be a conservative WNU polymorphism of Δ. Define $f' : \Delta'^k \to \Delta'$ by $f'(x_1, \cdots, x_k) = f(x_1, \cdots, x_k)$ (this is well defined since f is conservative). Then, since f is a conservative WNU polymorphism, f' is idempotent, and satisfies $f'(x, \cdots, x, y) = \cdots = f'(y, x, \cdots, x)$ for all $x, y \in \Delta'$, as well as $f'(x_1, \cdots, x_k) \in \{x_1, \cdots, x_k\}$ for all $x_1, \cdots, x_k \in \Delta'$. Therefore f' is a conservative WNU polymorphism of Δ'. □

From a conservative WNU polymorphism of arity 2 we can build a sequence of conservative WNU polymorphisms of arities $k \geq 2$. Let $f_2 : \Delta^2 \to \Delta$ be a conservative WNU of Δ. Note that, since f_2 is conservative,

$$f_2(x, f_2(x, y)) = f_2(x, y).$$

Let $f_k : \Delta^k \to \Delta$ be defined recursively by

$$f_k(x_1, \cdots, x_k) = f_2(f_{k-1}(x_1, \cdots, x_{k-1}), x_k) \text{ for } k > 2.$$

Note that f_k is idempotent for any $k \geq 2$, and so

$$f_k(x, \cdots, x, y) = f_2(x, y) \text{ for all } x, y \in G.$$

Additionally, every f_k with $k \geq 2$ has the WNU property. We can prove this by induction. We have already seen that f_2 has the WNU property. Assume that f_{k-1} has the WNU property for some $k > 2$. We will prove that

$$f_k(y, x, \cdots, x) = f_k(x, y, x, \cdots, x) = \cdots = f_k(x, \cdots, x, y, x).$$

Since f_{k-1} has the WNU property, we have $f_k(y, x, \cdots, x) = \cdots = f_k(x., \cdots, x, y, x)$ (from the definition of f_k). Furthermore,

$$
\begin{aligned}
f_k(x, \cdots, x, y) &= f_2(x, y), \\
f_k(y, x, \cdots, x) &= f_2(f_{k-1}(y, x, \cdots, x), x) \\
&= f_2(f_{k-1}(x, \cdots, x, y), x) \\
&= f_2(f_2(x, y), x) \\
&= f_2(x, y)
\end{aligned}
$$

Therefore f_k has the WNU property.

We see that Δ has a sequence of (conservative) WNU polymorphisms of arity k for $k \geq 2$. We summarize these results in the following theorem:

Theorem 2.20. *Let Δ be a relational structure with a conservative WNU polymorphism of arity 2. Then Δ has bounded width.*

Proof. From the previous construction, we see that Δ has conservative WNU polymorphisms of arity k for every $k \geq 2$. Therefore, from Lemma 2.19, the core Δ' of Δ has conservative WNU polymorphisms of arity k for every $k \geq 2$. Therefore Δ' has bounded width. Since Δ and Δ' are homomorphically equivalent, we have that Δ has bounded width. □

Corollary 2.21. *Let Δ be a structure and $g : \Delta^n \to \Delta$ a conservative WNU polymorphism of Δ such that*

$$g(x, \cdots, x, y) = g(y, \cdots, y, x) \text{ for all } x, y \in \Delta.$$

Then Δ has bounded width.

Proof. We can define $f_2 : \Delta^2 \to \Delta$ by $f_2(x, y) = g(x, \cdots, x, y)$. Then f_2 is idempotent and conservative (since g is idempotent and conservative), and $f_2(x, y) = g(x, \cdots, x, y) = g(y, \cdots, y, x) = f_2(y, x)$. Therefore, f_2 is a conservative WNU polymorphism of Δ. From Theorem 2.20, Δ has bounded width. □

Theorem 2.20 does not necessarily characterize structures with bounded width, but gives us a tool for finding structures with bounded width, even if the structure is not a core.

A special case of a conservative, binary WNU polymorphism is a conservative semilattice polymorphism, which is also associative. The term semilattice comes from the fact that a semilattice contains an operation with similar properties: it is idempotent, associative and commutative.

Definition 2.22. Let Δ be a relational structure. A binary polymorphism $f : \Delta^2 \to \Delta$ is **associative** if it satisfies

$$f(x, f(y, z)) = f(f(x, y), z) \text{ for all } x, y, z \in \Delta.$$

Definition 2.23. [23] Let Δ be a relational structure. A binary polymorphism $f : \Delta^2 \to \Delta$ is a **semilattice polymorphism** if f is a binary, associative WNU polymorphism.

Since conservative semilattice polymorphisms are binary conservative WNU polymorphisms, any directed graph G that admits a conservative semilattice polymorphism has bounded width, due to Theorem 2.20. However, not every directed graph with bounded width admits a conservative semilattice polymorphism. For example, the graph $\vec{C}_3$ (described in Example 1.17) does not admit a conservative semilattice polymorphism, since any semilattice polymorphism $f : \vec{C}_3^2 \to \vec{C}_3$ must satisfy

$$f(v_1, f(v_2, v_3)) = f(f(v_1, v_2), v_3) = f(v_3, f(v_1, v_2))$$

and

$$f(v_3, f(v_1, v_2)) \to f(v_1, f(v_2, v_3)),$$

but this cannot happen since $\vec{C}_3$ has no loops. However, $\vec{C}_3$ has bounded width (see Corollary 4.9).

Hell and Rafiey gave a characterization of directed graphs that admit a conservative

semilattice polymorphism in [23]:

Definition 2.24. [23] Let C be a circle with two distinguished points, N and S, and let H be a directed graph. Let $\{I_v\}, v \in H$ and $\{J_v\}, v \in H$ be two families of arcs of C such that each I_v contains N but not S, and each J_v contains S but not N. We say the families $\{I_v\}$ and $\{J_v\}$ are **consistent** if they have the same clockwise order of their clockwise ends; i.e., the clockwise end of I_v precedes in the clockwise order the clockwise end of I_w if and only if the clockwise end of J_v precedes in the clockwise order the clockwise end of J_w.

Definition 2.25. [23] Let H be a directed graph. A **bi-arc representation** of H is a consistent pair of families of circular arcs $\{I_v\}, \{J_v\}, v \in H$ (as described in Definition 2.24) such that $u \to v$ if and only if I_u and J_v are disjoint. A directed graph H is called a **bi-arc directed graph** if it has a bi-arc representation.

Theorem 2.26. *[23] The class of bi-arc directed graphs coincides with the class of directed graphs that admit a conservative semilattice polymorphism.*

We can redefine semilattice polymorphisms by replacing the associative property with a transitive property.

Definition 2.27. Let Δ be a relational structure. A binary polymorphism $f : \Delta^2 \to \Delta$ is a **transitive polymorphism** if $f(x, y) = x$ and $f(y, z) = y$ implies $f(x, z) = x$ for all $x, y, z \in \Delta$.

A conservative, transitive WNU polymorphism $f : \Delta \to \Delta$ defines an order on Δ by making $f(x, y) = \min\{x, y\}$. This was noted independently in [32]. This is well-defined since f is conservative, transitive, idempotent, and has the WNU property.

Theorem 2.28. *Let Δ be a relational structure. A binary polymorphism $f : \Delta^2 \to \Delta$ is a conservative semilattice polymorphism if and only if f is a conservative, transitive WNU polymorphism.*

Proof. First we prove that a semilattice polymorphism is transitive. Suppose that f is a semilattice polymorphism, and suppose that $f(x,y) = x, f(y,z) = y$ for some $x, y, z \in \Delta$. Then, since f is associative,

$$f(x,z) = f(f(x,y),z) = f(x,f(y,z)) = f(x,y) = x.$$

Therefore f is transitive. Now suppose that f is a conservative, transitive WNU polymorphism. To show that f is a semilattice polymorphism, we must show that f is associative. Since f defined an order on Δ by making $f(x,y) = \min\{x,y\}$, we have that

$$f(f(x,y),z) = \min\{x,y,z\}$$

and

$$f(x,f(y,z)) = \min\{x,y,z\}.$$

Therefore $f(x,(f(y,z))) = f(f(x,y),z)$ for all $x, y, z \in \Delta$. □

2.3 Bounded width

Structures with bounded width offer a computational advantage over other structures with tractable CSPs. While there exist efficient algorithms for both types of structure that decide whether a homomorphism from a given instance exists, in the case of a structure with bounded width this algorithm is explicitly known. If there exists a Datalog program that solves CSP(Δ), then Δ has bounded width, and CSP(Δ) is in particular tractable. However, it is not necessary to consider all Datalog programs. Instead, it is sufficient to consider the Canonical Datalog program, defined in [18].

The Canonical (ℓ, k)-Datalog program for Δ infers all constraints on ℓ variables that can be obtained from k variables. If the program can infer the empty relation FALSE from an instance S, then the instance has no solution. It is called canonical since it derives

every constraint that a Datalog program with the same parameters could derive, and solves CSP(Δ) if any Datalog program with the same parameters does. This is shown in the following theorem:

Theorem 2.29. *[18] If there exists an (ℓ, k)-Datalog program that solves CSP(Δ), then the Canonical (ℓ, k)-Datalog program for Δ solves it as well.*

We are therefore interested in identifying which structures with tractable CSPs also have bounded width and in finding classes of structures where a stronger CSP dichotomy exists. We state this in the following question:

Question 2.30. *For which classes $\mathcal{C}$ is the following dichotomy true: for every $\Delta \in \mathcal{C}$, either Δ has bounded width or CSP(Δ) is NP-complete?*

We will refer to a dichotomy described in Question 2.30 as a **bounded width dichotomy**.

Example 2.31. Hell and Nešetřil proved in [22] that simple graphs have an NP-complete CSP if and only if they are non-bipartite. Otherwise, they have a tractable CSP. We have seen in Example 1.61 that bipartite graphs have bounded width, since there is a Datalog program that solves their CSP. Therefore, simple graphs have a bounded width dichotomy. We will study classes of directed graphs that have a bounded width dichotomy in Chapter 4.

While the definition of bounded width relies on the existence of an appropriate Datalog program, we have shown ways to characterize or test for bounded width (as well as other complexity classes) through the presence of certain polymorphisms. We summarize these results in Figure 2.1. The square boxes represent complexity classes, while the rounded boxes represents the presence of a given type of operation in Pol(Δ). Arrows represent containment (e.g. if Δ admits a majority polymorphism, then CSP(Δ) is in P), while the red arrow represents part of the Algebraic Dichotomy Conjecture that remains open.

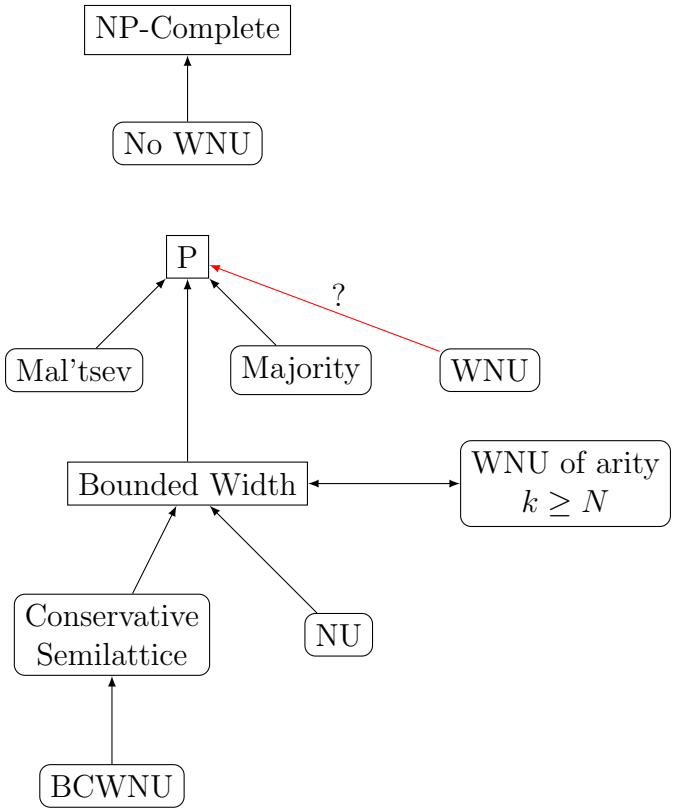


Figure 2.1: Relation between complexity classes and polymorphisms present in the structure.

Note that Theorem 3.2 does not hold for every structure Δ with a tractable CSP. We give an example:

Example 2.32. Consider the structure **Linear Equations** with domain $\mathbb{F}_2$, the two-element field, and relations

$$\delta^b_{\bar{a},n} = \{(x_1, ..., x_n) \in \{0,1\}^n | a_1x_1 + ... + a_nx_n = b\}$$

for every $n \in \mathbb{N}, b, a_1, ..., a_n \in \{0,1\}$, where $\bar{a} = (a_1, ..., a_n)$. An instance of this CSP is simply a system of linear equations over $\mathbb{F}_2$, and the problem is equivalent to deciding whether the system is consistent, which can be done in polynomial time in a number of different ways (e.g. through Gaussian elimination).

However, for any $k \in \mathbb{N}$, choose $n > k$ and look at the instance given by the system

$$\begin{cases} x_1 + ... + x_n = 0 \\ x_1 + ... + x_n = 1 \end{cases}.$$

This system is clearly inconsistent, so there is no homomorphism to our structure Δ. However, every assignment of k variables is trivially a partial homomorphism, since there are no (non-empty) relations of arity at most k in our instance.

The structure **Linear Equations** has a tractable CSP, but it does not have bounded width. Even though **Linear Equations** is not a finite structure since it has infinitely many relations, there are finite structures that share this property of having a tractable CSP but not bounded width. We give an example of this in 2.35.

One way to determine that a structure does not have bounded width is by showing it has the ability to count.

Definition 2.33. [18] A τ-structure Δ has the **ability to count** if the following conditions hold:

1. There exist elements $d_0, d_1 \in \Delta$ as well as a 3-ary relation C that contains the triples $(d_0, d_0, d_1), (d_0, d_1, d_0), (d_1, d_0, d_0)$ and a unary relation Z that contains d_0.

2. If an instance S of CSP(Δ) contains only the relations C and Z, and all tuples from these two relations can be partitioned into sets A and B such that A contains exactly one more tuple of the relation C than B, and each element of S appears in exactly one tuple from each set (or in none at all), then there is no homomorphism from S to Δ.

Theorem 2.34. *[18] If a τ-structure Δ has the ability to count, then it does not have bounded width.*

Example 2.35. Consider the structure **Boolean Algebra**, with domain $\{0, 1\}$ and relations representing the conjunction, disjunction and negation operations. We can think of these relations as

$$\wedge = \{(x, y, z) \in \{0, 1\}^3 | x \wedge y = z\},$$

$$\vee = \{(x, y, z) \in \{0, 1\}^3 | x \vee y = z\},$$

$$\neg = \{(x, y) \in \{0, 1\}^2 | x = \neg y\}.$$

Then

$$f(x, y, z) = (\neg y \wedge x) \vee (\neg y \wedge z) \vee (x \wedge y \wedge z)$$

is a polymorphism that satisfies the Mal'tsev condition $f(x, y, y) = f(y, x, x) = y$. This is because

$$f(x, y, y) = (\neg y \wedge x) \vee (x \wedge y) = x$$

and

$$f(y, y, x) = (\neg y \wedge x) \vee (y \wedge x) = x.$$

This implies that the CSP problem for **Boolean Algebra** is in P. We can define (in a positive-primitive way) the following relations in this structure:

$$Z = \{x \in \{0, 1\} \mid \exists y : x \wedge y = x, x = \neg y\},$$

$$C = \{(x, y, z) \in \{0, 1\}^3 \mid \exists u, v : x \vee y = u, u = \neg z, x \wedge y = v, z \wedge y = v, Z(v)\}.$$

Note that this yields $Z = \{0\}$ and $C = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. We can think of these relations as $x = 0$ when $x \in Z$, and $x + y + z = 1$ when $(x, y, z) \in C$. Therefore, when given an instance S of CSP(**Boolean Algebra**) that satisfies condition 2 of Definition 2.33, then by subtracting the tuples (interpreted as equations) in set B from the ones in set A, we are left with $0 = 1$, which cannot be satisfied. Therefore there is no homomorphism from such instance S to **Boolean Algebra**. We conclude that **Boolean Algebra** has the ability to count.

While there exist examples of structures with a tractable CSP and without bounded width, there are many classes of structures where this does not occur. We will study some of these classes in Chapter 4.

Chapter 3

Pebble Games

In order to further analyze CSPs, certain homomorphism-building games can be studied. The idea behind these games arises naturally when trying to construct a homomorphism between two given structures. They are particularly interesting when no homomorphism exists but some partial homomorphisms may be constructed.

3.1 k-Pebble games

In a k-pebble game, we have two players, Spoiler and Duplicator, who play on τ-structures S and Δ. Spoiler has k striped pebbles, labelled 1 through k. Similarly, Duplicator has k dotted pebbles, labelled 1 through k.

The game plays in rounds. In the first round, Spoiler starts by placing the striped pebbles $1, ..., k$ on elements $s_1^1, ..., s_k^1$ of S. Duplicator responds by placing the dotted pebbles $1, ..., k$ on elements $d_1^1, ..., d_k^1$ of Δ, which represent the images of the respective elements chosen by Spoiler under a partial homomorphism. In round j, Spoiler removes a striped pebble i from s_i^{j-1} and places it on a new element s_i^j of S. Duplicator removes dotted pebble i from d_i^{j-1} and places it on a new element d_i^j of Δ. If at any time the assignment $s_i^j \mapsto d_i^j, i = 1, ..., k$ is not a partial homomorphism from S to Δ, Spoiler wins the game. If Duplicator can ensure that Spoiler never wins, Duplicator wins the game.

We can assume that in the first round, Spoiler places the striped pebbles one at a time, with Duplicator responding by placing the matching dotted pebble each time. This will not affect winning strategies; only the number of rounds played might change.

We will study these games by fixing Δ and then analyzing the game for different structures S and values of k, in a similar way to how we analyze CSPs.

Note that if there is a homomorphism from S to Δ, Duplicator has a winning strategy, by always playing according to that homomorphism. We are therefore interested in situations where there is no homomorphism between S and Δ, and we are interested in finding which player has a winning strategy.

3.2 The case K_n

We begin with the case $\Delta = K_2$. We know that there is a homomorphism between S and K_2 if and only if S is bipartite, so Duplicator will have a winning strategy as long as S is bipartite. Additionally, if $k \leq 2$, Duplicator can also win by always placing different dotted pebbles on different vertices of K_2.

We will start by analyzing the case $\Delta = K_2$ with S being an odd cycle and $k = 3$. If $S = C_3$ then spoiler can win in one round by placing a striped pebble on each vertex, since there is no homomorphism from C_3 to K_2. Let $S = C_5$, and label the vertices of C_5 as a, b, c, d, e in clockwise order. Label the vertices of K_2 as x, y. Spoiler can start by playing on a and then, regardless of Duplicator's play (they are symmetric, so assume Duplicator plays on x), by playing on c. If Duplicator plays on y, Spoiler can place the third striped pebble on vertex b and win. If instead Duplicator places the second dotted pebble on x (same as the first dotted pebble), Spoiler can then move on e, and so Duplicator must play on y, since a and e are adjacent (see Figure 3.1). Then the round is over and Spoiler can remove the first pebble (Duplicator must do the same) and place it on d. Duplicator loses the game since the dotted pebble 2 is on x, the dotted pebble 3 is on y, and d is adjacent to

both c and e (see Figure 3.2).

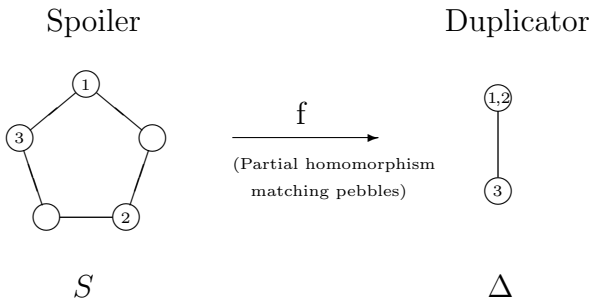


Figure 3.1: Game state after the first round of play.

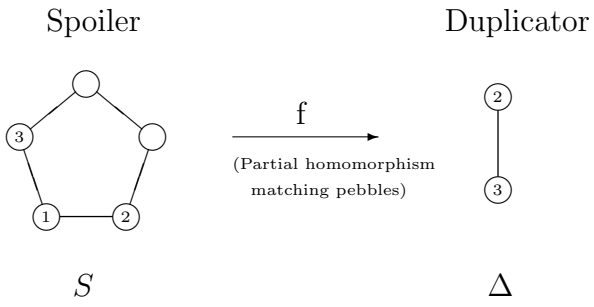


Figure 3.2: End of the game: Duplicator has no moves.

More generally, let $S = C_n$, with n odd. Label the vertices of C_n as $a_1, a_2, ..., a_n$ in clockwise order. Spoiler can start by playing on a_1 and then on $a_{\frac{n+1}{2}}$. Suppose again that Duplicator starts by playing on x. This breaks C_n down into two paths (going clockwise from a_1 to $a_{\frac{n+1}{2}}$ and from $a_{\frac{n+1}{2}}$ to a_1) of different parity, and so regardless of Duplicator's play, one of the two paths will not map homomorphically to K_2 with the two assignments chosen. Spoiler can then play in the midpoint of a path with no homomorphism. This again breaks the path down into two smaller paths, one of which will not map homomorphically to K_2 with the assignments chosen (depending on where Duplicator plays). This way, Spoiler can leave the two striped pebbles on the endpoints of the smaller path that does not map to K_2, and take the remaining striped pebble and place it on the midpoint of this path. As

before, this creates two smaller paths, one of which will not map homomorphically to K_2. By repeating this strategy, Spoiler can half the length of the path each round until the path has length 3, and Duplicator has no available play and loses the game. By playing with this "binary search" strategy, Spoiler can win in $\lfloor \log_2(n) \rfloor$ rounds. We have included a program in Appendix A that computes the number of rounds needed for Spoiler to win the 3-pebble game on C_n and K_2 for $n < 14$.

Note that if Spoiler has a winning strategy with k pebbles, then Spoiler will also have a winning strategy with $k+1$ pebbles. Indeed, if Spoiler has $k \geq 3$ pebbles in the previous example, Spoiler can perform $k-2$ steps of the binary search each round, and so can win in $\lfloor \frac{\log_2(n)}{k-2} \rfloor = \lfloor \log_{2^{k-2}}(n) \rfloor$ rounds.

In general, if S is non-bipartite, it will have an odd cycle in it. In the 3-pebble game, Spoiler can then play exclusively on the smallest odd cycle in S and win in the way described above.

We have shown that if $\Delta = K_2$ and $k \geq 3$, then Duplicator wins the k-pebble game between S and Δ if and only if there is a homomorphism from S to Δ. This property does not hold for K_n with $n \geq 3$. If $k > n$, there exists S such that there is no homomorphism from S to Δ, but Duplicator has a winning strategy. Note that if S is n-colourable, there is a homomorphism from S to K_n.

Let $k = n + 1$. A suitable candidate for S is the Mycielski M_k graph of order k (see 1.2.1). It has chromatic number k (and so there is no homomorphism from M_k to K_{k-1}), and it is triangle-free. Figure 3.3 shows the Mycielski graph of order 4. Does Duplicator have a winning strategy for the $k-$pebble game from M_k to K_{k-1}?

For a given stage of play, let P be the set of vertices of M_k with pebbles on them, and $f : P \to K_{k-1}$ be the partial homomorphism associated with Duplicator's play. Assume that Spoiler plays on a vertex $v \in M_k \setminus P$, and for all $w \in K_{k-1}$ let $f_w : P \cup \{v\} \to K_{k-1}$ be the extension of f that maps v to $w \in K_{k-1}$.

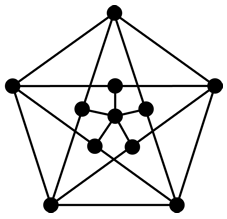


Figure 3.3: The Mycielski graph of order 4, M_4

Let $\psi : K_{k-1} \to \mathbb{N}$ be defined by

$$\psi(w) = \max_{u \in M_k} \{|f_w\left(N(u) \cap (P \cup \{v\})\right)|\}.$$

The winning strategy for Duplicator is to choose $w \in K_{k-1}$ that minimizes $\psi(w)$. In other words, Duplicator is trying to minimize the images of sets of pebble vertices of S that belong to the same neighbourhood. If we think of the partial homomorphism as a (partial) n-colouring of S, Duplicator is trying to minimize the number of colours in each neighbourhood of a vertex of S.

This way, for Spoiler to win, the last striped pebble must be placed on a vertex v that is adjacent to $k-1$ pebbles, each having a different colour. Duplicator would only choose the last available colour for the last dotted pebble if any other choice would have not formed a valid colouring. But $N(v)$ is an independent set for each $v \in M_k$, so Spoiler would need an additional $n-2$ striped pebbles to force the last colour to be used in $N(v)$. This requires too many pebbles, since $(k-1)+(k-2) > k$ when $k > 3$, so Spoiler does not have a winning strategy, and therefore Duplicator does.

3.3 (ℓ, k)-Pebble games

We can generalize k-pebble games by adding a variable $\ell < k$ that tells us how many striped pebbles Spoiler must pick up every round.

The (ℓ, k)-pebble game is played in rounds as well. In the first round, Spoiler places k striped pebbles on elements $s_1^1, ..., s_k^1$ of S and Duplicator responds by placing dotted pebbles $1, ..., k$ on elements $d_1^1, ..., d_k^1$ of Δ. In subsequent rounds, Spoiler picks up $k - \ell$ striped pebbles from S and places them on new elements, and Duplicator must respond by picking up the corresponding dotted pebbles and placing them on elements of Δ. If at any point the associated mapping is not a partial homomorphism, Spoiler wins. If Duplicator can ensure that Spoiler never wins, then Duplicator wins. We can see that the k-pebble game corresponds to the $(k-1, k)$-pebble game under this definition.

We can think of a winning strategy for Duplicator as a family of partial homomorphisms that corresponds to the partial assignments given by the choice of pebbles for each player, with the property that Duplicator must always have a valid play. This is achieved in the following way:

Definition 3.1. A **winning strategy** for Duplicator for the (ℓ, k)-pebble game between S and Δ is a non-empty set $\mathcal{H}$ of partial homomorphisms from S to Δ such that

- $\mathcal{H}$ is closed under restrictions of its members
- for all functions $h \in \Delta$ with $|\text{dom}(h)| = d \leq \ell$ and for all $a_1, ..., a_{k-d} \in S$ there is an extension $h' \in H$ of h that is defined on $a_1, ..., a_{k-d}$.

3.4 Pebble games and bounded width

We saw that for $\Delta = K_2$, Duplicator always wins if $k \leq 2$, while if $k \geq 3$ Duplicator wins if and only if S is bipartite; that is, if S maps to K_2. However, for $n > 2$, when $\Delta = K_n$, Duplicator will have a winning strategy for some S that do not map to Δ. What is the difference between these cases? This question is answered by the following theorem:

Theorem 3.2. *[28] Let Δ be a τ-structure. There exists a $(k-1, k)$-Datalog program that solves CSP(Δ) if and only if the following condition holds:*

for every τ-structure S, if Duplicator wins the k-pebble game on S and Δ, then there is a homomorphism from S to Δ.

Theorem 3.2 shows a connection between k-pebble games and structures that have a CSP that is solved by a $(k-1,k)$-Datalog program. There is also a connections between (ℓ,k)-pebble games and Canonical (ℓ,k)-Datalog programs, shown in the following theorem:

Theorem 3.3. *[18] The Canonical (ℓ,k)-Datalog program for Δ derives FALSE on S if and only if Spoiler has a winning strategy for the associated (ℓ,k)-pebble game.*

We can think of winning strategies for Spoiler as a decision tree, and compare it to the derivation tree of FALSE when running the Canonical Program on S. In this way it can be shown that a winning decision tree exists for Spoiler precisely when FALSE can be derived.

Example 3.4. Suppose that $S = C_5 = (\{a,b,c,d,e\};\{(a,b),...,(e,a)\})$ and $\Delta = K_2 = (\{u,v\};\{(u,v)\})$. We will give a $(2,3)$-Datalog program that solves CSP(K_2) and use it to derive a winning strategy for Spoiler in the associated game.

Consider the following (2,3)-Datalog Program π (discussed in example 1.59):

$$oddpath(x,y) : -E(x,y) \tag{3.1}$$

$$twopath(x,y) : -E(x,z), E(z,y) \tag{3.2}$$

$$oddpath(x,y) : -oddpath(x,z), twopath(z,y) \tag{3.3}$$

$$FALSE : -oddpath(x,x) \tag{3.4}$$

When running π with C_5 as an instance, we obtain the following relations in each rule:

3.1 $oddpath = E(C_5)$

3.2 $twopath = \{(a,c),(b,d),...,(e,b)\}$

3.3 $oddpath = \{a,b,c,d,e\}^2$

3.4 $FALSE$.

When running π with K_2 as an instance, we obtain instead:

3.1 $oddpath = \{(u, v)\}$

3.2 $twopath = \{(u, u), (v, v)\}$

3.3 $oddpath = \{(u, v)\}$.

In this case, $FALSE$ is not derived on rule 3.4.

We can write a derivation tree of $FALSE$ from running π on C_5 by backtracking, say, obtaining $FALSE$ from $oddpath(a, a)$

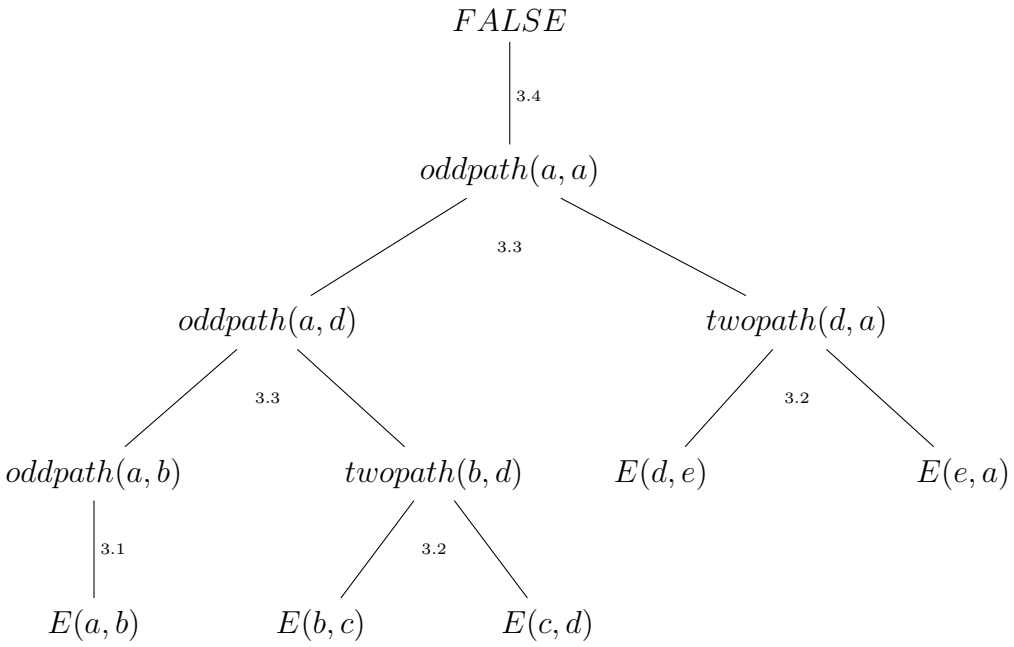


Figure 3.4: Derivation tree of FALSE on $\pi(C_5)$

We can use this derivation tree to construct a winning strategy for Spoiler. We will traverse the tree from its root to one of its leaves. When moving from a node to its children, Spoiler will place striped pebbles on the vertices of C_5 associated with the body of that rule.

In the first turn, we move from FALSE to the tuple $oddpath(a, a)$. In this case, Spoiler needs to play on the vertex a (it is irrelevant what Spoiler does with the rest of the striped

pebbles). Then Duplicator must choose an image of a in K_2, say u. Whatever Duplicator chooses, it will create a tuple that is not in the associated relation in the derivation of $\pi(K_2)$. In this case, the tuple (u, u) is not in *oddpath*. The next turn begins and Spoiler moves down from $oddpath(a, a)$ to its children. Since $oddpath(a, a)$ is obtained from the rule

$$oddpath(a, a) : -oddpath(a, d), twopath(d, a),$$

Spoiler will keep the striped pebble on a and place a new striped pebble on d (again, the third pebble is irrelevant this turn, since the rule only used two vertices). Then Duplicator must either choose the assignment $d \mapsto u$ or $d \mapsto v$. Each one creates a tuple that is not in one of the associated relations in the derivation of $\pi(K_2)$. Suppose that Duplicator chooses $d \mapsto v$. Then we have that $(a, d) \in twopath$, but $(u, v) \notin twopath$ in the associated relation. Therefore Spoiler chooses to play on the node $twopath(d, a)$. By going down to its children, we have that $twopath(d, a)$ is obtained from the rule

$$twopath(d, a) : -E(d, e), E(e, a).$$

Therefore Spoiler will keep the striped pebbles on d, a and place a striped pebble on e. Now each possible assignment of e creates a tuple not in E, which is an original relation of K_2 and thus will make the assignment invalid. At this point Spoiler wins.

Theorems 3.2 and 3.3 give us another set of tools to determine if a structure has bounded width, other than finding appropriate Datalog programs or looking for certain polymorphisms. We can also analyze the associated pebble games to determine if a structure has bounded width. In Section 4.5 we give a proof that transitive directed graphs have bounded width by analyzing the associated $(1, 2)$-pebble game.

Chapter 4

Directed Graphs

4.1 Reduction to directed graphs

Feder and Vardi [18] showed that every CSP is polynomial-time equivalent to the CSP of a directed graph. Therefore the dichotomy conjecture can be reduced to the case of directed graphs; that is, it is sufficient to show there is a dichotomy for directed graphs to prove the general case.

Bulín, Delic, Jackson and Niven [13] showed that the CSP of every structure Δ is logspace equivalent to the CSP of a directed graph $\mathcal{D}(\Delta)$, and gave an explicit construction for $\mathcal{D}(\Delta)$. Moreover, this graph preserves many of the polymorphism properties of the original structure.

We introduce some definitions that will be used in the theorem:

Definition 4.1. [13] An **operational signature** is a set of operation symbols with associated arities.

Definition 4.2. [13] An **identity** is an expression of the form $u \approx v$ where u, v are terms in some operational signature. An identity $u \approx v$ is **linear** if u and v involve at most one occurrence of an operational symbol. An identity $u \approx v$ is **balanced** if the sets of variables occurring in u and v are the same.

Example 4.3. [13] The identities $f(x,y) \approx g(x), h(x,y,z) \approx x$ are linear. The identity $f(x,x,y) \approx g(y,x,x)$ is balanced.

Definition 4.4. [13] A set of identities Σ is **linear** if it contains only linear identities; **balanced** if all the identities in Σ are balanced; and **idempotent** if for each operation symbol f appearing in an identity of Σ, the identity $f(x,...,x) \approx x$ is in Σ.

Definition 4.5. [13] Let Σ be a set of identities in a signature with operation symbols $\mathcal{F} = \{f_\lambda | \lambda \in \Lambda\}$. We say that a relational structure $\mathbb{A}$ satisfies Σ (and write $\mathbb{A} \models \Sigma$) if for every $\lambda \in \Lambda$ there is a polymorphism $f_\lambda^{\mathbb{A}} \in Pol(\mathbb{A})$ such that the identities in Σ hold universally in $\mathbb{A}$ when for each $\lambda \in \Lambda$ the symbol f_λ is interpreted as $f_\lambda^{\mathbb{A}}$.

We can now state the theorem from [13].

Theorem 4.6. *[13] For every relational structure Δ there exists a directed graph $\mathcal{D}(\Delta)$ such that the following hold:*

1. *CSP(Δ) and CSP($\mathcal{D}(\Delta)$) are logspace equivalent.*
2. *Δ is a core if and only if $\mathcal{D}(\Delta)$ is a core.*
3. *If Σ is a linear idempotent set of identities such that the algebra of polymorphisms of Z satisfies Σ and each identity in Σ is either balanced or contains at most two variables, then*

$$\Delta \models \Sigma \text{ if and only if } \mathcal{D}(\Delta) \models \Sigma$$

where Z is the directed path $a \to b \leftarrow c \to d$.

In particular, if Δ is a core and has bounded width, then $\mathcal{D}(\Delta)$ has bounded width.

Theorem 4.6 shows that it is sufficient to analyze the CSP of directed graphs when analyzing the CSP of finite structures, since for every finite structure there is a directed graph with an equivalent CSP. Moreover, understanding which classes of directed graphs

have a bounded width dichotomy will help us understand which classes of structures also have a bounded width dichotomy. We are thus interested in studying the CSP of finite directed graphs and understanding which directed graphs have bounded width.

4.2 Directed cycles

We start by analyzing a simple class of directed graphs to give an illustrative example and a simple result that we will use in Section 4.4.

Example 4.7. The directed cycle $\vec{C}_2$ has vertices v_1, v_2 and edges $v_1 \to v_2, v_2 \to v_1$. We illustrate this by drawing an undirected edge between the vertices. Note that $\vec{C}_2$ is not simple, while $\vec{C}_n$ is simple for $n > 2$.

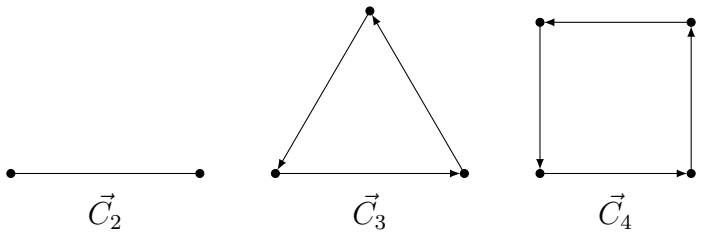


Figure 4.1: The directed cycles $\vec{C}_2, \vec{C}_3, \vec{C}_4$.

Note that there exists a homomorphism from $\vec{C}_m$ to $\vec{C}_n$ if and only if m is a multiple of n. We can use this to give the following useful result. This was independently found in [32].

Lemma 4.8. *A directed cycle $\vec{C}_n$ has a WNU polymorphism of arity k for every $k \geq 3$.*

Proof. Let $k \geq 3$ and consider $\vec{C}_n^{\,k}$, which has elements of the form $(v_1, \cdots, v_k)$ for $v_1, \cdots v_k \in \vec{C}_n$ and edges $(u_1, \cdots, u_k) \to (v_1, \cdots, v_k)$ whenever $u_i \to v_i$ for every $1 \leq i \leq k$. Every element $\bar{v} \in \vec{C}_n^{\,k}$ has exactly one incoming edge and one outgoing edge, where $\bar{v} = (v_1, ..., v_k)$ has an edge to $\bar{u} = (u_1, ..., u_k)$ if $v_i \to u_i$ for every i. Notice that $\bar{v}$ belongs to a cycle, since following the outgoing edges from $\bar{v}$ will get us back to $\bar{v}$ after n steps. Therefore $\vec{C}_n^{\,k}$ is the disjoint union of directed cycles with n elements each.

We are only interested in cycles of $\vec{C_n}^k$ that contain elements of the form $(y, x, ..., x)$, $(x, y, x, ..., x), ...,$ or $(x, ..., x, y)$, since for every other connected component a projection onto any coordinate will provide a WNU polymorphism for those elements. Suppose (without loss of generality) that a cycle contains the element $(x, ..., x, y)$ for some $x, y \in \vec{C_n}$. Then every element of that cycle is of the form $(x', ..., x', y')$ for some $x', y' \in \vec{C_n}$. Since $k \geq 3$ and x is repeated at least twice in $(x, ..., x, y)$, no other element from that cycle can have as its components a permutation of the components of $(x, ..., x, y)$. In other words, no other element in that cycle is of the form $(y, x, ..., x)$, $(x, y, x, ..., x), ...,$ or $(x, ..., x, y)$. We can then define a WNU polymorphism on these cycles by mapping $(y, x, ..., x)$, $(x, y, x, ..., x), ...,$ and $(x, ..., x, y)$ to the same element (say x). □

An example of a WNU of arity k of $\vec{C_n}$ is $f : \vec{C_n}^k \to \vec{C_n}$ where

$$f(\bar{x}) = f(x_1, ..., x_k) = \begin{cases} x & \text{if for some } j, x_i = x \text{ for every } i \neq j \\ x_1 & \text{otherwise.} \end{cases}$$

Every directed cycle is a core, so we can conclude that directed cycles have bounded width.

Corollary 4.9. *The directed cycle $\vec{C_n}$ has bounded width for every n.*

4.3 Tournaments

A special case of directed graphs is the class of tournaments. Recall from 1.18 that a tournament T is a complete directed graph; that is, for every $v_1, v_2 \in T$, either $(v_1, v_2) \in E$, $(v_2, v_1) \in E$, or $v_1 = v_2$. Every endomorphism of a tournament must be injective, so all tournaments are cores, and therefore we can study the existence of WNU polymorphisms to determine whether they have bounded width.

For the following theorem we need to introduce the concept of semi-complete graphs.

Definition 4.10. A directed graph $G = (V; E)$ is **semi-complete** if, for every pair of distinct vertices $v_1, v_2 \in V$, at least one of $v_1 \to v_2, v_2 \to v_1$ holds. Note that semi-complete graphs are not always simple, since we can have $u \to v$ and $v \to u$ for vertices u, v.

Semi-complete graphs are a natural extension of tournaments, and hold similar properties. It was shown in [3] that semi-complete graphs (and in particular tournaments) possess the CSP dichotomy: the CSP of a semi-complete graph is either NP-complete or in P.

Theorem 4.11. *[3] Let H be a semi-complete directed graph.*

- *If H contains at most one directed cycle, then CSP(H) is in P.*
- *Otherwise CSP(H) is NP-complete.*

We will show that tournaments have a bounded width dichotomy: the CSP of a tournament is either NP-complete or has bounded width. Although many of the results we give were already found in [3], we will provide our own proofs which employ different methods.

Up to isomorphism there is exactly one tournament with one vertex and one with two vertices, namely a single vertex and a single edge. These two have WNU polymorphisms of any arity. In fact, any tournament obtained from a finite linear order has a WNU polymorphism f of every arity, that is, the family of directed graphs $\vec{L_n} = (\{v_1, \cdots, v_n\}; \{(v_i, v_j) | i < j\})$ all have WNU polymorphisms of every arity. We show this in the following lemma:

Lemma 4.12. *The directed graph $\vec{L_n}$ (see Example 1.17) has a WNU polymorphism of arity k for $k \geq 2$.*

Proof. Let $f : \vec{L_n}^k \to \vec{L_n}$ be given by

$$f(x_1, ..., x_k) = \min\{x_1, ..., x_k\}.$$

Then f is a WNU polymorphism. To see this, note that f is idempotent and satisfies $f(y, x, ..., x) = ... = f(x, ..., x, y)$. Suppose $x_1 \to y_1, ..., x_k \to y_k$. Then $x_1 < y_1, ..., x_k <$

y_k, so $\min\{x_1, ..., x_k\} < \min\{y_1, ..., y_k\}$. Therefore f also preserves edges and is a WNU polymorphism. □

There are two non-isomorphic tournaments with 3 vertices: a 3-cycle and a linear order on 3 elements. It is easy to verify that these tournaments have WNU polymorphisms of every arity.

We provide a lemma that will help us determine the complexity of tournaments obtained from simple constructions.

Lemma 4.13. *If a tournament $T = (V; E)$ has a WNU polymorphism f of arity n, then the linear sums $T \oplus \{u\}$, $\{u\} \oplus T$ also have a WNU polymorphism of arity n.*

Proof. We will describe the WNU polymorphism for $T \oplus \{u\}$, but the same function is also a WNU polymorphism for $\{u\} \oplus T$.

Let $T \oplus \{u\} = T' = (V'; E')$ and let $f' : T'^n \to T'$ be given by

$$f'(x_1, ...x_n) = \begin{cases} f(x_1, ..., x_n) & \text{if } x_1, ..., x_n \in V \\ u & \text{if } u \in \{x_1, ..., x_n\} \end{cases}.$$

Since f is a WNU polymorphism, we see that f' is idempotent and satisfies $f'(y, x, ..., x) = ... = f'(x, ..., x, y)$. Now suppose $x_1 \to y_1, ..., x_n \to y_n$. Clearly $u \notin \{x_1, ..., x_n\}$. If $u \in \{y_1, ..., y_n\}$, then $f'(x_1, .., x_n) = f(x_1, .., x_n) \to f'(y_1, ..., y_n) = u$. If $u \notin \{y_1, ..., y_n\}$, then $f'(x_1, .., x_n) = f(x_1, .., x_n) \to f(y_1, ..., y_n) = f'(y_1, ..., y_n)$. So f' is a WNU polymorphism of arity n. □

There are four non-isomorphic tournaments with 4 vertices, illustrated in Figure 4.2.

The first three consist of a linear order, a linear sum of a vertex and a 3-cycle, and a linear sum of a 3-cycle and a vertex. From Lemmas 4.12 and 4.13, we see that they have WNU polymorphisms of every arity.

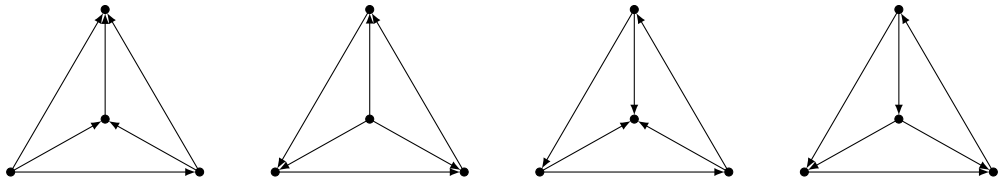


Figure 4.2: 4-vertex tournaments

The fourth tournament can be expressed as $T_4 = (V; E)$ where $V = \{a, b, c, d\}$ and $E = \{(a, b), (b, c), (c, d), (d, a), (d, b), (c, a)\}$. Note that this tournament does not have a 5-cycle. If it did, a vertex would be visited twice in the cycle, and this would imply the tournament has a 1-cycle or 2-cycle, which is impossible.

We will show that T_4 does not have a WNU polymorphism of arity $n = 3m + 1$ for each choice of m. Suppose f is a WNU polymorphism of T of arity n and consider the tuples $\bar{x}_1 = (b, c, d, ..., b, c, d, a)$, ..., $\bar{x}_n = (a, b, c, d, b, c, d, ..., b, c, d)$. Since f preserves edges, we have that $f(\bar{x}_1)...f(\bar{x}_n)f(\bar{x}_1)$ is an n-cycle. Since T_4 does not have 1-cycles or 2-cycles, and $n \equiv 1 \pmod 3$, we have that $\{f(\bar{x}_1), ..., f(\bar{x}_n)\} = \{a, b, c, d\}$. In T_4^n, there is a 5-path from the tuple $\bar{x}_i$ to the tuple $(a, ..., a, b, a, ..., a)$ where b is in the i-th position, by combining the 5-paths $(a \to b \to c \to d \to a \to b)$,$(b \to c \to a \to b \to c \to a)$, $(c \to d \to a \to b \to c \to a)$,$(d \to a \to b \to c \to d \to a)$. Since f is a WNU polymorphism, we have that $f(b, a, ..., a) = ... = f(a, ..., a, b)$. There is a 5-path in T_4 from every $f(\bar{x}_i)$ to $f(b, a, ..., a)$, and $f(\bar{x}_i) = f(b, a, ..., a)$ for some i, so T_4 has a 5-cycle, which is impossible. Therefore there is no WNU polymorphism of T_4 of arity n. From Theorem 2.13, we have that T_4 does not have bounded width.

Note that if a tournament T has a 4-cycle (namely $a \to b \to c \to d \to a$), then it does not have a 2-ary WNU. If $f : T^2 \to T$ is a WNU of T, then $f(a, c) \to f(b, d) \to f(c, a) = f(a, c)$, which cannot happen in T. Which tournaments do not have a 4-cycle in them?

If a tournament T has an $n-$cycle $a_1...a_n a_1$ with $n > 4$, then from Theorem 1.24 the subtournament induced by $a_1, ..., a_n$ is pancyclic and so it has a 4-cycle. Thus a tournament with no 4-cycles is of the form $T_1 \oplus ... \oplus T_n$, where T_i is either a vertex or a 3-cycle. However,

not every tournament of this form admits a WNU polymorphism. The following theorem was proved independently in [3], but we use different methods in our proof.

Theorem 4.14. *Let $C = \vec{C_3}^{(1)} \oplus \vec{C_3}^{(2)}$, where $\vec{C_3}^{(i)} = a_i \to b_i \to c_i \to a_i$ is a 3-cycle. Then C does not have a WNU polymorphism of arity n for any n.*

To prove this theorem we will use the following lemmas:

Lemma 4.15. *Let $f : C^n \to C$ be a WNU polymorphism of the graph C described in Theorem 4.14. If $\bar{x} = (x_1, ..., x_n) \in (\vec{C_3}^{(i)})^n$, then $f(\bar{x}) \in \vec{C_3}^{(i)}$.*

Proof. Assume without loss of generality that $\bar{x} \in (\vec{C_3}^{(1)})^n$. Then $f(\bar{x}) \to f(a_2, ..., a_2) = a_2$. Similarly, $f(\bar{x}) \to b_2, f(\bar{x}) \to c_2$. Then $f(\bar{x})$ cannot be a_2, b_2 or c_2, so it must be in $\vec{C_3}^{(1)}$. □

For Lemma 4.18 we will need to define the following notation.

Definition 4.16. Let S be a set and let $\bar{x} = (x_1, ..., x_n) \in S^n$. For $a \in S$, define $\boldsymbol{N_a(\bar{x})} = |\{i \in [n] | x_i = a\}|$.

Definition 4.17. Let $x \in \vec{C_3}^{(i)}$. Then $\boldsymbol{x^-}, \boldsymbol{x^+}$ are the vertices in $\vec{C_3}^{(i)}$ such that $x^- \to x \to x^+$.

Lemma 4.18. *Let $k > n \geq 1$ and $f : C^k \to C$ be a WNU polymorphism of the graph C described in Theorem 4.14. For all $x \in C$, if $N_x(\bar{z}) = k - n$ then $f(\bar{z}) = x$*

Proof. We proceed by induction on n. Let $n = 1$. If $N_x(\bar{z}) = k - 1$, then $\bar{z}$ is of the form $(x, ..., x, y, x, ..., x)$. Assume without loss of generality that $x \in \vec{C_3}^{(1)}$ and $\bar{z} = (y, x, ..., x)$. Then we have two cases:

Case 1. If $y \in \vec{C_3}^{(2)}$, then

$$x^- = f(x^-, ..., x^-) \to f(\bar{z}) \to f(y^+, ..., y^+) = y^+.$$

Therefore $f(\bar{z}) \in \{x, y\}$. We have that $f(\bar{z}) \to f(y^+, ..., y^+, y)$, as well as $f(\bar{z}) \to f(y^+, y^{++}, ..., y^{++})$ and $f(y^+, ..., y^+, y^{++}) \to f(y^+, y^{++}, ..., y^{++})$, because f is a WNU polymorphism. By Lemma 4.15,

$$f(y^+, ..., y^+, y), f(y^+, y^{++}, ..., y^{++}) \in \vec{C_3}^{(2)}.$$

Therefore $f(\bar{z})$ is connected to two vertices of $\vec{C_3}^{(2)}$ and must therefore be x.

Case 2. If $y \in \vec{C_3}^{(1)}$, then by lemma 4.15 $f(\bar{z}) \in \vec{C_3}^{(1)}$. Since $f(\bar{z}) \to f(a_2, x^+, ..., x^+)$, and from the previous case $f(a_2, x^+, ..., x^+) = x^+$, we have that $f(\bar{z}) = x$.

Now assume that $f(\bar{z}) = x$ whenever $N_x(\bar{z}) = k - m$ for all $m < n < k$. Suppose that $N_x(\bar{z}) = k - n$, and without loss of generality let us assume that $\bar{z} = (x, ..., x, y_1, ..., y_n)$ and $x \in \vec{C_3}^{(1)}$. We have three cases: either $y_1, ..., y_n \in \vec{C_3}^{(1)}$, or some $y_i \in \vec{C_3}^{(2)}$ but not all y_i are the same, or $y_1 = ... = y_n \in \vec{C_3}^{(2)}$. We will first look at the second case described, followed by the first and third.

Case 1. Assume without loss of generality that $y_1 \in \vec{C_3}^{(2)}$ and $y_1 \neq y_2$; that is, there exists $y \in \vec{C_3}^{(2)} \setminus \{y_1^+\}$ such that $y_2 \to y$. Then $f(x^-, ..., x^-, y_2^-, ..., y_n^-) \to f(\bar{z})$ (since $x^- \to y_1$) and by our hypothesis $f(x^-, ..., x^-, y_2^-, ..., y_n^-) = x^-$. Similarly,

$$f(\bar{z}) \to f(y_1^+, ..., y_1^+, y_2^+, ..., y_n^+) = y_1^+,$$

and

$$f(\bar{z}) \to f(y, ..., y, y_1^+, y, y_3^+, ..., y_n^+) = y.$$

Since $y_1^+ \neq y$, we have that $f(\bar{z}) \notin \vec{C_3}^{(2)}$, and since $x^- \to f(\bar{z})$, we have $f(\bar{z}) = x$.

Case 2. Suppose that every y_i is in $\vec{C_3}^{(1)}$. Then $f(\bar{z}) \in \vec{C_3}^{(1)}$, and from the previous case

$$f(\bar{z}) \to f(x^+, ..., x^+, a_2, b_2, ..., b_2) = x^+,$$

so $f(\bar{z}) = x$.

Case 3. Suppose that $y_1 = ... = y_n \in \vec{C}_3^{(2)}$. Then since $x^- \to f(\bar{z})$ and $f(\bar{z}) \to y_1^+$, we have that $f(\bar{z}) \in \{x, y_1\}$. But $f(\bar{z}) \to f(y_1, ..., y_1, y_1^+, ..., y_1^+)$, and by the previous case we have that $f(y_1, ..., y_1, y_1^+, ..., y_1^+) = y_1$. Therefore $f(\bar{z}) = x$.

□

We are now ready to give the proof for Theorem 4.14.

Proof of Theorem 4.14. Suppose that $f : C^n \to C$ is a WNU polymorphism of C. Then, by lemma 4.18,

$$f(a_1, ..., a_1, b_1) = a_1, f(b_1, a_1, ..., a_1) = b_1$$

which cannot happen since f is a WNU polymorphism.

□

From Theorem 4.14 we conclude that C has a hard CSP, which was found independently in [3].

Corollary 4.19. *CSP(C) is NP-complete, where C is the tournament described in Theorem 4.14.*

The tournaments T_4 and $\vec{C}_3 \oplus \vec{C}_3$ are the base cases for tournaments that have a hard CSP. We can see that any tournament containing T_4 or $\vec{C}_3 \oplus \vec{C}_3$ has a hard CSP with the following theorem:

Theorem 4.20. *[6] Let H be a directed graph with no sources or sinks, and let H′ be a loop-less digraph containing H as a subgraph. If H does not admit a homomorphism to a cycle of length greater than one, then CSP(H′) is NP-complete.*

With this result we can characterize tournaments with bounded width, and note that the class of tournaments have a bounded width dichotomy.

Theorem 4.21. *Let T be a tournament. If T has at most one cycle, then T has bounded width. Otherwise, CSP(T) is NP-complete.*

Proof. Let T be a tournament T with a least two cycles. Then it contains either T_4 or $\vec{C_3} \oplus \vec{C_3}$ as a subtournament. Since neither T_4 nor $\vec{C_3} \oplus \vec{C_3}$ admit a homomorphism to any cycle of length greater than one, it follows from Theorem 4.20 that CSP(T) is NP-complete.

If a tournament T has at most one cycle, then it is of the form

$$T = T_1 \oplus T_2 \oplus ... \oplus T_n$$

where at most one T_i is a copy of $\vec{C_3}$ and every other T_j is a single vertex. From lemma 4.13, we see that T has bounded width. □

4.4 Semi-complete graphs with bounded width

Theorem 4.11 shows that there exists a dichotomy in the CSPs of semi-complete graphs. This result was extended in [4] to a larger class of directed graphs, called locally semi-complete graphs.

Definition 4.22. A directed graph $G = (V; E)$ is **locally semi-complete** if, for every $v \in V$, the sets $\boldsymbol{N^+(v) = \{u \in V | v \to u\}}$ and $\boldsymbol{N^-(v) = \{u \in V | u \to v\}}$ each induce a semi-complete graph.

Remark 4.23. In this section we will be working with semi-complete graphs, which may not be simple. Therefore we will not assume directed graphs to be simple in this Section.

Example 4.24. Let $k \geq 4$. The directed cycle $\vec{C_k} = (\{v_1, \cdots v_k\}; \{(v_1, v_2), \cdots, (v_k, v_1)\})$ is locally semi-complete, since $|N^+(v_i)| = |N^-(v_i)| = 1$ for every $1 \leq i \leq k$, so the subgraphs induced by these sets are trivially semi-complete, as they only contain one vertex. However, the directed cycle $\vec{C_k}$ is not semicomplete, since there are pairs of vertices not connected by an edge (e.g. v_1 and v_3).

Locally semi-complete directed graphs are a natural generalization of semi-complete graphs. Bang-Jensen, MacGillivray and Swarts showed in [4] that locally semi-complete

directed graphs also possess a CSP dichotomy. We will use their construction to show that in fact connected locally semi-complete directed graphs have a bounded width dichotomy.

Lemma 4.25. *[2] A local tournament H is acyclic if and only if it admits an enumeration $v_1, v_2, ..., v_n$ of its vertices such that whenever $v_i \to v_j$, then $i < j$ and $v_k \to v_l$ for all $i \leq k < l \leq j$.*

To state the classification theorem from [4], we need to do the following construction (from [4]):

Definition 4.26. Let H be a loop-free directed graph with vertices $v_1, ..., v_\ell$ and let $D_1, ..., D_\ell$ be directed graphs. Define $\boldsymbol{H[D_1, ..., D_\ell]}$ to be the directed graph obtained by replacing each vertex v_i from H with D_i, and replacing every edge $v_i \to v_j$ with the set of edges $\{d_i \to d_j \mid d_i \in D_i, d_j \in D_j\}$

Let $T = (V; E)$ be a connected locally semi-complete graph with exactly one cycle, and let C be its cycle. Then

$$T = H[D_1, D_2, ..., D_\ell]$$

where H is an acyclic local tournament with an enumeration $h_1, ..., h_\ell$ such as the one from 4.25, $D_j = C$ for some j, and $|D_i| = 1$ for $i \neq j$. If $\ell \geq 2$, then C must be semi-complete, so either $C = \vec{C}_2$ or $C = \vec{C}_3$.

Let S be the set of neighbours of C. Partition $V \setminus S$ into the set of vertices that come before C in the ordering of H (call it A) and the set of vertices that come after C (call it B). Then A induces a subgraph T_1, S induces a subgraph T_2 and B induces a subgraph T_3.

Theorem 4.27. *[4] Let T be a connected locally semi-complete directed graph.*

- *If T is acyclic, then CSP(T) is in P.*
- *If T has exactly one cycle and T is a directed cycle or T_2 is semi-complete and at least one of T_1 or T_3 is empty, then CSP(T) is in P. Otherwise CSP(T) is NP-complete.*

- *If T contains at least two cycles, then CSP(T) is NP-complete.*

We refine this theorem by showing that connected locally semi-complete directed graphs have a bounded width dichotomy.

Theorem 4.28. *Let T be a connected locally semi-complete directed graph.*

- *If T is acyclic, then T has bounded width.*
- *If T has exactly one cycle and T is a directed cycle or T_2 is semi-complete and at least one of T_1 or T_3 is empty, then T has bounded width. Otherwise CSP(T) is NP-complete.*
- *If T contains at least two cycles, then CSP(T) is NP-complete.*

In order to prove Theorem 4.28 we follow the construction from [4] and refine some of their theorems to help us establish the bounded width dichotomy.

Lemma 4.29. *Let T be a connected, acyclic, locally semi-complete directed graph. Then T has a conservative WNU polymorphism of arity k for every $k \geq 2$.*

Proof. Since T is acyclic, from Lemma 4.25 it has an enumeration $v_1 < v_2 < ... < v_n$ of its vertices such that if $v_i \to v_j$, then $v_i < v_j$ and $v_k \to v_l$ for all $i \leq k < l \leq j$. Let $k \geq 2$ and consider the operation $f_k : T^k \to T$ defined by $f_k(x_1, ..., x_k) = \min\{x_1, ..., x_k\}$. Then f_k is clearly conservative, idempotent and has the WNU property. We are left with showing that f_k is a polymorphism.

Suppose that $x_1 \to y_1, x_2 \to y_2, ..., x_k \to y_k$, where $f(x_1, ..., x_k) = x_i = v_{i_1}$ for some i, and $f(y_1, ..., y_k) = y_j = v_{j_2}$ for some j, where $v_{i_1} = x_i \to y_1 = v_{i_2}$ and $v_{j_1} = x_j \to y_j = v_{j_2}$. Then

$$v_{i_1} \leq v_{j_1} < v_{j_2} \leq v_{i_2}$$

so $f_k(x_1, ..., x_k) \to f_k(y_1, ..., y_k)$. □

Corollary 4.30. *Let T be a connected, acyclic, locally semi-complete directed graph. Then T has bounded width.*

Proof. This follows from Lemma 4.29 and Theorem 2.20. □

The enumeration described in Lemma 4.25 is an example of an $\underline{X}-$enumeration, which we will use in the following results. We give a definition:

Definition 4.31. Let H be a directed graph. An enumeration $h_1 < h_2 < ... < h_n$ of the vertices of H is an $\underline{X}-$enumeration if it has the following property: if $h_i \to h_j$ and $h_k \to h_\ell$, then $\min\{h_i, h_k\} \to \min\{h_j, h_\ell\}$.

It was shown in [20] that graphs with an $\underline{X}$-enumeration have tractable CSPs.

Theorem 4.32. *[20] Let H be a directed graph with an $\underline{X}$-enumeration. Then CSP(H) is in P.*

We refine this theorem in terms of existence of WNU polymorphisms. This result was found independently in [32].

Theorem 4.33. *Let H be a directed graph with an $\underline{X}$-enumeration. Then H has a conservative WNU polymorphism of arity k for every $k \geq 2$.*

Proof. Let H have an $\underline{X}$-enumeration $h_1 < h_2 < ... < h_n$. Define $f_k : H^k \to H$ by $f(x_1, ..., x_k) = \min\{x_1, ..., x_k\}$. Clearly f_k is conservative, idempotent and has the WNU property. We are left with showing that f_k is a polymorphism. Suppose that $x_1 \to y_1, ..., x_k \to y_k$, and let $x_i = f(x_1, ..., x_k) = \min\{x_1, ..., x_k\}$, $y_j = f(y_1, ..., y_k) = \min\{y_1, ..., y_k\}$. Then, since $x_i \to y_i$ and $x_j \to y_j$, we have

$$\min\{x_i, x_j\} = x_i \to y_j = \min\{y_i, y_j\},$$

so $f(x_1, ..., x_k) \to f(y_1, ..., y_k)$. □

Corollary 4.34. *Let H be a directed graph with $\underline{X}$-enumeration. Then H has bounded width.*

Proof. This follows from Theorems 4.33 and 2.20. □

We introduce another construction used in [3] that preserves WNU polymorphisms.

Definition 4.35. Let H_1 be a loop-free directed graph with an $\underline{X}$-enumeration $h_1 < ... < h_n$. Let H_2 be a directed graph. Form the directed graph H by replacing h_n with H_2 and adding, for every edge $h_i \to h_n$ (or $h_n \to h_i$), an edge $h_i \to h$ (or $h \to h_i$) for every $h \in H_2$. The graph H is called the $\underline{X}$**-graft**(H_1, H_2).

Example 4.36. Let $H_1 = \vec{L_3} = \{(\{u_1, u_2, u_3\}; \{(u_1, u_2), (u_1, u_3), (u_2, 3)\})$ and $H_2 = \vec{L_2} = (\{v_1, v_2\}; \{(v_1, v_2\})$. Then H_1 has an $\underline{X}$-enumeration $u_1 < u_2 < u_3$, and $H = \underline{X}$**-graft**(H_1, H_2), where

$$H = (\{u_1, u_2, v_1, v_2\}; \{(u_1, u_2), (u_1, v_1), (u_1, v_2), (u_2, v_1), (u_2, v_2), (v_1, v_2)\}),$$

which is isomorphic to $\vec{L_4}$.

Theorem 4.37. *[20] Let $H = \underline{X}$-graft(H_1, H_2) such that CSP(H_2) is in P. Then CSP(H) is in P.*

Theorem 4.37 shows that the $\underline{X}$-graft construction preserves the property of a tractable CSP, since graphs with an $\underline{X}$-enumeration have a tractable CSP. We refine this theorem in terms of existence of WNU polymorphisms. This was found independently in [32].

Theorem 4.38. *Let $H = \underline{X}$-graft(H_1, H_2) such that H_2 has a WNU polymorphism of arity k. Then H has a WNU polymorphism of arity k.*

Proof. Suppose that $f_2 : H_2^k \to H_2$ is a WNU polymorphism. Let $f : H^k \to H$ be defined by

$$f(x_1, ..., x_k) = \begin{cases} f_2(x_1, ..., x_k) & \text{if } x_1, ..., x_k \in H_2 \\ \min\{x_i | x_i \in H_1\} & \text{otherwise.} \end{cases}$$

Clearly f is idempotent and has the WNU property. We are left with showing that f is a polymorphism. Suppose that $x_1 \to y_1, ..., x_k \to y_k$.

If $x_1, ..., x_k, y_1, ..., y_k \in H_2$ or if $x_i, y_j \in H_1$ for some i, j then clearly $f(x_1, ..., x_k) \to f(y_1, ..., y_k)$.

If $x_1, ..., x_k \in H_2$, and $y_i \in H_1$ for some i, let $y_j = f(y_1, ..., y_k) = \min\{y_\ell | y_\ell \in H_1\}$. Since $x_j \to y_j$, with $x_j \in H_2, y_j \in H_1$, we have that $h \to y_j$ for all $h \in H_1$, from the $\underline{X}$-graft construction. Therefore $f(x_1, ..., x_k) \to y_j$.

If $x_i \in H_1$ for some i and $y_1, ..., y_k \in H_2$, let $x_j = f(x_1, ..., x_k) = \min\{x_\ell | x_\ell \in H_1\}$. Since $x_j \to y_j$, with $x_j \in H_1, y_j \in H_2$, we have that $x_j \to h$ for all $h \in H_2$, from the $\underline{X}$-graft construction. Therefore $x_j \to f(y_1, ..., y_k)$. □

Recall from Theorem 2.13 that cores have bounded width precisely when they have WNU polymorphism of all but finitely many arities. It can easily be seen that tournaments (and semi-complete graphs in general) are cores, and it was shown in [4] that connected locally semi-complete directed graphs are cores as well.

Theorem 4.39. *[4] A connected locally semi-complete digraph H is a core.*

We can now give a proof for Theorem 4.28.

Proof of Theorem 4.28. We are left with showing that the connected locally semi-complete graphs with a tractable CSP have bounded width.

If T is acyclic, we know from Lemma 4.25 that its vertices admit an enumeration $v_1 < v_2 < ... < v_n$ such that whenever $v_i \to v_j$, then $i < j$ and $v_k \to v_l$ for all $i \leq k < l \leq j$. Clearly this is an $\underline{X}$-enumeration, so from Theorems 4.33 and 4.39 T has bounded width.

Suppose that T has exactly one cycle. If T is a directed cycle, then T has bounded width from Corollary 4.9. Let us assume that T_2 is semi-complete and one of T_1 or T_3 is empty. Suppose without loss of generality that T_3 is empty. (We can obtain the other case by reversing all the arrows in the graph.) Following the proof of Theorem 4.27, it can be shown that $T = \underline{X} - \text{graft}(T', T'')$, where T' has an $\underline{X}$-enumeration and T'' is a tournament

with exactly one cycle. From Theorem 4.21, T'' has bounded width, and so from Theorems 4.38 and 4.39, T has bounded width. □

4.5 Transitive directed graphs

We give an analysis for another class of directed graphs: transitive graphs.

Definition 4.40. A directed graph $G = (V; E)$ is **transitive** if E is a transitive relation. In other words, for every $u, v, w \in G$, if $u \to v, v \to w$, then $u \to w$.

Since any directed graph with a loop has a trivial CSP, we will focus on studying the CSP of loop-free transitive directed graphs. If a transitive directed graph contains a cycle as a subgraph (not necessarily induced), then by the transitive property it contains a loop. Therefore loop-free transitive directed graphs have no cycles, and are thus equivalent to the class of strict partial orders.

Remark 4.41. While some transitive graphs are not simple, we will be analyzing loop-free transitive graphs, which are simple. We will therefore assume directed graphs to be simple in this Section.

We will prove that loop-free transitive graphs have bounded width by finding a Datalog program that solves their CSP. We will also give a proof that utilizes the associated pebble game.

Theorem 4.42. *Let H be a loop-free transitive directed graph. Then H has bounded width.*

Proof. Let H be a loop-free transitive graph with longest path of length n, on the vertices $v_0 \to v_1 \to ... \to v_n$. Let G be a directed graph. If G contains a path of length more than n (allowing for repetition of vertices, so this includes cycles of any length), then there is no homomorphism from G to H, which can be seen by following the images of the path of length more than n. If G does not contain paths of length more than n, consider the following construction:

Let $G_0 = G$ and let S_0 be the set of vertices in G without an incoming edge in G. Let S_1 be the set of vertices of $G_0 \setminus S_0$ without an incoming edge in $G_1 = [G_0 \setminus S_0]$, the graph induced by the vertices $G_0 \setminus S_0$. For $i \geq 1$, let S_{i+1} be the set of vertices of $G_i \setminus S_i$ without an incoming edge in $G_{i+1} = [G_i \setminus S_i]$. Notice that S_{n+1} is empty. Clearly the sets $S_0, ..., S_n$ form a partition of the vertices of G (otherwise G would contain a cycle).

Let $f : G \to H$ be defined by $f(u) = v_i$ for $u \in S_i$. Then if $u_1 \to u_2$, where $u_1 \in S_i$, $u_2 \in S_j$, clearly $i < j$, so $f(u_1) = v_i \to v_j = f(u_2)$. Therefore f is a homomorphism.

If G has a path of length more than n, then in particular it has a path of length $n + 1$. Therefore there exists a homomorphism from G to H if and only if G does not have a path of length $n + 1$. This can be determined with the following $(2, 3)$-Datalog program π:

$$
\begin{aligned}
&\pi \qquad\qquad (4.1)\\
&P_1(x, y) : -E(x, y)\\
&P_2(x, y) : -P_1(x, z), P_1(z, y)\\
&P_3(x, y) : -P_2(x, z), P_1(z, y)\\
&\quad .\\
&\quad .\\
&\quad .\\
&P_n(x, y) : -P_{n-1}(x, z), P_1(z, y)\\
&FALSE : -P_n(x, z), P_1(z, y)
\end{aligned}
$$

The program computes all the paths of length n on step n, and returns FALSE if and only if there is a path of length $n + 1$. Therefore the program solves CSPH, which implies that H has bounded width.

□

We have seen that loop-free transitive directed graphs have width $(2, 3)$ from the program Π described in 4.1. We can show that in fact these graphs have width $(1, 2)$ by looking at the associated pebble games.

Theorem 4.43. *Let H be a loop-free transitive directed graph, and let G be a directed graph. Then Spoiler wins the $(1, 2)$-pebble game on G and H if and only if there is no homomorphism from G to H.*

Proof. If there is a homomorphism from G to H, then Duplicator wins the game by playing according to the homomorphism. Suppose that H has a longest path of length n and there is no homomorphism from G to H. Then G has a path of length $n + 1$, namely $v_0 \to v_1 \to ... \to v_{n+1}$ (note the v_i need not be different). Then Spoiler can place pebbles along this path, so that on round $i + 1$ there are pebbles on v_{i-1}, v_i, and spoiler takes the pebble from v_{i-1} and places it on v_{i+1}. This way, Duplicator is forced to trace a path of length $i + 1$ on H, so when Spoiler places a pebble on v_{n+1} (if Spoiler has not yet won), Duplicator will be unable to move and will lose the game. □

Corollary 4.44. *Let H be a loop-free transitive directed graph. Then H has width $(1, 2)$.*

Proof. This follows from Theorems 4.43 and 3.3. □

Corollary 4.44 ensures that for any loop-free transitive directed graph H there exists a $(1, 2)$-Datalog program that solves CSP(H). However, it is not necessarily a simple task to find such a program. We see that for loop-free transitive directed graphs, it is more intuitive to analyze the pebble games to establish their bounded width. This shows that pebble games can be a helpful method for verifying that certain structures have bounded width.

Theorem 4.42 can be extended to any acyclic directed graph with a large enough transitive component.

Theorem 4.45. *Let H be an acyclic directed graph with longest path of length n and such that H has a subgraph H' that is isomorphic to $\vec{L}_{n+1}$. Then H has bounded width.*

Proof. If H is an acyclic directed graph with longest path of length n, then there is a homomorphism from H to $\vec{L}_{n+1}$. If H' is a subgraph of H that is isomorphic to $\vec{L}_{n+1}$, then there is a natural homomorphism from $\vec{L}_{n+1}$ to H and therefore H is homomorphically equivalent to $\vec{L}_{n+1}$. Then the Datalog program Π from 4.1 solves CSP(H).

□

4.5.1 Transitive directed graphs and X-enumerations

We know from Theorem 4.33 that directed graphs that are cores and have an $\underline{X}$-enumeration have bounded width. We would like to know if the converse is also true: whether every cycle-free directed graph with bounded width has an $\underline{X}$-enumeration. Note that graphs with a cycle do not admit an $\underline{X}$-enumeration, so we must make the distinction.

Consider enumerations with the property that if $v_i \to v_j$, then $i < j$. We call these **topological enumerations**.

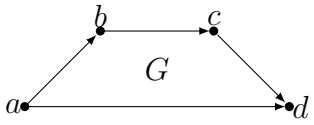


Figure 4.3: A graph with no topological X-enumeration.

Consider the graph G in Figure 4.3. Any topological enumeration must satisfy $a < b < c < d$. However, this is not an $\underline{X}$-enumeration, since $a \to d, b \to c$, but $a \not\to c$. We can allow topological $\underline{X}$-enumerations in G to exist by making G transitive; that is, by adding $u \to w$ every time $u \to v, v \to w$, for any $u, v, w \in G$.

We have shown in Theorem 4.42 that loop-free transitive directed graphs have bounded width. However, not every transitive directed graph has a topological $\underline{X}$-enumeration. Consider the graph H in Figure 4.4. Any topological enumeration of H must satisfy $a < b < c, u < v < w$. If $a < u$, then the crossing $a \to w, u \to w$ shows that this is not an $\underline{X}$-enumeration. If instead $u < a$, the same happens with $u \to c, a \to b$. Therefore H

does not have a topological $\underline{X}$-enumeration.

Even though H does not have topological $\underline{X}$-enumerations, $c < b < a < u < w < v$ is an $\underline{X}$-enumeration (although not topological) of H. This can be checked by inspecting the crossings from Figure 4.5. There are three crossings. The first one corresponds to the edges $u \to v, v \to w$. The crossing is highlighted in blue, and the edge $\min\{u, v\} \to \min\{v, w\}$ is blue as well. The other two crossings are highlighted in red, and they correspond to the pairs of edges $a \to w, u \to c$ and $a \to b, u \to c$. In both cases, the $\underline{X}$-edge corresponds to $a \to c$, which is coloured red in the figure.

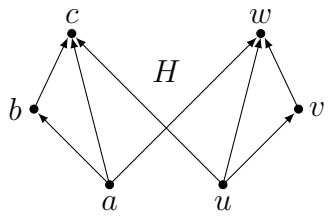


Figure 4.4: A transitive graph with no topological X-enumeration.

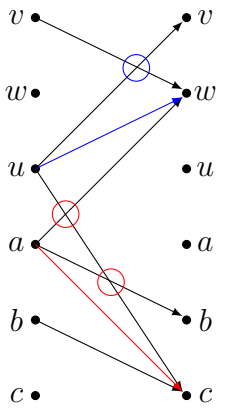


Figure 4.5: Crossings in the X-enumeration of H with X edges highlighted.

It turns out that the class of directed graphs that admit an $\underline{X}$-enumeration coincides with the class of directed graphs that admit a conservative semilattice polymorphism, or equivalently, a conservative, transitive WNU polymorphism. (For a more detailed discussion on these polymorphisms see Section 2.2.1.) This was found independently in [23].

Theorem 4.46. *Let G be a directed graph. Then G admits an $\underline{X}$-enumeration if and only if G admits a binary, conservative, transitive WNU polymorphism.*

Proof. ($\Rightarrow$)Suppose that G admits an $\underline{X}$-enumeration. From Theorem 4.33, G admits a binary, conservative WNU polymorphism $f : G^2 \to G$ defined by

$$f(u, v) = \min\{u, v\}.$$

We will prove that f is a transitive WNU polymorphism. Suppose that $f(u, v) = \min\{u, v\} = u, f(v, w) = \min\{v, w\} = v$. Then

$$\begin{aligned} f(u, w) = \min\{u, w\} &= \min\{\min\{u, v\}, w\} \\ &= \min\{u, \min\{v, w\}\} \\ &= \min\{u, v\} = u. \end{aligned}$$

Therefore f is a transitive polymorphism.

($\Leftarrow$)Suppose that $f : G^2 \to G$ is a conservative, transitive WNU polymorphism. Enumerate the vertices of G by $v_i < v_j$ if and only if $f(v_i, v_j) = v_i$, for $i \neq j$ (this enumeration is well-defined because f is conservative, transitive and has the WNU property). Suppose that $v_i \to v_j$ and $v_k \to v_\ell$. Since f is a polymorphism,

$$\min\{x_i, x_k\} = f(x_i, x_k) \to f(x_j, x_\ell) = \min x_j, x_\ell.$$

Therefore the enumeration is an $\underline{X}$-enumeration. □

Even though directed graphs only admit an $\underline{X}$-enumeration when they admit a transitive WNU polymorphism, not every transitive graph admits an $\underline{X}$-enumeration. Consider the graph H and its transitive closure H' (see Figure 4.6). The directed graph H has an $\underline{X}$-enumeration $a < b_1 < b_2 < c_1 < c_2 < d$, which can be easily checked, since the enumeration

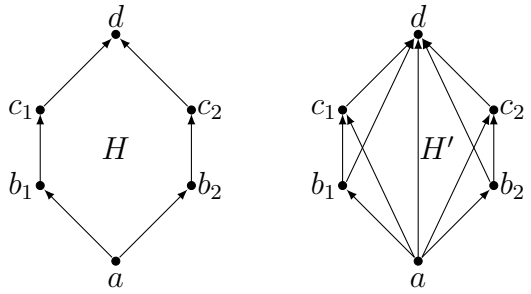


Figure 4.6: The directed graph H and its transitive closure H'.

has no crossings. However, its transitive closure, H', has no $\underline{X}$-enumeration. We have checked this with the help of a program that determines for every enumeration of the vertices of H' whether it is an $\underline{X}$-enumeration. See Appendix B for the program written in C++. The graph H' gives us an example of a directed graph with no cycles that has bounded width, but no $\underline{X}$-enumeration.

Thanks to Hell and Rafiey's classification of bi-arc graphs from [23] and Theorem 4.46 we can see that the class of directed graphs that admit a $\underline{X}$-enumeration corresponds to the class of bi-arc directed graphs.

Theorem 4.47. *[23] The class of bi-arc directed graphs coincides with the class of directed graphs that a admit an $\underline{X}$-enumeration.*

Proof. This follows from Theorems 2.26, 2.28 and 4.46. □

4.6 Directed graphs that do not allow a binary conservative WNU polymorphism

We know from Theorem 2.20 that graphs that admit a binary conservative WNU (BCWNU) polymorphism have bounded width. The converse is not always true. For example, we know from Lemma 4.9 that the directed graph

$$\vec{C_4} = (\{1,2,3,4\}; \{(1,2)(2,3)(3,4)(4,1)\})$$

has bounded width. However, this graph does not admit any binary polymorphism, since any such polymorphism f must satisfy

$$f(1,3) \to f(2,4) \to f(3,1) = f(1,3),$$

which is impossible. Note that in general, if a directed graph G does not admit a BCWNU polymorphism, then any directed graph H which contains G as an induced subgraph will also not admit a BCWNU polymorphism.

We are interested in determining which directed graphs share this property of having bounded width but not admitting a BCWNU polymorphism. We are not interested in graphs with loops, since these have a trivial CSP. Note that a directed graph with a double edge (i.e. with vertices a, b such that $a \to b$ and $b \to a$) does not allow a BCWNU polymorphism unless the graph has a loop. With the help of a computer program, we found all directed graphs without loops or double edges, that have at most 4 vertices and do not admit a BCWNU polymorphism. They are the following: (see Figure 4.6).

$$\vec{C_4} = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(4,1)\})$$

$$R_4 = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(1,4)\})$$

$$G = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(1,4),(4,2)\})$$

$$H = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(4,1),(4,2)\})$$

$$T_4 = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(4,1),(3,1),(4,2)\})$$

We have shown in Theorems 4.9 and 4.21 that the directed cycle $\vec{C}_4$ has bounded width, while the tournament $T_4 = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(4,1),(3,1),(4,2)\})$ has an NP-complete CSP.

Let us analyze the second directed graph from the list, R_4. We will show that it has bounded width by finding a 3-ary NU polymorphism for it. Let $f_3 : R_4^3 \to R_4$ be such that

$$f_3(x,x,y) = f_3(x,y,x) = f_3(y,x,x) = x \text{ for all } x, y \in R_4$$

and be defined in the following way when x, y, z are pairwise different:

$$f_3(x,y,z) = \begin{cases} 1 & \text{if } \{x,y,z\} = \{1,2,3\} \\ 4 & \text{if } \{x,y,z\} = \{2,3,4\} \\ x & \text{otherwise.} \end{cases}$$

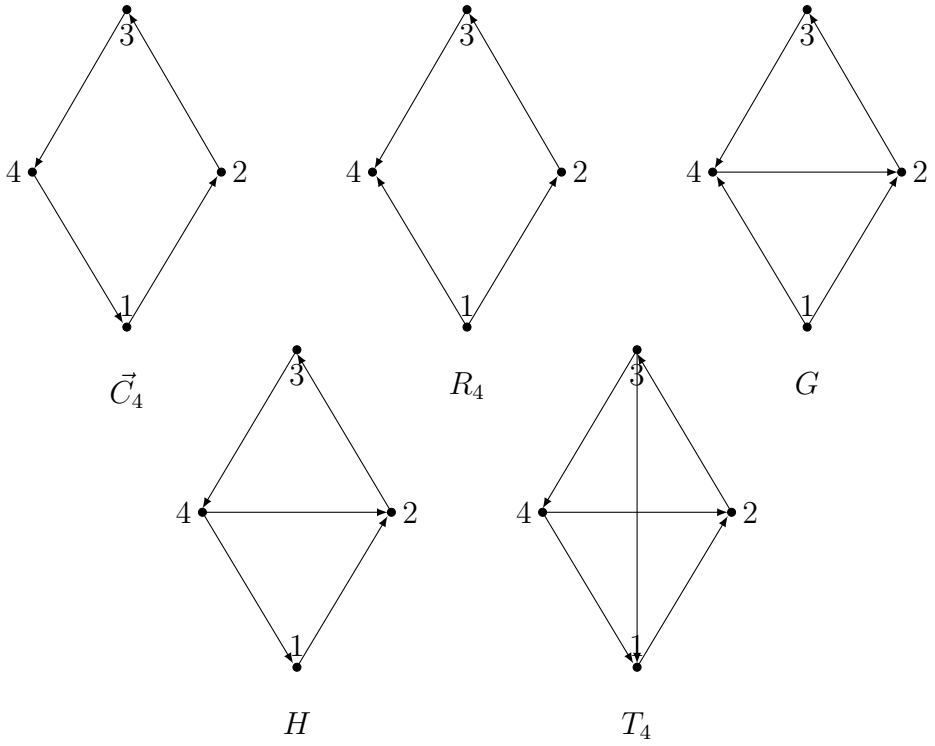
Clearly, f_3 is a NU operation, since it is a majority function. To see that f_3 is a polymorphism, let $\bar{x} = (x_1, x_2, x_3)$ and $\bar{y} = (y_1, y_2, y_3)$ be such that $x_1 \to y_1, x_2 \to y_2, x_3 \to y_3$. Then we have that either $\bar{x}$ has a repeated component, or $\{x_1, x_2, x_3\} = \{1,2,3\}$. Similarly, either $\bar{y}$ has a repeated component, or $\{y_1, y_2, y_3\} = \{2,3,4\}$. We then have 4 cases:

Case 1. If $\bar{x}$ and $\bar{y}$ have repeated components, then clearly $f_3(\bar{x}) = f_3(\bar{y})$, since f_3 is a majority function.

Case 2. If $\bar{x}$ has a repeated component, and $\{y_1, y_2, y_3\} = \{2,3,4\}$, then the repeated component of $\bar{x}$ is connected to at least two of the vertices 2, 3, 4. Since the only vertex in R_4 that is connected to two different vertices is 1, we have that $f_3(\bar{x}) = 1$, and since $f_3(\bar{y}) = 4$, we have $f_3(\bar{x}) \to f_3(\bar{y})$.

Case 3. If $\{x_1, x_2, x_3\} = \{1,2,3\}$ and $\bar{y}$ has a repeated component, then at least two of the vertices 1,2,3 are connected to the repeated component of $\bar{y}$. Since the only vertex in R_4 that is connected from two different vertices is 4, we have that $f_3(\bar{y}) = 4$, and since $f_3(\bar{x}) = 1$, we have $f_3(\bar{x}) \to f_3(\bar{y})$.

Figure 4.7: 4-vertex directed graphs that do not admit a binary conservative WNU polymorphism.



Case 4. If $\{x_1, x_2, x_3\} = \{1, 2, 3\}$ and $\{y_1, y_2, y_3\} = \{2, 3, 4\}$, then clearly $f_3(\bar{x}) \to f_3(\bar{y})$.

Since R_4 has a NU polymorphism, and R_4 is a core, we have from Theorem 2.17 that R_4 has bounded width.

We will now look at the next two graphs in our list, G and H. We will prove that both of these graphs have a hard CSP.

Let $G = (\{1, 2, 3, 4\}; \{(1, 2), (1, 4), (2, 3), (3, 4), (4, 2)\})$ (see Figure 4.6). We will show CSP(G) is NP-complete, by showing that it has no WNU polymorphisms. Note that G is a core.

Notation: Given a polymorphism $f : G^n \to G$, we will write $x_1x_2 \cdots x_n$ for $f(x_1, x_2, \cdots, x_n)$. Furthermore, if the tuple $\bar{x} = (x_1, \cdots, x_1, x_2, \cdots, x_2)$ has k copies of x_1 and ℓ copies of x_2, we will write $\bar{x} = x_1^k x_2^\ell$. We write $x_1x_2 \cdots x_n =^* x$ if

$$f(x_{\sigma(1)}, x_{\sigma(2)}, \cdots, x_{\sigma(n)}) = x \text{ for all } \sigma \in S_n,$$

Figure 4.8: Base case for Theorem 4.48.

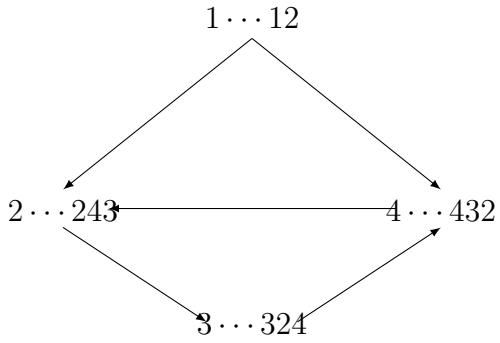
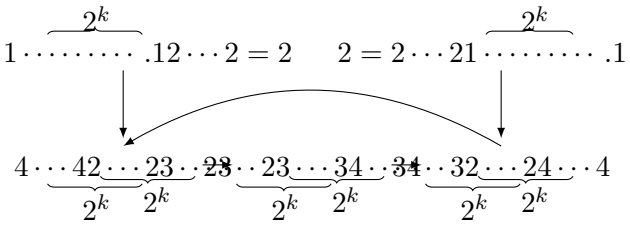


Figure 4.9: Case 2 in Theorem 4.48.



that is, the formula holds for any permutation of the tuple $(x_1, \cdots, x_n)$. Therefore, since f is a WNU polymorphism, we have $x^k y =^* x^k y$ for all $x, y \in G$.

To prove that CSP(G) is NP-complete, we need the following Lemma.

Theorem 4.48. *Let $G = (\{1,2,3,4\}; \{(1,2),(1,4),(2,3),(3,4),(4,2)\})$. Then G does not have a WNU polymorphism of arity n for any $n \geq 2$.*

Proof. We prove that G does not have WNU polymorphisms by contradiction. Let $f : G^n \to G$ be a WNU polymorphism. We will first prove that such an operation has the following property:

Claim: Let $k \geq 0$ be such that $2^k < n$. Then $1^{n-2^k}2^{2^k} =^* 1$ and $2^{n-2^k}1^{2^k} =^* 2$.

We proceed by induction.

For $k = 0$, we have that $1^{n-1}2 =^* 1^{n-1}2$, so $1^{n-1}2 \to 4^{n-2}23$ and $1^{n-1}2 \to 2^{n-2}34$. Since $4^{n-2}23 \to 2^{n-2}34 \to 3^{n-2}42 \to 4^{n-2}23$, we have that $1^{n-1}2$ is connected to two different vertices in a 3-cycle, so $1^{n-1}2 =^* 1$ (see Figure 4.6). Furthermore, $3^{n-1}4 \to 4^{n-1}2 \to 2^{n-1}3 \to 3^{n-1}4$, and since $1 = 1^n \to 4^{n-1}2$, $1 = 1^{n-1}2 \to 2^{n-1}3$, we have that $3^{n-1}4 =^* 3$. Since

$2^{n-1}1 \to 3^{n-1}4 =^* 3$, we have that $2^{n-1}1 =^* 2$.

Let $k > 0$, with $2^k < n$, and assume that $1^{n-2^{k-1}}2^{2^{k-1}} =^* 1$ and that $2^{n-2^{k-1}}1^{2^{k-1}} =^* 2$. Since

$$4^{n-2^k}2^{k-1}3^{k-1} \to 2^{n-2^k}3^{k-1}4^{k-1} \to 3^{n-2^k}4^{k-1}2^{k-1} \to 4^{n-2^k}2^{k-1}3^{k-1},$$

and

$$1 = 1^{n-2^{k-1}}2^{2^{k-1}} \to 4^{n-2^k}2^{k-1}3^{k-1}$$

as well as

$$1 = 1^{n-2^k}2^{2^{k-1}}1^{2^{k-1}} \to 2^{n-2^k}3^{k-1}4^{k-1},$$

then $3^{n-2^k}4^{k-1}2^{k-1} = *3$. Given that $2^{n-2^k}1^{2^k} \to 3^{n-2^k}4^{k-1}2^{k-1}$, we get that $2^{n-2^k}1^{2^k} = *2$. From this, we see that $3^{n-2^k}4^{2^k} = *3$, $4^{n-2^k}2^{2^k} = *4$, $2^{n-2^k}3^{2^k} = *2$, as well as $3^{n-2^k}2^{2^k} = *3$, $4^{n-2^k}3^{2^k} = *4$. Since

$$1^{n-2^k}2^{2^k} \to 2^{n-2^k}3^{2^k}$$

and

$$1^{n-2^k}2^{2^k} \to 4^{n-2^k}3^{2^k},$$

we see that $1^{n-2^k}2^{2^k} =^* 1$.

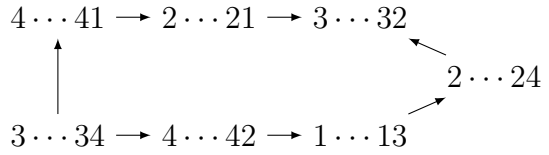
We can now go back to our proof by contradiction. Let $n \geq 2$ and let $f : G^n \to G$ be a WNU polymorphism. We have two cases: either $n = 2^k$ for some k, or there exists k such that $2^k < n, n < 2^{k+1}$.

Case 1. If $n = 2^k$ for some n, then from $1^{2^{k-1}}2^{2^{k-1}} =^* 1$ and $2^{2^{k-1}}1^{2^{k-1}} =^* 2$, which is impossible. Therefore G does not have a WNU polymorphism of arity n.

Case 2. Let k be such that $2^k < n, n < 2^{k+1}$. Then we have a 3-cycle with vertices

$$4^{n-2^k}2^{2^{k+1}-n}3^{n-2^k} \to 2^{n-2^k}3^{2^{k+1}-n}4^{n-2^k} \to 3^{n-2^k}4^{2^{k+1}-n}2^{n-2^k}.$$

Figure 4.10: Base case for Theorem 4.50.



However,

$$2 = 1^{2^k}2^{n-2^k} \to 4^{n-2^k}2^{2^{k+1}-n}3^{n-2^k}$$

and

$$2 = 2^{n-2^k}1^{2^k} \to 3^{n-2^k}4^{2^{k+1}-n}2^{n-2^k},$$

which is impossible, since the vertex 2 is not connected to two different vertices in a 3-cycle (see Figure 4.6). Therefore G does not have a WNU polymorphism of arity n. □

Corollary 4.49. *Let $G = (\{1,2,3,4\}; \{(1,2),(1,4),(2,3),(3,4),(4,2)\})$. Then CSP(G) is NP-complete.*

We now analyze the fourth graph on our list.

Let $H = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(4,1),(4,2)\})$ (see Figure 4.6). We will show that H has a hard CSP by showing it does not admit any WNU polymorphisms.

Notation: when working with this directed graph, given a vertex x we will write x^+ to denote the next vertex in the cycle $1 \to 2 \to 3 \to 4 \to 1$, and x^- to denote the previous vertex. For example $3^+ = 4, 3^- = 2$. We will write $x^{++} = x^{+^+}$, so $4^{++} = 2$.

Theorem 4.50. *Let $H = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(4,1),(4,2)\}$. Then H does not have a WNU polymorphism of arity n for any $n \geq 2$.*

Proof. We prove that H has no WNU polymorphism by contradiction. Let $n \geq 3$ and $f : H^n \to H$ be a WNU polymorphism. We will show that such an operation has the following property:

Claim: $x^{n-k}x_1x_2\cdots x_k =^* x$ for $0 \leq k < n, x, x_1, \cdots, x_n \in H$.

We proceed by induction. Clearly, the statement is true for $k = 0$ since f is idempotent.

For $k = 1$, since $4^k1 \to 2$, either $4k^1 =^* 4$ or $4^k1 =^* 1$. Suppose that $4^k1 =^* 1$. Then

$$2^k1 =^* 2, 3^k2 =^* 3, 2^k1 =^* 2.$$

Additionally, $3^k4 =^* 4$, but $3^k4 \to 4^k2 \to 1^k3 \to 2^k4$, and there is no such path from 4 to 2. Therefore $4^k1 =^* 4$. Then $3^k4 =^* 3$ and so $4^k2 =^* 4$. This implies that $3^k1 =^* 3, 2^k4 =^* 2, 3^k2 =^* 3$, so $4^k3 =^* 4$. We conclude that $4^kx =^* 4$ for all $x \in H$. Therefore $3^kx =^* 3$ and $2^kx =^* 2$ for all $x \in H$. This implies that $1^kx =^* 1$ for all $x \in H$.

Now let $k \geq 1$ and suppose that $x^{n-k}x_1 \cdots x_k =^* x$ for all $x, x_1, \cdots, x_k \in H$. We will show that $x^{n-k-1}x_1 \cdots x_{k+1} =^* x$ for all $x, x_1, \cdots, x_{k+1} \in H$ by first showing that $4^{n-k-1}x_1 \cdots x_{k+1} =^* 4$ for all $x, x_1, \cdots, x_{k+1} \in H$.

Since, by assumption, $4^{n-k}3x_1 \cdots x_{k-1} =^* 4$ and $32^{n-k}x_1^+ + \cdots x_{k-1}^+ + =^* 2$ for all $x_1, \cdots, x_{k-1} \in H$, and we have that

$$4^{n-k}3x_1 \cdots x_{k-1} \to 21^{n-k-1}4x_1^+ \cdots x_{k-1}^+ \to 32^{n-k}x_1 + + \cdots x_{k-1} + +$$

we conclude that $1^{n-k-1}24x_1^+ \cdots x_{k-1}^+ + =^* 1$. Therefore

$$2^{n-k-1}3yx_1^{++} \cdots x_{k-1}^{++} =^* 2$$

$$3^{n-k-1}4y^+x_1^- \cdots x_{k-1}^- =^* 3$$

$$4^{n-k-1}1y^{++}x_1 \cdots x_{k-1} =^* 4$$

$$4^{n-k-1}2y^{++}x_1 \cdots x_{k-1} =^* 4$$

for all $x, x_1, \cdots, x_{k+1} \in H$ and $y \in \{1, 2\}$. Note that $y^{++} \in \{3, 4\}$. We are then left with showing that $4^{n-k-1}x_1 \cdots x_{k+1} =^* 4$ when $x_1, \cdots, x_{k+1} \in \{1, 2\}$ and when $x_1, \cdots, x_{k+1} \in \{3, 4\}$.

Since $3^{n-k-1}42x_1..x_{k-1} =^* 3$ for all for all $x, x_1, \cdots, x_{k-1} \in H$, we have that

$$2^{n-k-1}34x_1^-..x_{k-1}^- =^* 2$$

$$3^{n-k-1}41x_1..x_{k-1} =^* 3$$

$$4^{n-k-1}y2x_1^+..x_{k-1}^+ =^* 4,$$

where $y \in \{1,2\}$. Furthermore, $3^{n-k-1}4^{k+1} =^* 3$ and so $4^{n-k-1}1^{k+1} =^* 4$. Since we know by our induction hypothesis that $4^{n-k-1}x_1 \cdots x_{k+1} = *4$ when $x_i = 4$ for some i, we are left with showing that $4^{n-k-1}3^{k+1} =^* 4$.

Since $4^{n-k-1}2^{k+1} =^* 4$, we have

$$3^{n-k-1}1^{k+1} =^* 3$$

$$2^{n-k-1}4^{k+1} =^* 2$$

$$3^{n-k-1}2^{k+1} =^* 3$$

$$4^{n-k-1}3^{k+1} =^* 4.$$

This completes our proof that $4^{n-k-1}x_1 \cdots x_{k+1} = *4$ for all $x_1, \cdots, x_{k+1} \in H$. Therefore $3^{n-k-1}x_1^- \cdots x_{k+1}^- = *3$, $2^{n-k-1}x_1^{++} \cdots x_{k+1}^{++} = *2$, and $1^{n-k-1}x_1^+ \cdots x_{k+1}^+ = *1$ for all $x_1, \cdots, x_{k+1} \in H$, so we have $x^{n-k-1}x_1 \cdots x_{k+1} =^* x$ for all $x, x_1, \cdots, x_{k+1} \in H$.

We can now finish our proof by contradiction. Let $n \geq 2$ and let $f : H^n \to H$ be a WNU polymorphism. Then, for any $1 < k < n$, $1^{n-k}2^k =^* 1$ and $1^{n-k}2^k =^* 2$, which is impossible. □

Since H is a core, Theorem 4.50 guarantees that H has a hard CSP.

Corollary 4.51. *Let* $H = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(4,1),(4,2)\}$. *Then CSP(H) is NP-complete.*

We conclude that the only directed graphs with 4 vertices, that do not have loops or

double edges, and have bounded width but no BCWNU polymorphism, are $\vec{C}_4$ and R_4. We know from Lemma 4.9 that, in general, $\vec{C}_{2k}$ has bounded width. However, $\vec{C}_{2k}$ does not admit any binary WNU polymorphisms, since any such polymorphism f must satisfy

$$f(1,k+1) \to f(2,k+2) \to \cdots \to f(k,2k) \to f(k+1,1) = f(1,k+1),$$

which is impossible since there is no homomorphism from $\vec{C}_k$ to $\vec{C}_{2k}$. We will obtain a similar generalization for R_4.

Let $R_n = (\{1,\cdots,n\}; \{(1,2),(2,3),\cdots,(n-1,n),(1,n)\})$ be a directed n-cycle with one edge reversed. We will prove that, for $k \geq 2$, the graph R_{2k} has bounded width, but does not admit a BCWNU polymorphism.

Suppose $f : {R_{2k}}^2 \to R_{2k}$ is a WNU polymorphism. Then we have that

$$f(1,k) \to f(2,k+1) \to \cdots \to f(k+1,2k)$$

and that

$$f(1,k) \to f(2k,k+1) = f(k+1,2k)$$

therefore we have that R_{k+1} is a subgraph of R_{2k}, which is impossible. We see that R_{2k} does not admit a binary WNU polymorphism.

Theorem 4.52. *The directed graph R_{2k} has bounded width for $k \geq 2$.*

Proof. Let $k \geq 2$. Since R_{2k} is a core, it is enough to prove that R_{2k} admits a NU polymorphism, from Theorem 2.17.

Let $f : {R_{2k}}^3 \to R_{2k}$ be such that

$$f(x,x,y) = f(x,y,x) = f(y,x,x) \text{ for all } x,y \in R_{2k}$$

and is defined in the following way when x,y,z are pairwise different:

$$f(x,y,z) = \begin{cases} 2k & \text{if } \{2,2k\} \subseteq \{x,y,z\} \\ 1 & \text{if } \{1,2k-1\} \subseteq \{x,y,z\} \\ x & \text{otherwise.} \end{cases}$$

We will say that a tuple (x,y,z) is of type 1 if $x = y, x = z$ or $y = z$. We will say that (x,y,z) is of type 2 if it is not of type 1 and either $\{2,2k\} \subseteq \{x,y,z\}$ or $\{1,2k-1\} \subseteq \{x,y,z\}$. We will say that that (x,y,z) is of type 3 if it is not of type 1 or 2.

Suppose $x_1 \to x_2, y_1 \to y_2, z_1 \to z_2$. Note that this implies that either the tuples (x_1,y_1,z_1) and (x_2,y_2,z_2) are of the same type, or one is of type 1 and the other is of type 2.

If both tuples are of the same type, then clearly $f(x_1,y_1,z_1) \to f(x_2,y_2,z_2)$. If (x_1,y_1,z_1) is of type 1 and (x_2,y_2,z_2) is of type 2, then it must be the case that $\{2,2k\} \subseteq \{x_2,y_2,z_2\}$, since 1 is not connected from any vertex of R_{2k}. Then $f(x_1,y_1,z_1) = 1$ and so $f(x_1,y_1,z_1) \to f(x_2,y_2,z_2)$. If (x_1,y_1,z_1) is of type 2 and (x_2,y_2,z_2) is of type 1, then it must be that $\{1,2k-1\} \subseteq \{x_1,y_1,z_1\}$, since $2k$ is not connected to any vertex of R_{2k}. Then $f(x_2,y_2,z_2) = 2k$ and so $f(x_1,y_1,z_1) \to f(x_2,y_2,z_2)$. Therefore f is a NU polymorphism.

□

Chapter 5

Conclusion

While the CSP dichotomy conjecture is widely believed to be true, the notion of bounded width provides the possibility of formulating and studying refined and stronger versions of the dichotomy. While there exist directed graphs with a tractable CSP that do not have bounded width, we find that many popular classes of graphs exhibit a bounded width dichotomy. The class of tournaments, as well as the more general class of connected locally semi-complete directed graphs, have a bounded width dichotomy. Transitive directed graphs have bounded width.

There are many tools that can help us determine whether a structure has bounded width, including looking for special polymorphisms, as well as analyzing some pebble games and Datalog programs. While the presence of a semilattice polymorphism determines that a structure has bounded width, this only works for a reduced class of directed graphs: bi-arc graphs. The presence of a binary, conservative WNU is still sufficient to determine that a structure has bounded width, and this criterion works for a larger class of directed graphs, including odd directed cycles. The complete graphs K_n, $n \geq 3$ are known to have an NP-complete CSP; however, we found that the Mycielski graphs explicitly witness their lack of bounded width through the associated pebble games.

We have by no means resolved all the problems that naturally arise. We discuss some

questions left unanswered, which could form the basis for future work.

Question 5.1. *If S is a substructure of Δ and Δ has a tractable CSP (bounded width) is it true that S has a tractable CSP (bounded width)?*

This is true for simple graphs, but not in general. A simple counterexample is the following:

Let $S = (V; E) = K_3$, and let $\Delta = (V; E')$, where $E' = V^2$, that is, the (not simple) graph obtained by adding to K_3 a loop at every vertex. Note that $CSP(S)$ is NP-complete, while CSP(Δ) is trivial (every finite graph maps to it).

However, some special cases can be studied. We give an example

Proposition 5.2. *Let $G = (V; E)$ be a (simple) graph, with $v \notin V$ and let $G' = (V'; E')$, where $V' = V \cup \{v\}$, $E' = E \cup \{(g, v)|g \in V\} \cup \{(v, g)|g \in V\}$. That is, G' is obtained by adding a vertex v to G and an edge from every vertex of G to v. Then CSP(G) can be reduced to CSP(G').*

Proof. Let $\mathcal{S} = (S; E_S)$ be an instance of CSP(G). Create an instance $\mathcal{S}' = (S'; E_{S'})$ of CSP(G') by adding a vertex v' to $\mathcal{S}$ and an edge from every vertex of $\mathcal{S}$ to v'. Then there is a homomorphism from $\mathcal{S}$ to G if and only if there is a homomorphism from $\mathcal{S}'$ to G'. To see this, suppose that there exists $f : \mathcal{S} \to G$. We can extend f to a homomorphism $f' : \mathcal{S}' \to G'$ by making $f'(v') = v$. Next, suppose that there exists a homomorphism $f' : \mathcal{S}' \to G'$. If $f'(v') = v$, then the image of S under f' is contained in S, so the restriction of f' to S is a homomorphism from $\mathcal{S}$ to G. Otherwise, if $f'(v') = s \in S$, then $(s, f'(t)) \in E'$ for all $t \in S$. Therefore G' has an isomorphism g that swaps v and s, and fixes every other vertex of G. Then $f \circ g : \mathcal{S}' \to G'$ is a homomorphism and $f(g(v')) = v$, which is our previous case.

□

We see that we can saturate the relations of a structure to make a hard problem tractable. We then refine our question:

Question 5.3. *If S is an induced substructure of Δ and Δ has a tractable CSP (bounded width) is it true that S has a tractable CSP (bounded width)?*

This is true for simple graphs. Hell and Nešetřil proved in [22] that the CSP of a graph G is tractable if and only if G is bipartite. If G is bipartite, any (induced) subgraph of G is also bipartite, and so will have a tractable CSP. A bipartite graph G will have the same CSP as K_2 if G has an edge, and the same CSP as $\bar{K_2}$ if G has no edges. Both K_2 and $\bar{K_2}$ have bounded width, so G has bounded width as well. Therefore a graph G has bounded width if and only if G is bipartite, so if G has bounded width any induced subgraph of it will have bounded width. Simple graphs have a bounded width dichotomy.

Question 5.3 is still false for (not simple) graphs. If we construct a graph G by adding a loop to a vertex of K_4, then G has a trivial CSP, but K_3 is an induced subgraph of G.

We can consider Question 5.3 for the class of tournaments. We can reduce this question to the following one:

Question 5.4. *Let $T = (V; E)$ be a tournament with a WNU polymorphism of T. Does T have a conservative WNU polymorphism of arity n? That is, is there a WNU polymorphism $f : V^n \to V$ of T such that*

$$f(x_1, ..., x_n) \in \{x_1, ..., x_n\} \text{ for all } x_1, ..., x_n \in V?$$

A positive answer to this question would imply that if a tournament T has bounded width, then any subtournament of T has bounded width as well. This is because, since all tournaments are cores, if T has bounded width then it has a WNU of all but finitely many arities. For each of these arities n, there is a WNU polymorphism g_n with the above property. If S is a subtournament of T, then $g_n|_S$ is a WNU of S of arity n, and so S has bounded width.

Alternatively, we can try to answer Question 5.3 for tournaments by studying the associated pebble games. From Theorem 3.2, we know that a tournament T has bounded width

if and only if there exists k such that, for all directed graphs G, Spoiler wins the k-pebble game on G and T if and only if there is no homomorphism from G to T.

Suppose a tournament T has bounded width (assume that $\neg\text{CSP}(T)$ is expressible in k-Datalog). Given a subtournament T' of T, does T' have bounded width? Or, in particular, is $\neg\text{CSP}(T')$ expressible in k-Datalog? To prove that we must prove that, for any directed graph G', Spoiler wins the k-pebble game on G' and T' if and only if there is no homomorphism from G' to T'. If there is no homomorphism from G' to T, then there is no homomorphism from G' to T', and since T has bounded width, Spoiler wins the k-pebble game on G' and T. This winning strategy naturally produces a winning strategy for Spoiler in the k-pebble game on G' and T'. If there is a homomorphism from G' to T', then Duplicator wins the k-pebble game on G' and T'. Therefore we are left with the case where there is no homomorphism between G' and T', but there is one between G' and T. We can assume without loss of generality that T' has precisely one fewer vertex than T, since then we can iterate this argument to obtain any desired subtournament of T. Suppose that $T' = (V'; E')$ and $T = (V' \cup \{v\}; E)$. If there is no homomorphism from G' to T', but there is one from G' to T, can we construct a directed graph G by adding a vertex u to G' such that there is no homomorphism from G to T? If this is the case, then Spoiler wins the k-pebble game on G and T, and in particular Spoiler wins the $k+1$-pebble game on G and T with an unmovable pebble on $u \in G$ and $v \in T$, since this game would be at least as hard for Duplicator to win as the k-pebble game on G and T. With Spoiler's winning strategy in this game we might be able to recover a winning strategy for Spoiler in the k-pebble game on G' and T'.

However, consider the following example:

Example 5.5. Let $T' = \vec{L_5}$ with vertices $v_1, ..., v_5$ (where $v_i \to v_j$ if and only if $i < j$) and let T have vertices $v_1, ..., v_5, v$ where $v_1 \to v, v \to v_2, v_4 \to v, v \to v_5$ (either $v_3 \to v$ or $v \to v_3$ will work). If $G' = \vec{C_3}$, say $u_1 \to u_2 \to u_3 \to u_1$, then clearly there is no homomorphism from G' to T' and there is a homomorphism from G' to T since G' is a subtournament of G.

However, we cannot add a vertex to G' to create a graph G that does not map to T, since T has every 4-vertex tournament as a subtournament. So we see that given G' that does not map to T' but maps to T, we cannot always find a G that does not map to T by adding a vertex to G'.

Theorem 4.21 answers Question 5.4, since the only tournaments that admit a WNU polymorphism are the ones that contain at most one cycle, which have conservative WNU polymorphism of arity k for every $k \geq 2$. However, the question still remains open for (simple) directed graphs.

Question 5.6. *Let $G = (V; E)$ be a (simple) directed graph and $f : G^n \to G$ a WNU polymorphism of G. Does G have a conservative WNU polymorphism of arity n?*

Consider the graph $G = \vec{C}_1 \cup \vec{C}_2 = (\{0, 1, 2\}; \{(0, 0), (1, 2), (2, 1)\})$. This graph does not have a binary conservative WNU polymorphism, since $\vec{C}_2$ does not admit a BCWNU polymorphism. However, the operation $f : G^2 \to G$ defined by

$$f(x, y) = \begin{cases} x & \text{if } x = y \\ 0 & \text{otherwise.} \end{cases}$$

is a WNU polymorphism of G. We can also find an example that does not contain loops. Consider the graph $H = \vec{C}_3 \cup \vec{C}_6$, the disjoint union of a copy of $\vec{C}_3$ and a copy of $\vec{C}_6$. This graph does not have a binary conservative WNU polymorphism, since $\vec{C}_6$ does not admit a BCWNU polymorphism. However, we know that there exists a binary WNU $f : \vec{C}_3 \to \vec{C}_3$ and a homomorphism $h : \vec{C}_6 \to \vec{C}_3$. From these, we can create a homomorphism $h' : \vec{C}_3 \cup \vec{C}_6 \to \vec{C}_3$ defined by

$$h'(x) = \begin{cases} x & \text{if } x \in \vec{C}_3 \\ h(x) & \text{if } x \in \vec{C}_6 \end{cases}$$

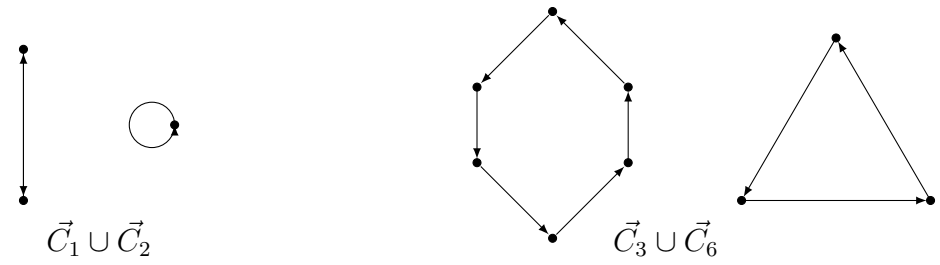


Figure 5.1: Directed graphs with a WNU polymorphism but no conservative WNU polymorphism of the same arity.

and a polymorphism $f' : H^2 \to H$ defined by

$$f'(x,y) = \begin{cases} x & \text{if } x = y \\ f(h'(x), h'(y)) & \text{otherwise.} \end{cases}$$

We see that there are directed graphs (with or without loops) with a WNU polymorphism of arity 2, that do not have a conservative WNU of the same arity (see Figure 5.1). We do not know if this is true for connected directed graphs, or for other arities.

In Section 4.6 we found that the directed graphs R_{2k} have bounded width, but do not admit a BCWNU polymorphism. The only directed graphs on at most 4 vertices, without loops or double edges, that have bounded width and do not admit a BCWNU polymorphism are $\vec{C}_4$, and R_4. Are there graphs outside of these families with similar properties?

Question 5.7. *Is there a directed graph without loops or double edges, not having either of $\vec{C}_{2k}, R_{2k}$, $k \geq 2$, as an induced subgraph, that has bounded width but does not admit a BCWNU polymorphism?*

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[2] Jørgen Bang-Jensen. Locally semicomplete digraphs: A generalization of tournaments. *Journal of Graph Theory*, 14:371–390, 1990.

[3] Jørgen Bang-Jensen, Pavol Hell, and Gary MacGillivray. The complexity of colouring by semicomplete digraphs. *SIAM J. Discret. Math.*, 1(3):281–298, August 1988.

[4] Jørgen Bang-Jensen, Gary MacGillivray, and Jacobus Swarts. The complexity of colouring by locally semicomplete digraphs. *Discrete Mathematics*, 310(20):2675 – 2684, 2010. Graph Theory — Dedicated to Carsten Thomassen on his 60th Birthday.

[5] Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, January 2014.

[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 J. Comput.*, 38(5):1782–1802, January 2009.

[7] Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *CoRR*, abs/0809.2386, 2008.

[8] Manuel Bodirsky and Jaroslav Nešetřil. Constraint satisfaction with countable homogeneous templates. *J. Logic Comput.*, 16(3):359–373, 2006.

[9] Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Logic*, 12(4):24:1–24:66, July 2011.

[10] Andrei A. Bulatov. Conservative constraint satisfaction re-revisited. *Journal of Computer and System Sciences*, 82(2):347 – 356, 2016.

[11] Andrei A. Bulatov and Víctor Dalmau. A simple algorithm for maltsev constraints. *SIAM J. Comput.*, 36(1):16–27, 2006.

[12] Andrei A. Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, March 2005.

[13] Jakub Bulin, Dejan Delic, Marcel Jackson, and Todd Niven. On the reduction of the CSP dichotomy conjecture to digraphs. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 184–199, 2013.

[14] Jakub Bulin, Dejan Delic, Marcel Jackson, and Todd Niven. A finer reduction of constraint problems to digraphs. *Logical Methods in Computer Science*, 11(4), 2015.

[15] Hubie Chen. Logic column 17: A rendezvous of logic, complexity, and algebra. *CoRR*, abs/cs/0611018, 2006.

[16] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. *Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics*, pages 310–326. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[17] David Eppstein. https://oeis.org/A122695. Number of edges in the n-th Mycielski graph.

[18] 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, 1998.

[19] David Geiger. Closed systems of functions and predicates. *Pacific J. Math.*, 27:95–100, 1968.

[20] Wolfgang Gutjahr, Emo Welzl, and Gerhard Woeginger. Polynomial graph-colorings. *Discrete Applied Mathematics*, 35(1):29 – 45, 1992.

[21] Frank Harary and Leo Moser. The theory of round robin tournaments. *The American Mathematical Monthly*, 73(3):231–246, 1966.

[22] Pavol Hell and Jaroslav Nešetřil. On the complexity of h-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92 – 110, 1990.

[23] Pavol Hell and Arash Rafiey. Bi-arc digraphs and conservative polymorphisms. *CoRR*, abs/1608.03368, 2016.

[24] Wilfrid Hodges. *Model Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1993.

[25] Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200(1):185 – 204, 1998.

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

[27] Phokion G. Kolaitis and Moshe Y. Vardi. On the expressive power of datalog: Tools and a case study. *Journal of Computer and System Sciences*, 51(1):110 – 134, 1995.

[28] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302 – 332, 2000.

[29] Phokion G. Kolaitis and Moshe Y. Vardi. A game-theoretic approach to constraint satisfaction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 175–181. AAAI Press, 2000.

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

[31] Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra universalis*, 56(3):439–466, Jun 2007.

[32] Gary MacGillivray and Jacobus Swarts. Weak near-unanimity functions and digraph homomorphism problems. *Theoretical Computer Science*, 477:32 – 47, 2013.

[33] Gary MacGillivray, Jacobus Swarts, and Jørgen Bang-Jensen. A graph theoretic proof of the complexity of colouring by a local tournament with at least two directed cycles. *Contributions to Discrete Mathematics*, 6(2), 2011.

[34] Miklós Maróti and Ralph McKenzie. Existence theorems for weakly symmetric operations. *Algebra universalis*, 59(3):463–489, Dec 2008.

[35] John W. Moon. *Topics on tournaments*. Athena series: Selected topics in mathematics. Holt, Rinehart and Winston, 1968.

[36] Jan Mycielski. Sur le coloriage des graphs. *Colloquium Mathematicae*, 3(2):161–162, 1955.

[37] Jakob Nordstrom. Pebble Games, Proof Complexity, and Time-Space Trade-offs. *ArXiv e-prints*, July 2013.

[38] Christos H. Papadimitriou. Computational complexity. In *Encyclopedia of Computer Science*, pages 260–265. John Wiley and Sons Ltd., Chichester, UK.

[39] 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.

[40] Eric W. Weisstein. https://oeis.org/A266550. Independence number of the n-Mycielski graph.

[41] Eric W. Weisstein. https://oeis.org/A137890. Number of (directed) Hamiltonian paths in the n-Mycielski graph.

[42] Eric W. Weisstein. https://oeis.org/A143247. Number of (directed) Hamiltonian circuits in the Mycielski graph of order n.

[43] Eric W. Weisstein. https://oeis.org/A234625. Numbers of undirected cycles in the n-Mycielski graph.

[44] Eric W. Weisstein. https://oeis.org/A193148. Numbers of spanning trees of the Mycielski graphs.

[45] Eric W. Weisstein. https://oeis.org/A287432. Number of connected dominating sets in the n-Mycielski graph.

Appendix A

Pebble Game program

The following C++ program computes the number of rounds it takes for Spoiler to win in a $(k-1,k)$-pebble game played on C_n (an odd cycle of length n) and K_2, for $n < 14$. It recursively analyzes game positions and stores how many rounds are needed for Spoiler to win at a given position. If at the current position the associated partial homomorphism is not valid, the game is considered to be won by Spoiler and so it needs zero more rounds for Spoiler to win.

```
#include<iostream>
#include<list>
#include<string>

using namespace std;
int mem1[2000000][10][2];
int mem2[2000000][10][10][2];

int play1(string s, int level);

int play2(string s, int p, int level);
list<string> L;
int k;
```

```
int n;
int conv(string s)
{
    int num=0;
    int power=1;
    for (int i=0; i<n;i++)
    {
        num+= (power*(int(s[i])-int('0')));
        power*=3;
    }
    return num;
}
int iph(string s, int i, int p)
{
    int aux1 = (i-1)%n;
    int aux2 = (i+1)%n;
    if (int(s[aux1])-int('0')==p || (int(s[aux2])-int('0')==p))
    return 0;
    else return 1;
}
int genlist(string s, string sub, int l)
    {
                    if (l==k) return 0;
                    else if (sub.length()==n)
                    {
                          int test=0;
                          for (int i=0;i<n;i++)
                          { if (sub[i] != '0') test++;}
                          if (test !=0) L.push_back(sub);
                    }
                    else if (l==k-1)
                    {
                          for (int i=sub.length(); i<n;i++)
```

```
                {
                    sub.push_back('0');
                }

                L.push_back(sub);
            }


              else
              {
                  string cur = sub;
                  cur.push_back(s[sub.length()]);
                  if (s[sub.length()] == '0')
                  {
                     genlist(s,cur,l);
                  }
                  else
                  {
                      string aux = cur;
                      aux[sub.length()]='0';
                      genlist(s,aux,l);
                      genlist(s,cur,l+1);
                  }
              }
            return 0;
    }

int play1(string s,int level)
    {
            if (mem1[conv(s)][level][0]==1)
            {return mem1[conv(s)][level][1];}
            int assigned=0;
            for (int i=0; i< n; i++)
```

```
{
    if (s[i]!='0') {assigned++;}
}
if (assigned<k)
{
                int aux1;
                int min1=50;
                for (int i=0; i<n;i++)
                {
                    if (s[i]=='0')
{aux1=play2(s,i,level); min1= (min1<aux1? min1 : aux1);}
                }
                mem1[conv(s)][level][0]=1;
                mem1[conv(s)][level][1]=min1;
                return min1;
}
if (assigned ==k)
{
          int min2=50;
          int aux2;
          string curstring;
          list<string> auxlist;
          genlist(s, "", 0);
          auxlist=L; L.clear();
          for (list<string>::iterator it=auxlist.begin();
          it != auxlist.end(); ++it)
          {
               curstring = *it;
               for (int i=0; i<n; i++)
               {
                    if (curstring[i]=='0')
                    {
                        aux2=
```

```
play2(curstring, i, level+1);
                                        min2=
(min2<aux2? min2 :aux2);
                                        }
                                    }
                                }
                                mem1[conv(s)][level][0]=1;
                                mem1[conv(s)][level][1]=1+min2;
                                return 1+min2;
                        }

    }

int play2(string s, int p, int level)

{
    if (level ==5) {return 50;}
    string hold=s;
    if (mem2[conv(s)][p][level][0]==1) {return  mem2[conv(s)][p][level][1];}
    if (iph(s,p,1)==0 && iph(s,p,2)==0) return 1;
    if (iph(s,p,1)==0 && iph(s,p,2)==1)
    {
      s[p]='2'; int ret1=play1(s,level);
      mem2[conv(hold)][p][level][0]=1;
      mem2[conv(hold)][p][level][1] = ret1; return ret1;
    }
    if (iph(s,p,1)==1 && iph(s,p,2)==0)
    {
      s[p]='1'; int ret2= play1(s,level);
mem2[conv(hold)][p][level][0]=1;
      mem2[conv(hold)][p][level][1] = ret2 ; return ret2;
    }
    else
```

```
    {
        string auxs = s;
        auxs[p]='1';
        s[p]='2';
        int val1=play1(auxs,level);
        int val2=play1(s,level);
        int ret3=(val1>val2? val1 : val2);
        mem2[conv(hold)][p][level][0]=1;
        mem2[conv(hold)][p][level][1]=ret3;
        return ret3;
    }


}

    int main()
    {

        k=3;
        n=9;
        cout <<"k = "<< k<< endl<< "n     value"<<endl;
        for (int odd=1; odd<7; odd++)
        {

        n=2*odd+1;

        memset(mem1,0,sizeof(mem1));
        memset(mem2,0,sizeof(mem2));
        string start="";
        for (int i=0;i<n;i++) {start.push_back('0');}
            cout <<n << "      " <<play1(start,1)<<endl;
        }
```

```
    system("PAUSE");
    return 0;
}
```

It returns the following table

n	value
3	1
5	2
7	2
9	3
11	3
13	3

Appendix B

Program for finding X-enumerations in H'

The following program goes through every enumeration of the vertices of H' (from Figure 4.6) and prints the ones that are $\underline{X}$-enumerations. Since it does not print any enumeration, we conclude that H' has no $\underline{X}$-enumeration. The function isX takes a permutation of the vertices of H' and checks whether $\min\{h_i, h_k\} \to \min\{h_j, h_\ell\}$ every time $v_i \to v_j$ and $v_k \to v_\ell$.

```
#include <iostream>
#include <string>
using namespace std;
int isPerm(int a, int b, int c, int d, int e, int f)
    {   //checks whether abcdef is a permutation of 123456.
        int check=1;
        check*=a-b;
        check*=a-c;
        check*=a-d;
        check*=a-e;
        check*=a-f;
        check*=b-c;
        check*=b-d;
        check*=b-e;
```

```
        check*=b-f;
        check*=c-d;
        check*=c-e;
        check*=c-f;
        check*=d-e;
        check*=d-f;
        check*=e-f;
        if (check!= 0) {return 1;}
        else              {return 0;}
    }

int isEdge(int a, int b)
    {   //checks whether ab is an edge of H'
        if ((a==1 && b!=1 ) || (a==2 && b==4)
         || (a==3 && b==5) ||  (a!=6 && b==6))
        {return 1;}
        else return 0;
    }

int isX(int a, int b, int c, int d, int e, int f)
    {    // checks if abcdef is an X-enum
        for (int i=1; i<6;i++)
        {  // we check for crossings of the form i->j,
           // k->l, assuming i<k, l<j.
            for (int l=1; l<6; l++)
            {
                for (int k=i+1;k<7;k++)
                {
                    for (int j=l+1; j<7;j++)
                    {
                        if (isEdge(i,j)==1 && isEdge(k,l)==1
                                && isEdge(i,l)==0)
                        {return 0;}
```

```
                    }
                }
            }
        }
        return 1;
    }
int main()
{
  for (int i1=1; i1<7;i1++)
  {
      for (int i2=1; i2<7;i2++)
      {
          for (int i3=1; i3<7;i3++)
          {
              for (int i4=1; i4<7;i4++)
              {
                  for (int i5=1; i5<7;i5++)
                  {
                      for (int i6=1; i6<7;i6++)
                      {
                          if (isPerm(i1,i2,i3,i4,i5,i6)==1
                                && isX(i1,i2,i3,i4,i5,i6)==1)
                          {
                               cout<< "The permutation " << i1
                               << i2 << i3<< i4<< i5<< i6
                                << " is an X-enumeration"<<endl;
                          }
                      }
                  }
              }    //this iterated loop checks for every element
          }        //of [6]^6 if it is a valid X-enumeration
      }            //and prints the valid ones.
  }
```

```
    return {0};
}
```

Appendix C

Computer program for finding 4-vertex directed graphs without a BCWNU polymorphism.

The following computer program, written in Python, finds all 4-vertex directed graphs, without loops or double edges, that do not admit a binary, conservative WNU polymorphism. It lists them by their incidence matrix, and counts them by "type", which counts how many vertices have in-degree 0,1,2,3, and how many vertices have out-degree 0,1,2,3.

```
def create(matrix, source):
    matrix[0][1]=source[0][0]
    matrix[0][2]=source[0][1]
    matrix[0][3]=source[0][2]
    matrix[1][0]=source[1][0]
    matrix[1][2]=source[1][1]
    matrix[1][3]=source[1][2]
    matrix[2][0]=source[2][0]
    matrix[2][1]=source[2][1]
    matrix[2][3]=source[2][2]
    matrix[3][0]=source[3][0]
    matrix[3][1]=source[3][1]
```

```
    matrix[3][2]=source[3][2]
    return

def checktype(matrix, typearray): #returns 1 if matrix has type typearray, 0 otherwise
    array=[0,0,0,0,0,0]
    for x in range (0,4):
        count=0
        for y in range(0,4):
            if matrix[x][y]==1:
                count+=1
        if count==0:
            array[0]+=1
        if count==1:
            array[1]+=1
        if count==2:
            array[2]+=1
    for x in range (0,4):
        count=0
        for y in range(0,4):
            if matrix[y][x]==1:
                count+=1
        if count==0:
            array[3]+=1
        if count==1:
            array[4]+=1
        if count==2:
            array[5]+=1
    if array==typearray:
        return 1


def dotype(matrix):
    array=[0,0,0,0,0,0]
```

```
    for x in range (0,4):
        count=0
        for y in range(0,4):
            if matrix[x][y]==1:
                count+=1
        if count==0:
            array[0]+=1
        if count==1:
            array[1]+=1
        if count==2:
            array[2]+=1
    for x in range (0,4):
        count=0
        for y in range(0,4):
            if matrix[y][x]==1:
                count+=1
        if count==0:
            array[3]+=1
        if count==1:
            array[4]+=1
        if count==2:
            array[5]+=1
    print(array)
    return

def print43(matrix):
    for x in range(0,4):
        print('{0} {1} {2}\n'.format(matrix[x][0], matrix[x][1],
                matrix[x][2]))
    return

def issimple(matrix):
    for i in range (0,3):
```

```
        for j in range (i+1,4):
            if matrix[i][j]==1 and matrix[j][i]==1:
                return 0 #we have a 2-cycle
    return 1 #we dont have 2-cycles

def print44(matrix):
    for x in range(0,4):
        print('{0} {1} {2} {3}\n'.format(matrix[x][0], matrix[x][1],
                matrix[x][2], matrix[x][3]))
    return

def ispoly(f, G):
    for a in range(0,3): #we check for all instances of a->c, b->d
        for b in range(a+1,4): #we assume wlog that a<b
            for c in range(0,3):  #we analyze the case c<d
                for d in range(c+1,4):
                    if G[a][c]==1 and G[b][d]==1:
                        if f[a][b]==0 and f[c][d]==1 and G[a][d]==0:
                            return 0 #it is not a polymorphism
                        if f[a][b]==1 and f[c][d]==0 and G[b][c]==0:
                            return 0 #it is not a polymorphism
            for d in range(0,3): #we analyze the case c>d
                for c in range(d+1,4):
                    if G[a][c]==1 and G[b][d]==1:
                        if f[a][b]==0 and f[c][d]==0 and G[a][d]==0:
                            return 0 #it is not a polymorphism
                        if f[a][b]==1 and f[c][d]==1 and G[b][c]==0:
                            return 0 #it is not a polymorphism
    return 1 #if we didnt find any problems, it is a polymorphism
graph=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
auxgraph=[[0,0,0],[0,0,0],[0,0,0],[0,0,0]]
poly=[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
graphcount=0
```

```
for x in range(0,4096):
    d=2048
    i=0
    j=0
    for m in range(0,12):
        if x>=d:
                x=x-d
                auxgraph[i][j]=1
        else:
            auxgraph[i][j]=0
        j+=1
        if j==3:
            j=0
            i+=1
        d=d/2
    create(graph,auxgraph)
    if issimple(graph)==1:
        marker=0
        for y in range(0,64):
            div=32
            k=0
            l=0
            for m in range(0,16):
                if k==l:
                    poly[k][l]=0
                elif k>l:
                    poly[k][l]=poly[l][k]
                elif y>=div:
                    y=y-div
                    poly[k][l]=1
                else:
                    poly[k][l]=0
                if k<l:
```

```
                div=div/2
            l+=1
            if l==4:
                l=0
                k+=1

    if ispoly(poly,graph)==1:
        marker=1
if marker==0:
    print44(graph)
    dotype(graph)
    graphcount+=1
```

Appendix D

A 4-ary WNU polymorphism for the directed graph R_4.

In Section 4.6 we saw that the graph $R_4 = (\{1,2,3,4\}; \{(1,2),(2,3),(3,4),(1,4)\})$ has bounded width, since it has a 3-ary NU polymorphism, and this implies that it has a a $k-$ary NU polymorphism for every $k \geq 3$. It is natural to ask, however, whether this graph allows other kinds of WNU polymorphisms.

We were able to find a 4-ary WNU polymorphism for R_4 by taking a candidate function and changing its image by one value at a time until a an algorithm confirmed it was a WNU polymorphism. To describe the polymorphism, we need to define the following function:

Let $maj(x_1, x_2, x_3, x_4)$ be equal to the most repeated component of the tuple (x_1, x_2, x_3, x_4), if there is one, or zero otherwise. Note that $maj(x_1, x_2, x_3, x_4) = 0$ only if x_1, x_2, x_3, x_3 are pairwise different or when there are two different repeated components in (x_1, x_2, x_3, x_4), that is, $x_{\sigma(1)} = x_{\sigma(2)}$, $x_{\sigma(3)} = x_{\sigma(4)}$, $x_{\sigma(1)} \neq x_{\sigma(3)}$ for some $\sigma \in S_4$.

We now describe a 4-ary WNU polymorphism of R_4. Let $f_4 : F_4^4 \to R_4$ be such that

$$
\begin{aligned}
f_4(1,2,2,3) &= 1 \\
f_4(1,2,3,2) &= 1 \\
f_4(1,3,2,2) &= 1 \\
f_4(3,1,2,2) &= 1 \\
f_4(3,2,1,2) &= 1 \\
f_4(3,2,2,1) &= 1 \\
f_4(2,4,3,3) &= 2 \\
f_4(2,3,4,3) &= 2 \\
f_4(2,3,3,4) &= 2 \\
f_4(4,2,3,3) &= 2 \\
f_4(4,3,2,3) &= 2 \\
f_4(4,3,3,2) &= 2
\end{aligned}
$$

and is defined in the following way for other tuples:

$$
f_4(x_1,x_2,x_3,x_4) = \begin{cases} x_1 & \text{if } maj(x_1,x_2,x_3,x_4) = 0 \\ maj(x_1,x_2,x_3,x_4) & \text{otherwise.} \end{cases}
$$

We used a computer program to check that f_4 is a WNU polymorphism of R_4.

The reason why the function f_4 needs to have different values from maj on particular tuples comes from the fact that f_4 needs to satisfy the following requirements:

$$
f_4(1,2,2,3) \to f_4(4,3,3,4), f_4(1,2,2,1) \to f_4(2,3,3,4).
$$

In this case, $maj(1,2,2,3) = 2$ but $maj(4,3,3,4) = 4$, and $maj(1,2,2,1) = 1$, but $maj(2,3,3,4) = 3$. The same is true for certain permutations of the coordinates of these tuples.

We used the following Python program to check that f_4 is a polymorphism. It stores the images of f_4 in the 4-dimensional list 'poly4'. For example, the image of the tuple x_1, x_2, x_3, x_4 is stored in poly4[x_1][x_2][x_3][x_4]. It then checks for every pair of tuples (x_1, x_2, x_3, x_4), (y_1, y_2, y_3, y_4) such that $x_i \to y_i, 1 \leq i \leq 4$, whether $f_4(x_1, x_2, x_3, x_4) \to f_4(y_1, y_2, y_3, y_4)$ and outputs 'it is a polymorphism of the graph' if

this is the case.

```
graph=[[0,1,0,1],[0,0,1,0],[0,0,0,1],[0,0,0,0]]

def isPoly4(f, G): #checks if a 3ary operation is a polymorphism of a graph G
    for a1 in range(0,4):
        for a2 in range(0,4):
            for b1 in range(0,4):
                for b2 in range(0,4):
                    for c1 in range(0,4):
                        for c2 in range(0,4):
                            for d1 in range(0,4):
                                for d2 in range(0,4):
                                    value1=f[a1][b1][c1][d1]
                                    value2=f[a2][b2][c2][d2]
                                    if G[a1][a2]==1 and G[b1][b2]==1
                                    and G[c1][c2]==1 and G[d1][d2]==1
                                    and G[value1][value2]==0:
                                        print(a1,b1,c1,d1, a2,b2,c2,d2)
                                        print(value1, value2)
                                        return 0
    return 1

def dominating(a,b,c,d):
    n=[0,0,0,0]
    n[a]+=1
    n[b]+=1
    n[c]+=1
    n[d]+=1
    if n[0]>n[1] and n[0]>n[2] and n[0]>n[3]:
        return 0
    if n[1]>n[0] and n[1]>n[2] and n[1]>n[3]:
        return 1
```

```
    if n[2]>n[1] and n[2]>n[0] and n[2]>n[3]:
        return 2
    if n[3]>n[1] and n[3]>n[2] and n[3]>n[0]:
        return 3
    return 4


def size(a,b,c,d):
    if a==b and b==c and c==d:
        return 1
    elif (a==b and b==c)or(a==b and b==d)or(a==c and c==d)or(b==c and c==d):
        return 2
    elif (a==b and c==d) or (a==c and b==d) or (a==d and b==c):
        return 2
    elif a==b or a==c or a==d or b==c or b==d or c==d:
        return 3
    else:
        return 4
dim=4
poly4 = [[[[0 for l in xrange(dim)] for k in xrange(dim)]
for j in xrange(dim)] for i in xrange(dim)]
poly4[0][0][0][0]
for i in xrange(4):
    for j in xrange(4):
        for k in xrange(4):
            for l in xrange(4):
                if size(i,j,k,l)==3:
                    if dominating(i,j,k,l)==0:
                        poly4[i][j][k][l]=0
                    if dominating(i,j,k,l)==1:
                        poly4[i][j][k][l]=1
                    if dominating(i,j,k,l)==2:
                        poly4[i][j][k][l]=i
                    if dominating(i,j,k,l)==3:
```

```
                    poly4[i][j][k][l]=3
                if size(i,j,k,l)==4:
                    poly4[i][j][k][l]=i
                else:
                    if dominating(i,j,k,l)==0:
                        poly4[i][j][k][l]=0
                    if dominating(i,j,k,l)==1:
                        poly4[i][j][k][l]=1
                    if dominating(i,j,k,l)==2:
                        poly4[i][j][k][l]=2
                    if dominating(i,j,k,l)==3:
                        poly4[i][j][k][l]=3
                    if dominating(i,j,k,l)==4:
                        poly4[i][j][k][l]=i
poly4[1][3][2][2]=1
poly4[1][2][3][2]=1
poly4[1][2][2][3]=1
poly4[0][1][1][2]=0
poly4[0][1][2][1]=0
poly4[0][2][1][1]=0
poly4[3][1][2][2]=1
poly4[3][2][1][2]=1
poly4[3][2][2][1]=1
poly4[2][0][1][1]=0
poly4[2][1][0][1]=0
poly4[2][1][1][0]=0

if isPoly4(poly4, graph)==1:
    print('it is a polymorphism of the graph')
```