Theory of Computing Systems
https://doi.org/10.1007/s00224-022-10091-y

The Complexity of the Distributed Constraint Satisfaction Problem

Silvia Butti[1] · **Víctor Dalmau**[1]

Accepted: 29 May 2022
© The Author(s) 2022

Abstract

We study the complexity of the Distributed Constraint Satisfaction Problem (DCSP) on a synchronous, anonymous network from a theoretical standpoint. In this setting, variables and constraints are controlled by agents which communicate with each other by sending messages through fixed communication channels. Our results endorse the well-known fact from classical CSPs that the complexity of fixed-template computational problems depends on the template's invariance under certain operations. Specifically, we show that DCSP(Γ) is polynomial-time tractable if and only if Γ is invariant under symmetric polymorphisms of all arities. Otherwise, there are no algorithms that solve DCSP(Γ) in finite time. We also show that the same condition holds for the search variant of DCSP. Collaterally, our results unveil a feature of the processes' neighbourhood in a distributed network, its iterated degree, which plays a major role in the analysis. We explore this notion establishing a tight connection with the basic linear programming relaxation of a CSP.

This article belongs to the Topical Collection: *Theoretical Aspects of Computer Science (STACS 2021)*
Guest Editors: Markus Bläser and Benjamin Monmege

The project that gave rise to these results received the support of a fellowship from "la Caixa" Foundation (ID 100010434). The fellowship code is LCF/BQ/DI18/11660056. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sk lodowska-Curie grant agreement No. 713673.

Victor Dalmau was supported by MICCIN grants TIN2016-76573-C2-1P and PID2019-109137GB-C22.

✉ Silvia Butti
silvia.butti@upf.edu

Víctor Dalmau
victor.dalmau@upf.edu

[1] Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain

Published online: 08 July 2022

1 Introduction

The Constraint Satisfaction Problem (CSP) consists of a collection of variables and a collection of constraints where each constraint specifies the valid combinations of values that can be taken simultaneously by the variables in its scope. The goal is to decide if there exists an assignment of the elements of a domain to the variables which satisfies all constraints. The CSP is a very rich mathematical framework that is widely used both as a fruitful paradigm for theoretical research, and as a powerful tool for applications in AI, such as scheduling and planning [19, 23].

While, in its full generality, the finite-domain CSP is known to be NP-complete, applying specific restrictions on the instances can yield tractable subclasses of the problem. One of the most studied approaches consists in requiring that, in each constraint, the set of allowed combinations for its values be drawn from a prescribed set Γ, usually called the constraint language or the template. Thanks to the proof of the CSP dichotomy conjecture obtained separately in [10] and [30], which culminated a decades-long research program, it is possible to determine the complexity (P or NP-complete) of each family of CSPs, CSP(Γ), which is obtained by fixing Γ. This proof confirmed that the complexity of the constraint satisfaction problem is deeply tied to certain algebraic properties of the constraint language. Specifically, it depends on whether or not the constraint language is invariant under certain operations known as its polymorphisms. The polymorphisms of a constraint language enforce a symmetry on the space of solutions of a CSP instance that can possibly be exploited by an algorithm. This connection with algebra is also present in our work.

We study the computational complexity of the distributed counterpart of CSP, which is known as DCSP. This was introduced by Yokoo et al. [27] as a formal framework for the study of cooperative distributed problem solving. In particular, we consider a deterministic, synchronous, anonymous network of agents controlling variables and constraints, and we study the complexity of message passing algorithms on this network. A number of practical applications can be encoded in the DCSP model, for instance resource allocation tasks in wireless networks, routing, networking, and mobile technologies (see for instance [7, 11]).

We notice that this framework is general enough to encompass some simple Graph Neural Network architectures (see for example [15, 22]). In particular, when training a GNN to classify graphs, it is customary that the GNN network ignores the node label when updating its feature vector. This is, in fact, essential as otherwise there would be no way to apply the network trained on a given graph to another one. However, whereas in all variants of GNNs the computation is limited to a reduced number of operations over feature vectors, in the DCSP model the computation at each node is governed by an arbitrary algorithm. GNNs have a wide range of applications including molecule classification or image classification (see [6] for example). Recently, GNNs have been deployed to solve CSPs [24].

While there are a variety of well-performing distributed algorithms for constraint satisfaction and optimisation (see for instance [12, 21, 29]), the theoretical aspects of distributed complexity are to date not well understood. In this paper we initiate the

study of the complexity of DCSP parametrized by the constraint language, obtaining a complete characterization of its tractable classes. More specifically, building on the connection between the CSP and algebra, we show that for any finite constraint language Γ, the decision problem for DCSP(Γ) is tractable whenever Γ is invariant under symmetric polymorphisms of all arities, where an operation is symmetric if its result does not depend on the order of its arguments. Otherwise, there are no message passing algorithms that solve DCSP(Γ). Collaterally, we show that the same holds for the search problem for DCSP.

Our work begins with the identification of a feature of the nodes in a distributed network, its iterated degree, which plays a major role in how messages are transmitted in the network. The iterated degree is an extension of the similar concept introduced in the study of the isomorphism problem which turns out to have a variety of alternative characterizations in terms of fractional isomorphisms, the Weisfeiler-Leman test, and definability with counting logics (see [15]). It turns out that, due to the network anonymity, in every distributed algorithm all equivalent agents (with respect to iterated degree) must necessarily behave identically at each round. A similar phenomenon has been observed independently in the context of GNNs in [22, 25] leading to further study in [3].

We use this fact to show that, under the absence of symmetric polymorphisms of any arity in Γ, it is always possible to construct two instances of DCSP(Γ), one satisfiable and the other unsatisfiable, that cannot be distinguished by any message passing algorithm in an anonymous network.

On the other hand, invariance under symmetric polymorphisms is connected with the basic linear programming relaxation of a CSP instance. More precisely, if Γ has symmetric polymorphisms of all arities then one can decide the satisfiability of every instance of CSP(Γ) by checking whether its basic linear programming relaxation is feasible (see for instance [5]). Whereas it is not clear how to directly use this fact to obtain a distributed algorithm for DCSP(Γ), it can be applied to establish a structure theorem that unveils a simple yet surprising structure in the solution space of every satisfiable instance in DCSP(Γ): it must contain a solution that assigns the same value to all variables that have the same iterated degree. The proof of the structure theorem uses the weighted majority algorithm, a weight update method that is widely used in optimisation and machine learning applications (see [2]). The structure theorem is key in the proof of the positive results as it allows to run an adapted variant of the jpq-consistency algorithm [18] that overcomes the absence of unique identifiers for the variables, by using instead their iterated degree.

This paper is organised as follows. In Section 2 we introduce some definitions and technical concepts about the DCSP model. In Section 3 we present the basic LP relaxation for CSPs and we show its connection to the symmetry on the solution space, culminating in the statement of the structure theorem. Section 4 is dedicated to the proof of the dichotomy theorem for the complexity of DCSP, with the hardness results in Section 4.1, the details of the distributed algorithm for tractable languages in Section 4.2, and its extension to the search problem in Section 4.3. Finally, in the Conclusion we discuss some directions into which our work could be extended.

2 Preliminaries

Constraint Satisfaction Problems An instance I of the finite-domain *Constraint Satisfaction Problem* (CSP) is a triple (X, D, C) where X is a set of variables, D is a finite set called the domain, and C is a set of constraints where a constraint $c \in C$ is a pair $(\mathbf{s}, R)$ where $R \subseteq D^k$ is a relation over D of positive integer arity k, and $\mathbf{s}$ is a tuple of k variables, known as the *scope* of c. We use $arity(\cdot)$ to denote the arity of a relation, tuple, or constraint and we write $x \in c$ for any variable x in the scope of c. An *assignment* $\nu : X \to D$ is said to be *satisfying* if for all constraints $c = (\mathbf{s}, R) \in C$ we have $\nu(\mathbf{s}) \in R$, where ν is applied to $\mathbf{s}$ coordinate-wise. Usually we denote the number of variables by n and the number of constraints by m.

Let Γ be a set of relations over some finite domain D, and let CSP(Γ) denote the set of CSP instances with all constraint relations lying in Γ. In this context, Γ is known as the *constraint language*. Throughout this paper, we will assume that Γ is always finite. Then, the *decision problem* for CSP(Γ) is the problem of deciding whether a satisfying assignment exists for an instance $I \in$ CSP(Γ). The *search problem* for CSP(Γ) is the problem of deciding whether a satisfying assignment exists and, if it does, to find one such assignment.

The Distributed Model We consider the DCSP model of [27] with some small modifications. The basic idea is to assign the task of solving a constraint satisfaction problem to a multi-agent system. In the original model, which assumes that all constraints are binary [28, 29], the assumption is that each variable is controlled by an agent, and two agents can communicate with one another if and only if they share a constraint. Here we deviate slightly from the original model to allow for non-binary constraints and we assume that both variables and constraints are controlled by distributed agents in the network. An instance [12] of the *Distributed Constraint Satisfaction Problem* (DCSP) is a tuple (A, X, D, C, α), where X, D, and C are as in the classical CSP, A is a finite set of agents, and $\alpha : X \cup C \to A$ is a surjective function which assigns the control of each variable $x \in X$ and each constraint $c \in C$ to an agent $\alpha(x)$, $\alpha(c)$ respectively. For the purpose of this paper, we assume that there are exactly $n + m$ agents, and therefore each agent controls exactly one variable or one constraint. Under this assumption, there is a one-to-one correspondence between instances of CSP and DCSP, and thus we shall switch freely between them, while maintaining the distinction between agents and their controlled variables and constraints for clarity when discussing distributed algorithms.

Distributed Networks and Message Passing We now present some fundamental concepts relating to the message-passing paradigm for distributed networks. For a general introduction to distributed algorithms, we refer the reader to [13]. A *distributed system* consists of a finite set of agents or processes, which are connected through communication channels to form a network. Any process in the network can perform events of three kinds: *send*, *receive* and *internal*. Send and receive events are self-explanatory, as they denote the sending or receiving of a message over a communication channel. Any kind of local computation performed at the process level, as well as state changes and decisions, are classified as internal events.

We assume a fully *synchronous* communication model, meaning that the send event at a process a and the corresponding receive event at a process a' can be considered *de facto* as a unique event, with no time delay. As a whole, a synchronous system proceeds in rounds, where at each round a process can perform some internal computation and then send messages to and receive messages from its neighbours. A round needs to terminate at every process before the next round begins. Note that while for simplicity we assume a synchronous network, all our algorithms can be adapted to asynchronous systems by applying a simple synchronizer. Nonetheless, we point out that our negative results rely on the network operating in synchronous rounds.

We make the fundamental assumption that the network is *anonymous*, meaning that variables, constraints and agents do not have IDs. For practical purposes, we still refer to variables and constraints with names (such as x_i, c_i), however these cannot be communicated through the channels. The assumption of anonymity can have various practical justifications: the processes may actually lack the hardware to have an ID, or they may be unable to reveal their ID due to security or privacy concerns. For instance, the basic architecture of GNNs is anonymous. This is a very desirable property as it allows to deploy GNNs in different networks than those in which they were trained.

We assume that all the processes run locally the same deterministic algorithm, therefore IDs cannot be created and deadlocks cannot be broken by for instance flipping a random coin. Hence, the lack of IDs makes the processes essentially indistinguishable from one another - except, as we will see later, for the structure of their neighbourhood in the network.

Leader election is a procedure by which the processes in a network select a single process to be the leader in a distributed way. If a leader is elected, then she can, for instance, dictate the output to the other processes. Moreover, all the information about the instance can be gathered to the leader, which if the network had unique IDs, would be sufficient to solve the CSP locally at the leader. It is a well-known result that there does not exist a terminating deterministic algorithm to elect a leader in an anonymous ring [1]. Therefore, the assumptions of anonymity and determinism ensure that the DCSP model is intrinsically different from the (centralised) CSP framework, and open up the way for establishing novel, non-trivial complexity results. We remark that while considerable effort has been put into characterizing under what conditions an anonymous network is able to elect a leader [8, 26] or compute relations [9], our work focuses on characterizing the complexity of the DCSP as parametrised by the constraint language. Therefore, all of our algorithms work regardless of the topology of the network, and hence regardless of whether or not a leader can be elected.

The encoding of a DCSP instance into the message passing framework is straightforward. The processes correspond to the agents of the network, and there is a labelled communication channel between a variable agent $\alpha(x)$ and a constraint agent $\alpha(c)$ if and only if $x \in c$. More formally, the *Factor Graph* [12] G_I of an instance $I = (X, D, C)$ of CSP is the undirected bipartite graph with vertex set $X \cup C$ and edge set $\{\{x, c\} \mid x \in c\}$. Each edge in G_I that is incident to a variable x and a constraint c where $c = (\mathbf{s}, R)$ has a label $\ell_{\{x,c\}} = (S, R)$ for $S = \{i \mid \mathbf{s}[i] = x\}$, where for a tuple

$\mathbf{t}$, $\mathbf{t}[i]$ denotes the ith entry of $\mathbf{t}$.[1] Then, the message passing network corresponds to the factor graph where every node (variable or constraint) is replaced by their associated agent and every edge by a communication channel of the same label. Note that between any two agents there is at most one channel. If privacy is a concern, we point out that labeling channels does not reveal any more information about the processes than what is strictly necessary for the problem instance to be well defined. It is easy to prove (see Appendix B) that in the case that all relations are binary, the original model where only variables are controlled by agents is equivalent to our model.

We say that I is *connected* if G_I is connected in the standard graph-theoretic sense. Unless explicitly stated we only consider instances whose factor graph consists of a unique connected component.

At the start of an algorithm, a process only has access to very limited information. All processes know the total number n of variables in the CSP instance, the total number m of constraints, the labels of the communication channels that they are incident to in the network, and naturally whether they are controlling a variable or a constraint. During a run of the algorithm a process can acquire further knowledge from the messages that it receives from its neighbours. We assume that at any time each process is in one of a set of states, a subset of which are terminating states. When it enters a terminating state, a process performs no more send or internal events, and all receive events are disregarded. The local algorithm is then a deterministic function which determines the process's next state, and the messages it will send to its neighbours. The output of such function only depends on the process's current knowledge (including the messages it has received so far), and on its state. We allow processes to send different messages through different channels. However, since processes can only distinguish the channels based on their labels, identical messages must be sent through channels with identical labels. Note that the power of the model would not decrease if only one message was allowed to be passed through all the channels, since a process can simulate sending a separate message through each channel by tagging each message with the label of the desired channel and concatenating them in a unique string. This, however, comes at the cost of increased message size. Moreover, if a process needs to broadcast multiple messages, these can be concatenated into one. We say that an algorithm terminates when all processes are in a terminating state. For a precise formalisation of the definition of distributed algorithm, see Appendix A.

We say that a distributed algorithm solves an instance I of DCSP if the algorithm terminates and at termination every process correctly outputs that I is satisfiable if it is, and that it is not satisfiable otherwise. Moreover, we consider the search version of DCSP, denoted DCSP-Search. In the search version, if the input instance I is satisfiable, at termination every variable process $\alpha(x)$ must additionally specify a value $\nu(x) \in D$ such that $\nu : X \to D$ is a satisfying assignment. For every constraint language Γ, we denote by DCSP(Γ) and DCSP-Search(Γ) the restrictions of DCSP and DCSP-Search, respectively, to instances containing only constraint relations from Γ.

[1]For mathematical clarity, we label edges with the relation itself. However, in algorithmic applications, every relation can be substituted with a corresponding symbol.

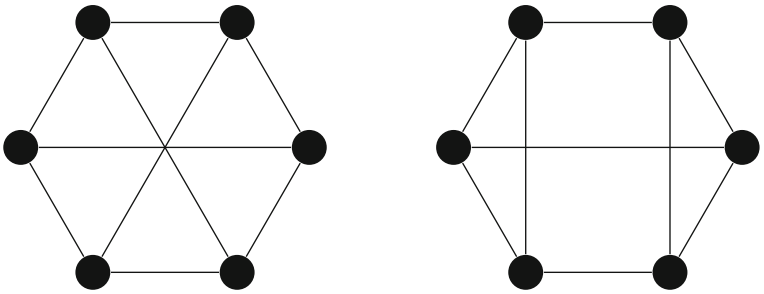

Fig. 1 Both graphs depicted above are 3-regular and hence they have the same iterated degree sequence. However, they are clearly not isomorphic, since the left graph is bipartite while the right one is not

In terms of algorithmic complexity, there are a number of measures that can be of interest. Time complexity, which is our primary concern, corresponds to the total amount of time required for the algorithm to terminate, including the time needed for internal events. This is closely related to the number of rounds of the algorithm, which is another measure that we are concerned with. Message complexity and bit complexity measure the total number of messages and bits exchanged respectively. These can be bounded easily from the maximum size of a message.

Iterated Degree and Degree Sequence We present a number of concepts from graph theory that carry over to CSPs. Their adaptation to DCSPs is straightforward in all cases. In an undirected graph G, the degree of a vertex v is the number of edges incident at v. The zeroth iterated degree of v is equal to its degree. For $k \geq 1$, the kth iterated degree of v is the multiset of $(k-1)^{th}$ degrees of v's neighbours in G. The kth iterated degree sequence of a graph is the multiset of kth iterated degrees of its vertices.

Example 1 In the context of graph theory the *colour refinement algorithm*, which calculates the iterated degree sequence of a graph, is often used as a simple heuristic for the graph isomorphism problem. If two graphs are isomorphic then they must have the same iterated degree sequence, but the opposite is not true (see for example Fig. 1).

We extend the notion of iterated degree to CSPs as follows. Consider the labelled factor graph G_I of an instance I described in the previous paragraph. In what follows it will be convenient to allow instances I with a disconnected factor graph G_I. Let v be a node of G_I and denote its neighbourhood in the factor graph by $N(v)$. The (zeroth) *degree*, denoted $\delta_0(v)$, of a node in the factor graph is simply a symbol that distinguishes variables from constraints: we set $\delta_0(x) =$ '●' for all $x \in X$ and $\delta_0(c) =$ '▲' for all $c \in C$. The kth *iterated degree*[2] ($k \geq 1$) of a node v is defined as $\delta_k(v) = \{\{(\ell_{\{v,w\}}, \delta_{k-1}(w)) \mid w \in N(v)\}\}$, where the notation $\{\{\ldots\}\}$ stands for

[2]We remark that the notions of degree and iterated degree are well-defined concepts in graph theory. We borrow this terminology to refer to the analogous concepts in CSPs.

multiset. We write $v \sim_\delta^k v'$ if $\delta_k(v) = \delta_k(v')$, and simply $v \sim_\delta v'$ if $v \sim_\delta^k v'$ for all $k \geq 0$. In this case, we say that v and v' are *iterated degree equivalent*. It can be shown (see the following proposition) that as k increases, the partition induced by $\sim_\delta^k$ gets more refined, and indeed it reaches a fixed point for some $k \leq 2n$ where $n = |X|$.

Proposition 1 *Let $I = (A, X, D, C, \alpha)$ be an instance of* DCSP(Γ) *and let $v, v' \in X \cup C$. Let $k \geq 2n$ where $n = |X|$. Then, $v \sim_\delta^k v'$ implies $v \sim_\delta v'$.*

Proof We start by showing that for all non-negative integers k, k' with $k \leq k'$, the partition induced by $\sim_\delta^{k'}$ on $X \cup C$ is at least as refined as the partition induced by $\sim_\delta^k$. The proof goes by induction. Let v, v' be arbitrary nodes in G_I. Clearly if $\delta_0(v) \neq \delta_0(v')$, then $\delta_k(v) \neq \delta_k(v')$ for all $k \in \mathbb{N}$, so in particular $\delta_1(v) \neq \delta_1(v')$. Now assume that for all $v, v' \in X \cup C$, $\delta_k(v) = \delta_k(v')$ implies $\delta_{k-1}(v) = \delta_{k-1}(v')$. Then it is a clear consequence of the definition of δ_k that $\delta_{k+1}(v) = \delta_{k+1}(v') \implies \delta_k(v) = \delta_k(v')$ for all $v, v' \in X \cup C$ too as required.

Now it remains to show that if $v \sim_\delta^{2n} v'$, then $v \sim_\delta^k v'$ for all $k \geq 2n$. The result is immediate if we replace $2n$ by $n + m$. To achieve $2n$ we use the fact that the factor graph is bipartite. Denote by P^k and Q^k the partitions induced by $\sim_\delta^k$ on X and C respectively and note that if $P^{k-2} = P^k$ then (P^k, Q^k) is a fixed point. We notice that $P^{k-2} = P^k$ must occur for some $k \leq 2n$ and we are done. □

The notion of iterated degree is strikingly relevant in our work as it captures what it means for two processes in a network to be indistinguishable. This implies that no distributed algorithm can differentiate between two iterated degree equivalent nodes, as we illustrate in the following result.

Proposition 2 *Let $I = (A, X, D, C, \alpha)$ be an instance of* DCSP(Γ) *whose factor graph is not necessarily connected and consider two nodes v, v' in G_I. Then, $v \sim_\delta v'$ if and only if any terminating decision algorithm over I outputs the same decision at $\alpha(v)$ and $\alpha(v')$. Furthermore, if $v, v' \in X$ and I is satisfiable, then any terminating search algorithm outputs the same values $\nu(v) = \nu(v')$ at $\alpha(v)$ and $\alpha(v')$.*

Proof ($\Rightarrow$). At the beginning of the algorithm, all processes are in the same state. Let v be a node in the factor graph of I, and denote by s_t^v the message broadcast at time t by $\alpha(v)$ to its neighbours. For any two nodes v, v', $\delta_1(v) = \delta_1(v')$ is equivalent to v and v' having the same initial knowledge – that is, v and v' are either both variables or both constraints and their adjacent channels have the same multiset of labels. This means that the first internal and send events are the same at $\alpha(v)$ and at $\alpha(v')$, hence $s_1^v = s_1^{v'}$. Then, it is easy to see by induction that $\delta_t(v) = \delta_t(v') \implies s_t^v = s_t^{v'}$, which in turn implies that

$$v \sim_\delta v' \implies s_t^v = s_t^{v'} \quad \text{at all times } t = 1, 2, \ldots$$

This implies that at any time t, $\alpha(v)$ and $\alpha(v')$ send and receive the same messages, so they have the same knowledge and hence the internal events at $\alpha(v)$ and $\alpha(v')$ are the same at all time. In particular, if the algorithm terminates, then the terminating

state is the same at $\alpha(v)$ and $\alpha(v')$, and therefore the decision and, in case of search, the value of ν at $\alpha(v)$ and $\alpha(v')$ are the same.

($\Leftarrow$). Consider the algorithm that calculates the iterated degree at each node (we detail the procedure in the proof of Theorem 15). If $v \not\sim_\delta v'$, then we can find an algorithm that on the basis of the iterated degree gives different outputs at $\alpha(v)$ and $\alpha(v')$. □

Example 2 Let $\Gamma = \{\neq_{\mathbb{B}}\}$ where $\neq_{\mathbb{B}}$ is the inequality relation on the Boolean domain $\mathbb{B} = \{0, 1\}$, that is, $\neq_{\mathbb{B}} = \{(0, 1), (1, 0)\}$. Let I be an instance of DCSP(Γ) with variables $\{x_1, \ldots, x_n\}$ and constraints $((x_1, x_2), \neq_{\mathbb{B}}), ((x_2, x_3), \neq_{\mathbb{B}}), \ldots, ((x_n, x_1), \neq_{\mathbb{B}})$. That is, I can be seen as a directed ring. If n is even, then clearly I is satisfiable. Therefore, if there was a distributed algorithm that solved the search version of DCSP(Γ), any two agents controlling two consecutive variables x_i, x_{i+1} should output at termination a partial assignment ν for their controlled variable such that $\nu(x_i) \neq \nu(x_{i+1})$. But it is easy to see that all variable agents in I have the same iterated degree, and so by Proposition 2, such an algorithm cannot exist.

The following is a direct consequence of Proposition 2. We say that two instances I and I' have the same *iterated degree sequence* if there exists a bijection γ between the nodes of G_I and the nodes of $G_{I'}$ such that for every $k \geq 0$ and every node v of G_I, the kth degree of v in I is equal to the kth degree of $\gamma(v)$ in I'. We note that in this case, if we construct the (disconnected) instance $I \cup I'$ containing all the variables and constraints in I and I', then $v \sim_\delta \gamma(v)$ for every node v of G_I. Hence the result below follows.

Corollary 3 *Let $I, I' \in$ DCSP(Γ) have the same iterated degree sequence. Then with both inputs any terminating decision algorithm will report the same decision.*

Example 3 Let $\Gamma = \{\neq_{\mathbb{B}}\}$ be as in Example 2 and consider the two instances pictured in Fig. 1 (where nodes represent variables and edges represent constraints). By Corollary 3, any terminating decision algorithm will report the same decision on both instances. However, one of the instances is satisfiable and the other is not, so there cannot be a distributed algorithm that solves DCSP(Γ).

Polymorphisms Let R be a k-ary relation over a finite domain D. An ℓ-ary *polymorphism* of R is an operation $f : D^\ell \to D$ such that the coordinate-wise application of f to any set of ℓ tuples from R gives a tuple in R. More precisely, for any $\mathbf{t}_1, \ldots, \mathbf{t}_\ell \in R$, we have that $(f(\mathbf{t}_1[1], \ldots, \mathbf{t}_\ell[1]), \ldots, f(\mathbf{t}_1[k], \ldots, \mathbf{t}_\ell[k])) \in R$. We say that a function f is a polymorphism of a constraint language Γ if f is a polymorphism of all relations $R \in \Gamma$. Equivalently, we say that Γ is *invariant* under f. The set of polymorphisms of a constraint language Γ will be denoted by $\mathsf{Pol}(\Gamma)$. There is a particular construction of a CSP instance that is closely related to the clone of polymorphisms of the corresponding constraint language. Let Γ be a constraint language over a finite domain D. For any positive integer r, the *indicator problem of order r* for Γ is the instance $I = (X, D, C) \in$ CSP(Γ) where $X = D^r$ and C contains for every relation $R \in \Gamma$ and for every $\mathbf{t}_1, \ldots, \mathbf{t}_r \in R$, the constraint $(\mathbf{s}, R)$

where $\mathbf{s}[i] = (\mathbf{t}_1[i], \ldots, \mathbf{t}_r[i])$ for every $i \in \{1, \ldots, arity(R)\}$. It follows easily that for every $\nu : D^r \to D$, ν satisfies I if and only if ν is a polymorphism of Γ.

An ℓ-ary operation f is said to be *symmetric* if for all $x_1, \ldots, x_\ell$ and for all permutations σ of $\{1, \ldots, \ell\}$ we have that $f(x_1, \ldots, x_\ell) = f(x_{\sigma(1)}, \ldots, x_{\sigma(\ell)})$.

Example 4 Once again let $\neq_{\mathbb{B}}$ be as in Example 2. It is easy to see that the ternary minority operation f given by $f(x, y, z) = x \oplus y \oplus z$ is a polymorphism of $\neq_{\mathbb{B}}$. On the other hand, one can show that $\neq_{\mathbb{B}}$ does not have symmetric polymorphisms of arity 2. In particular, let $\mathbf{t}_1 = (0, 1)$ and $\mathbf{t}_2 = (1, 0)$. Since a symmetric binary operation f needs to satisfy $f(0, 1) = f(1, 0)$, the coordinate-wise application of f to $\mathbf{t}_1, \mathbf{t}_2$ would yield a reflexive tuple, which cannot possibly belong to $\neq_{\mathbb{B}}$.

Our work unveils a novel structure in the space of solutions of a CSP instance that is deeply connected to the symmetry of its polymorphisms. In particular, $\mathsf{Pol}(\Gamma)$ containing symmetric polymorphisms of all arities is equivalent to the existence of a satisfying assignment to every satisfiable instance of CSP(Γ) that preserves the partition induced by $\sim_\delta$. This is the main result of Section 3.

pp and efpp definability We conclude this section by presenting a notion of definability for relations that is closely associated to polymorphisms. A relation $R \subseteq D^k$ is said to be *primitive positive definable*, most commonly shortened to *pp-definable*, from a constraint language Γ over the same domain D if there exists a pair $((x_1, \ldots, x_k), I)$ with $I = (X, D, C)$ an instance of CSP($\Gamma \cup \{eq_D\}$) where $eq_D = \{(d, d) \mid d \in D\}$ is the equality relation, and $x_1, \ldots, x_k$ are distinct variables in X such that for every tuple $\mathbf{t} \in D^k$

$$\mathbf{t} \in R \iff \text{there is a solution } \nu \text{ of } I \text{ such that } \mathbf{t}[i] = \nu(x_i) \text{ for all } 1 \leq i \leq k.$$

A constraint language Γ' is pp-definable from Γ if all the relations in Γ' can be pp-defined from Γ. The following complexity reduction between CSP classes is well known.

Theorem 4 (see [4]) *Let Γ, Γ' be finite constraint languages. If Γ pp-defines Γ', then* CSP(Γ') *is log-space reducible to* CSP(Γ).

However, in the distributed setting, allowing equality introduces a few technical difficulties. Fortunately, this obstacle could be overcome by considering a more restricted notion of pp-definability which, following [17], we shall call efpp-definability, where equality is not allowed. More precisely, we shall say that a relation R is *equality-free primitive positive definable* (efpp-definable, for short) from Γ if it is pp-definable and, in addition, the instance I witnessing the pp-definition belongs to CSP(Γ). That is, we are not allowed to use the equality relation in the instance, unless, of course, it belongs already to Γ. Then, we have:

Proposition 5 *Let Γ, Γ' be finite constraint languages, and assume that Γ' is efpp-definable from Γ. If* DCSP(Γ) *is solvable in polynomial time (resp. finite time) then so is* DCSP(Γ').

Proof Given an algorithm Alg that solves DCSP(Γ) we can design a new algorithm Alg$'$ for DCSP(Γ') that given an instance $I' = (A', X', D, C', \alpha')$ of DCSP(Γ') simulates the execution of Alg with the instance $I = (A, X, D, C, \alpha)$ of DCSP(Γ) defined as follows. For every constraint $c = ((s_1, \ldots, s_k), R)$ in C', consider the pair $((x_1, \ldots, x_k), I_R)$ defining R over Γ and replace constraint c by the instance I_c (meaning all its variables and constraints) obtained from I_R by renaming the variables so that $x_i = s_i$ for every $1 \leq i \leq k$ and the rest of variables in I_c are fresh. Note that I_R (and hence I_c) does not contain equalities. Then α is defined such that it agrees with α' over X' and, as usual, every variable and constraint in I is controlled by a different agent.

The simulation is as follows. At each round, for every $x \in X'$, $\alpha'(x)$ simulates the execution of $\alpha(x)$ as in Alg, and for every $c \in C'$, $\alpha'(c)$ simulates the execution of all constraints and fresh variables in I_c. We note that no new communication channels need to be created as this simulation is done internally by $\alpha'(c)$. The transmission of messages can be also easily simulated for every pair of neighbours $\alpha(x)$ and $\alpha(c)$ in I. In fact, if $x \notin X'$, then both $\alpha(x)$ and $\alpha(c)$ are simulated by the same agent $\alpha'(c)$ in I' (and, hence, no communication is required). Otherwise, if $x \in X'$, $\alpha(x)$ is simulated by $\alpha'(x)$ and $\alpha(c)$ is simulated by some neighbour $\alpha'(c')$ of $\alpha(x)$. □

Remark 6 We note here that for every $r \geq 1$, the indicator problem of order r for Γ constitutes an efpp-definition of the $|D|^r$-ary relation U encoding the set of all polymorphisms of arity r. It then follows from Proposition 5 that if DCSP(Γ) is solvable in finite time then so is DCSP($\{U\}$).

3 Basic Linear Programming Relaxation

For any CSP instance $I = (X, D, C)$ there is a LP relaxation (usually called *basic LP relaxation*, see for example [20]) denoted BLP(I), which is defined as follows. It has a variable $v(x, d)$ for each $x \in X$ and $d \in D$, and a variable $v(c, \mathbf{t})$ for each $c \in C$ and $\mathbf{t} \in R$ where R is the constraint relation of c. All variables must take values in the range [0, 1]. The value of $v(x, d)$ is interpreted as the probability that v is assigned to d. Similarly, the value of $v(c, \mathbf{t})$ is interpreted as the probability that the scope of c is assigned component-wise to the tuple $\mathbf{t}$. In this paper we only deal with a feasibility problem (that is, there is no objective function). The variables are restricted by the following equations:

$$\sum_{d \in D} v(x, d) = 1 \quad \text{for all } x \in X \tag{1}$$

$$\sum_{\substack{\mathbf{t} \in R_c \\ \mathbf{t}[i]=d}} v(c, \mathbf{t}) - v(\mathbf{s}_c[i], d) = 0 \quad \text{for all } c \in C, \text{ all } i \in \{1, \ldots, arity(c)\}, \text{ and all } d \in D \tag{2}$$

where we denote the relation and scope of a constraint c by R_c and $\mathbf{s}_c$ respectively. We say that BLP decides CSP(Γ) if for every instance $I \in$ CSP(Γ), I is satisfiable whenever BLP(I) is feasible. We will use the following well-known result.

Theorem 7 (see [20]) *If Γ has symmetric polymorphisms of all arities, then BLP decides* CSP(Γ). *Moreover, if $I \in$* CSP(Γ) *is satisfiable then it has a solution ν such that for all x, x' with $v(x, d) = v(x', d)$ for all $d \in D$, we have $\nu(x) = \nu(x')$.*

Proof It is clear that if there is no feasible solution to BLP(I), then I is not satisfiable, regardless of the polymorphisms of Γ. So let Γ be such that $\mathsf{Pol}(\Gamma)$ contains symmetric polymorphisms of all arities and let I be an instance of CSP(Γ) such that BLP(I) is feasible. We can assume that there exists some natural number n such that in the feasible solution to BLP(I), all variables take rational values of the form $\frac{r}{n}$ for some integer r. That is, for all $x \in X$, $d \in D$, $c \in C$ and $\mathbf{t} \in R_c$ (where R_c is the relation of c) there exist corresponding integers such that

$$v(x, d) = \frac{r(x, d)}{n} \quad \text{and} \quad v(c, \mathbf{t}) = \frac{r(c, \mathbf{t})}{n}.$$

Let f be a symmetric polymorphism of Γ of arity n. For every $x \in X$ we shall denote by f_x the value of f when applied to an n-tuple where each $d \in D$ appears exactly $r(x, d)$ times. We claim that the assignment $\nu : X \to D$ given by $\nu(x) = f_x$ satisfies I. To see this, consider an arbitrary constraint $c = (\mathbf{s}, R)$. Denote by $\mathbf{t}'$ the tuple obtained by applying f coordinate-wise to n tuples $\mathbf{t}_1, \ldots, \mathbf{t}_n$ chosen as follows: each tuple $\mathbf{t} \in R$ is chosen exactly $r(c, \mathbf{t})$ times. Clearly $\mathbf{t}' \in R$ since f is a polymorphism of R. So, to show that c is satisfied by ν, it is enough to show that $\nu(\mathbf{s}[i]) = \mathbf{t}'[i]$. Now, $\mathbf{t}'[i]$ is the result of applying f to the set of the ith elements of $\mathbf{t}_1, \ldots, \mathbf{t}_n$. But any $d \in D$ occurs in $\mathbf{t}_1[i], \ldots, \mathbf{t}_n[i]$ exactly $\sum_{\mathbf{t}[i]=d} r(c, \mathbf{t}) = n \cdot v(\mathbf{s}[i], d) = r(\mathbf{s}[i], d)$ times, and so, given that f is symmetric, we have

$$\mathbf{t}'[i] = f_{\mathbf{s}[i]} = \nu(\mathbf{s}[i])$$

as required. □

The following theorem reveals a useful structure inside the solutions of the BLP.

Theorem 8 *Let $I = (X, D, C)$ be an instance of* CSP(Γ) *such that* BLP(I) *is feasible. Then,* BLP(I) *has a feasible solution such that for every $x, x' \in X$ with $x \sim_\delta x'$ and every $d \in D$, $v(x, d) = v(x', d)$.*

Proof We start by rewriting the program in the form

$$\exists ?\mathbf{v} \in [0, 1]^V \quad B\mathbf{v} \geq \mathbf{b}. \tag{3}$$

by replacing every equality $a = b$ by the inequalities $a \geq b$ and $-a \geq -b$.

It will be convenient to index the rows and columns of B not using positive integers. Let us start with the columns. Each column is associated to a variable of BLP(I), i.e, a variable of the form $v(x, d)$, $x \in X$, $d \in D$ or $v(c, \mathbf{t})$, $c \in C$, $\mathbf{t} \in R_c$.

In the first case, we index the corresponding column with the pair (x, d) whereas in the second case we index it with the pair $(c, \mathbf{t})$, and we denote by V the set of all such indices.

Now, let us turn our attention to the rows. Every equation in (1) gives rise to two rows that we shall index with $(x, +)$ and $(x, -)$. Similarly, every equation in (2) also gives rise to two rows that we shall index with $(c, i, d, +)$ and $(c, i, d, -)$. Let us denote by W the set of all indexes for rows.

We shall see later how to define an oracle which, given a probability W-vector $\mathbf{p}$ (i.e, a vector $\mathbf{p}$ with non-negative entries such that the sum of all its entries is 1), outputs a vector $\mathbf{v}$ which is a solution to the weaker problem

$$\exists ?\mathbf{v} \in [0, 1]^V \quad \mathbf{p}^T B\mathbf{v} \geq \mathbf{p}^T \mathbf{b} \tag{4}$$

if one exists, or correctly states that no such vectors exist otherwise. Note that if a solution exists to (3), then it is necessarily also a solution to (4), while the opposite is not true in general.

For every $w \in W$, let us denote by B_w the row corresponding to w. If $w = (x, +)$ then, since the vector returned by the oracle satisfies $\mathbf{v} \in [0, 1]^V$ it follows easily that $B_w\mathbf{v} - \mathbf{b}[w] \in [-1, |D|]$. Similarly, if $w = (c, i, d, +)$ then $B_w\mathbf{v} - \mathbf{b}[w] \in [-1, \max_{R\in\Gamma} |R|]$. It follows that by setting $\ell = 1$ and $\rho = \max\{|D|, \max_{R\in\Gamma} |R|\}$ any such oracle-given vector $\mathbf{v}$ satisfies the following condition: there is a fixed subset $J \subseteq W$ (consisting precisely of the positive rows) such that

$$\begin{aligned} B_w\mathbf{v} - \mathbf{b}[w] &\in [-\ell, \rho] \quad \forall w \in J, \\ B_w\mathbf{v} - \mathbf{b}[w] &\in [-\rho, \ell] \quad \forall w \notin J. \end{aligned}$$

□

Such an oracle is known as a (ℓ, ρ)-*bounded oracle*. Then we have:

Theorem 9 ([2]) *Let $\varepsilon > 0$ be an arbitrary error parameter. Suppose that there exists an (ℓ, ρ)-bounded oracle for the feasibility problem (4). Assume that $\ell \geq \frac{\varepsilon}{2}$. Then there exists an algorithm which either finds $\mathbf{v}$ such that $B\mathbf{v} \geq \mathbf{b} - \varepsilon$ whenever such $\mathbf{v}$ exists, or correctly concludes that no such $\mathbf{v}$ exists otherwise. Such algorithm makes $\mathcal{O}(\ell\rho \log(|W|)/\varepsilon^2)$ calls to the oracle.*

The algorithm that Theorem 9 refers to is Multiplicative Weight Update (MWU), a well-known technique that is widely used in optimisation and machine learning. MWU was discovered independently by researchers of different communities; for a survey of its different variants we refer the reader to [2]. The version that is relevant to our paper is described in Algorithm 1. Recall that the algorithm assumes that there is a feasible solution.

Algorithm 1 Multiplicative weight update.

Initialisation: Fix $\eta \in [0, \frac{1}{2}]$ and let $\mathbf{w}^{(1)}$ be a W-vector, whose entries, called weights, are initially set to 1.
for $t = 1, \ldots, T$ **do**
  Compute the probability vector $\mathbf{p}^{(t)} = \frac{1}{\Phi(t)}\mathbf{w}^{(t)}$, where $\Phi(t) = \sum_{j=1}^{|W|} \mathbf{w}^{(t)}[j]$
  Let $\mathbf{v}^{(t)}$ be a solution satisfying $(\mathbf{p}^{(t)})^T B\mathbf{v}^{(t)} \geq (\mathbf{p}^{(t)})^T \mathbf{b}$ given by oracle O
  Compute the losses $\boldsymbol{\ell}^{(t)} = \frac{1}{\rho}(B\mathbf{v}^{(t)} - \mathbf{b})$
  Compute the new weights $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} \odot (1 - \eta\boldsymbol{\ell}^{(t)})$ (where $\odot$ denotes the element-wise product)
end
return $\mathbf{v} := \frac{1}{T}\sum_{t=1}^{T} \mathbf{v}^{(t)}$

We shall see that if we choose the oracle O wisely then for every $x, x' \in X$ with $x \sim_\delta x'$ and every $d \in D$, the solution returned by the MWU algorithm assigns the same value to $v(x, d)$ and $v(x', d)$.

To see this we need some more notation. We note that $\sim_\delta$ induces in a natural way an equivalence relation $\sim_V$ on V. In particular, we have that $v, v' \in V$ are $\sim_V$-related if $v = (x, d)$ and $v' = (x', d)$ where $x \sim_\delta x'$ and $d \in D$, or $v = (c, \mathbf{t})$ and $v' = (c', \mathbf{t})$ where $c \sim_\delta c'$ and $\mathbf{t} \in R_c$ (note that, necessarily, $R_c = R_{c'}$). Similarly $\sim_\delta$ induces an equivalence relation, denoted $\sim_W$, on W. More specifically, we have that $w, w' \in W$ are $\sim_W$-related if $w = (x, s)$ and $w' = (x', s)$ where $x \sim_\delta x'$ and $s \in \{+, -\}$ or $w = (c, i, d, s)$ and $w' = (c', i, d, s)$ where $c \sim_\delta c'$, $i \in \{1, \ldots, arity(c)\}$, $d \in D$, and $s \in \{+, -\}$.

Now, we say that a V-vector $\mathbf{v}$ is $\sim_V$*-preserving* if $\mathbf{v}[v] = \mathbf{v}[v']$ whenever $v \sim_V v'$ and we similarly define $\sim_W$-preserving W-vectors. So it is enough to show that there exists some oracle O that guarantees that at each iteration t of the WMU algorithm, $\mathbf{v}^{(t)}$ is $\sim_V$-preserving. To this end we need the following easy properties.

Claim 1 For all $\sim_V$-preserving V-vectors $\mathbf{v}$ and for all $\sim_W$-preserving W-vectors $\mathbf{w}$, we have that

1. $B\mathbf{v}$ is $\sim_W$-preserving;
2. $\mathbf{w}^T B$ is $\sim_V$-preserving.

Proof of Claim 1. We include only the proof of (2) as the proof of (1) is analogous and, indeed, simpler. Let $\mathbf{v} := \mathbf{w}^T B$. An easy computation shows that

$$\mathbf{v}(x, d) = \mathbf{w}(x, +) - \mathbf{w}(x, -) - \sum_{c \in C_x} \sum_{\substack{1 \leq i \leq arity(c) \\ \mathbf{s}_c[i] = x}} \big(\mathbf{w}(c, i, d, +) - \mathbf{w}(c, i, d, -)\big)$$

where we write C_x to denote the set of all constraints in C where x appears in the scope, and

$$\mathbf{v}(c, \mathbf{t}) = \sum_{\substack{1 \le i \le arity(c) \\ \mathbf{t}[i]=d}} \big(\mathbf{w}(c, i, d, +) - \mathbf{w}(c, i, d, -)\big)$$

It is immediate to see that, if $\mathbf{w}$ is $\sim_W$-preserving, then $\mathbf{v}(c, \mathbf{t}) = \mathbf{v}(c', \mathbf{t})$ whenever $c \sim_\delta c'$. Let us show that $\mathbf{v}(x, d) = \mathbf{v}(x', d)$ whenever $x \sim_\delta x'$. Since $\mathbf{w}$ is $\sim_W$-preserving we have that $\mathbf{w}(x, s) = \mathbf{w}(x', s)$ for $s \in \{+, -\}$ and hence we only need to show that $\varphi_x(C_x) = \varphi_{x'}(C_{x'})$ where $\varphi_x(C_x)$ is a shorthand for

$$\sum_{c \in C_x} \sum_{\substack{1 \le i \le arity(c) \\ \mathbf{s}_c[i]=x}} \big(\mathbf{w}(c, i, d, +) - \mathbf{w}(c, i, d, -)\big)$$

and $\varphi_{x'}(C_{x'})$ is defined analogously.

Now, for every $R \in \Gamma$, every $S \subseteq \{1, \ldots, arity(R)\}$, and every class $[c]$ of equivalent constraints, let $C_{x,R,S,[c]}$ be the set of constraints in C_x that belong to $[c]$, whose constraint relation is R, and whose scope $\mathbf{s}$ satisfies the following: $(i \in S) \Leftrightarrow \mathbf{s}[i] = x$ for every $i \in \{1, \ldots, arity(R)\}$. Note that since C_x and $C_{x'}$ can be partitioned as the union of sets of this form it is only necessary to show that $\varphi_x(C_{x,R,S,[c]}) = \varphi_{x'}(C_{x',R,S,[c]})$ for every choice of R, S, and $[c]$. To see this it is enough to note that $|C_{x,R,S,[c]}| = |C_{x',R,S,[c]}|$ (because $x \sim_\delta x'$) and that, since $\mathbf{w}$ is $\sim_W$-preserving, for every constraint $c' \in [c]$ and every choice of d, i, and s, we have $\mathbf{w}(c', i, d, s) = \mathbf{w}(c, i, d, s)$. □

Proof Now, consider the oracle $\mathbf{O}$ that, given a W-vector $\mathbf{p}$, returns the V-vector $\mathbf{v}$ defined as $\mathbf{v}[v] = 1$ if $\mathbf{p}^T B[v]$ is positive and $\mathbf{v}[v] = 0$ otherwise. Since $\mathbf{v}$ maximizes $\mathbf{p}^T B\mathbf{v}$ under the restriction $\mathbf{v} \in [0, 1]^V$ it follows that $\mathbf{v}$ satisfies (4). Furthermore, it is easy to see that if $\mathbf{p}$ is $\sim_W$-preserving then $\mathbf{v}$ is $\sim_V$-preserving.

Now, note that by definition both $\mathbf{w}^{(1)}$ - which is an all-ones W-vector - and $\mathbf{b}$ are $\sim_W$-preserving. It follows easily by induction that for each t, $\mathbf{v}^{(t)}$ is $\sim_V$-preserving and $\mathbf{w}^{(t)}$ is $\sim_W$-preserving. Hence, if we call algorithm WMU iteratively with $T \to \infty$ we obtain in the limit a feasible solution satisfying the conditions of the statement. We note here that, although we have not included explicitly any inequalities requiring that all the variables in BLP(I) take values in the range [0, 1], this is guaranteed by the fact that all the entries of the vector returned by $\mathbf{O}$ are in the range [0, 1]. This concludes the proof of Theorem 8. □

We finalize the section by presenting the theorem on the structure of the solution space of CSP instances.

Theorem 10 *Let Γ be a finite constraint language. The following are equivalent:*

1. *Γ has symmetric polymorphisms of all arities.*
2. *For all satisfiable instances $I = (X, D, C) \in$ CSP(Γ) there exists a satisfying assignment $\nu : X \to D$ such that for all pairs of variables $x, x' \in X$, if $x \sim_\delta x'$ then $\nu(x) = \nu(x')$.*

Proof (1)⇒ (2). Let I be a satisfiable instance of CSP(Γ), where Γ has symmetric polymorphisms of all arities. Consider the solution of BLP(I) given by Theorem 8 and note that it satisfies $v(x, d) = v(x', d)$ for all $x \sim_\delta x'$ and all $d \in D$. Then, by Theorem 7, I has a solution ν which satisfies $\nu(x) = \nu(x')$ for all $x \sim_\delta x'$.

(2)⇒ (1). Let Γ satisfy (2) and let $r \geq 1$. We shall prove that Γ has a symmetric polymorphism of arity r. Let $I = (X, D, C)$ be the indicator problem of order r. Recall that every solution to I corresponds to an r-ary polymorphism of Γ, and hence the indicator problem is always satisfiable since for instance the projection to the first coordinate is a polymorphism of Γ. Let ν be a solution of the indicator problem which satisfies condition (2). It is easy to show by induction that for every tuple $(t_1 \ldots, t_r) \in D^r$, every permutation σ of $\{1, \ldots, r\}$ and every $k \geq 0$, $(t_1, \ldots, t_r) \sim_\delta^k (t_{\sigma(1)}, \ldots, t_{\sigma(r)})$ which implies that $\nu(t_1, \ldots, t_r) = \nu(t_{\sigma(1)}, \ldots, t_{\sigma(r)})$. We conclude that ν is symmetric as required. □

4 The Complexity of DCSP

The primary goal of this section is to prove the main theorem of this paper, namely, the dichotomy theorem for tractability of DCSP(Γ), which we now state.

Theorem 11 DCSP(Γ) *is solvable in polynomial time if and only if* Pol(Γ) *contains symmetric polymorphisms of all arities. Otherwise,* DCSP(Γ) *cannot be solved in finite time.*

We show hardness of constraint languages that do not have symmetric polymorphisms of all arities in Section 4.1 and tractability of the remaining languages in Section 4.2. In addition, in Section 4.3 we extend the decision algorithm so that, additionally, it also provides a solution to the search problem. Hence we have:

Theorem 12 DCSP-Search(Γ) *is solvable in polynomial time if and only if* Pol(Γ) *contains symmetric polymorphisms of all arities. Otherwise,* DCSP-Search(Γ) *cannot be solved in finite time.*

4.1 Intractable Languages

In this section we focus on intractable languages, that is, the hardness part of Theorem 11.

Theorem 13 *Let* Γ *be a constraint language on a finite domain* D. *If* Pol(Γ) *does not contain symmetric operations of all arities, then there is no algorithm that solves* DCSP(Γ) *in finite time.*

Schematically, the proof goes as follows. Assume that Γ does not have symmetric polymorphisms of some arity r. Consider the relation U defined by the set of solutions of the indicator problem of order r. It can be shown that if DCSP(Γ) is solvable in polynomial (or finite) time then so is DCSP($\{U\}$). Then, we show that there always

exist two connected instances of DCSP($\{U\}$), of which one is satisfiable and the other one is not, that have the same iterated degree sequence. Therefore, any algorithm will return the same output on both instances, meaning that one of these outputs is wrong. Before embarking on the proof we state the following useful combinatorial lemma.

Lemma 14 *Let $0 < k < d$ be positive integers. If n is a large enough multiple of k, then there exists a collection $\mathbb{S}$ of n^k k-element subsets of $\{0, 1, \ldots, kn-1\}$ satisfying the following properties:*

(a) *$\mathbb{S}$ contains every k-element subset of $\{0, \ldots, d-1\}$*
(b) *Every element of $\{0, 1, \ldots, kn-1\}$ appears in the same number of sets of $\mathbb{S}$.*
(c) *If $k \geq 2$, the k-uniform hypergraph with vertex set given by $\{0, 1, \ldots, kn-1\}$ and edge set given by $\mathbb{S}$ is connected.*

Proof If $k = 1$ we can just define $\mathbb{S}$ to be the set containing all singletons in $\{0, 1, \ldots, kn-1\}$ so we can assume that $k \geq 2$. Pick some n that is a multiple of k and consider the subsets of $\{0, 1, \ldots, kn-1\}$. We say that one such set is bad if $S = S + i \mod kn$ for some $i \neq 0$, and good otherwise where the right-hand side of the equation is a shorthand for the set $\{s + i \mod kn \mid s \in S\}$. The following facts hold. □

Claim 2 If $n > \frac{2(d-1)}{k}$, then all subsets of $\{0, \ldots, d-1\}$ are good.

Proof of Claim 2. Let $S \subseteq \{0, \ldots, d-1\}$ and assume that S is bad. Then, there exists i such that $S = S + i \mod kn$. Denote by s_m the smallest element of S. Then, we need $s_m + i \in S$, which implies that $i \leq d-1$. On the other hand, $kn + s_m = x + i$ for some $x \in S$, implying that $kn + s_m \leq 2(d-1) < kn$, a contradiction. □

Claim 3 There are at least n^k good sets.

Proof of Claim 3. Let S be a bad set and let $0 < i < kn$ be minimal such that $S = S + i$. We note that S is fully determined by $S \cap [0, i-1]$ and i. Now, we observe that since $0 < i$, it follows that $|S \cap [0, i-1]| \leq k/2$ and, hence, there are at most $i^{k/2} \leq (kn)^{k/2}$ choices for $S \cap [0, i-1]$. If $s \in S$ and S is bad it follows that $s + j \cdot i \mod kn \in S$ for every integer j. Since S has k elements it follows that $i \cdot j = kn$ for some $j \leq k$. Hence, we have at most k choices for i. Hence, in total we have at most $O(n^{k/2})$ bad sets. This implies that there are at least $\binom{kn}{k} - O(n^{k/2})$ good sets, which is at least n^k for n large enough.

Therefore, consider the collection of good k-element subsets of $\{0, 1, \ldots, kn-1\}$. We say that two sets S, S' are *related* if $S = S' + i \mod kn$ for some $i \neq 0$. Note that, since we are only considering good sets, every class of related sets has exactly kn members and, hence, there are at least n^k/kn many classes. Also it is immediate that every class of related sets satisfies condition (b).

Hence, to construct $\mathbb{S}$ we just need to remove some of the classes of good sets so that we end up having exactly n^k/kn classes, which corresponds to n^k sets. We have to keep all the classes containing one of the sets of condition (b), which is always possible if we pick n large enough so that $\binom{d}{k} \leq n^k$.

It only remains to show that the hypergraph with vertex set $\{0, 1, \ldots, kn-1\}$ and edge set $\mathbb{S}$ is connected. To do this it is sufficient to show that each $i \in \{0, 1, \ldots, kn-1\}$ is connected to, say, vertex 0 by a walk. This is immediate to see as, for instance, $S_0 = \{0, 1, \ldots, k-1\} \in \mathbb{S}$ (because $S_0 \subseteq \{0, 1, \ldots, d-1\}$) and therefore $S_i := S_0 + i \in \mathbb{S}$ for all $i \in \{0, 1, \ldots, kn-1\}$, hence, the sets $S_0, S_1, \ldots, S_{i-k+1}$ witness that 0 and i are connected. □

Proof of Theorem 13 Assume that $\mathsf{Pol}(\Gamma)$ does not contain symmetric polymorphisms of arity $r \geq 2$. Fix any arbitrary order $\mathbf{t}_1, \ldots, \mathbf{t}_{|D|^r}$ on the tuples of D^r and consider the relation U defined as

$$\{(f(\mathbf{t}_1), \ldots, f(\mathbf{t}_{|D|^r})) \mid f \text{ is a polymorphism of } \Gamma \text{ of arity } r\}$$

This is, U is the set of solutions of the indicator problem of order r. Given that U is efpp-definable from Γ, it follows from Proposition 5 and Remark 6 that if DCSP($\{U\}$) is not solvable in finite time then neither is DCSP(Γ).

Partition D^r into equivalence classes where two tuples $\mathbf{t}, \mathbf{t}' \in D^r$ are related, denoted $\mathbf{t} \equiv \mathbf{t}'$, if there exists some permutation σ on $\{1, \ldots, r\}$ such that $\mathbf{t}'[i] = \mathbf{t}[\sigma(i)]$ for every $i \in \{1, \ldots, r\}$. We shall use $D^r_{\equiv}$ to refer to the collection of classes and $[\mathbf{t}]_{\equiv}$ to refer to the class of tuple $\mathbf{t}$. For every $\mathbf{t} \in D^r$, define $k_{[\mathbf{t}]_{\equiv}}$ to be the number of tuples in $[\mathbf{t}]_{\equiv}$. Then we can choose an integer n large enough such that for every $\mathbf{t} \in D^r$, n is a multiple of $k_{[\mathbf{t}]_{\equiv}}$, and n satisfies Lemma 14 for $k = k_{[\mathbf{t}]_{\equiv}}$ and $d = k_{[\mathbf{t}]_{\equiv}} \cdot |D|$. Notice that since $\mathsf{Pol}(\Gamma)$ does not contain symmetric polymorphisms of all arities, we must have that $|D| \geq 2$, and hence there is some $[\mathbf{t}]_{\equiv} \in D^r_{\equiv}$ such that $k_{[\mathbf{t}]_{\equiv}} \geq 2$.

We are now ready to construct two instances I_1 and I_2 of DCSP($\{U\}$), which are indistinguishable with respect to their iterated degree sequence, but differ with regards to satisfiability. The two instances have the same set of variables, defined to be $\bigcup_{[\mathbf{t}]_{\equiv} \in D^r_{\equiv}} V_{[\mathbf{t}]_{\equiv}}$ where $V_{[\mathbf{t}]_{\equiv}} = \{v^i_{[\mathbf{t}]_{\equiv}} \mid 0 \leq i < k_{[\mathbf{t}]_{\equiv}} n\}$ is a set of $k_{[\mathbf{t}]_{\equiv}} n$ distinct variables.

We start by constructing the constraints of the unsatisfiable instance I_1, which we will do in two stages. First, for every class $[\mathbf{t}]_{\equiv}$, let $\mathbb{S}_{[\mathbf{t}]_{\equiv}}$ be the collection of $n^{k_{[\mathbf{t}]_{\equiv}}}$ sets of cardinality $k_{[\mathbf{t}]_{\equiv}}$ given by Lemma 14, as before with $d = k_{[\mathbf{t}]_{\equiv}} \cdot |D|$ and $k = k_{[\mathbf{t}]_{\equiv}}$. Note that each set in $\mathbb{S}_{[\mathbf{t}]_{\equiv}}$ defines naturally a subset of $V_{[\mathbf{t}]_{\equiv}}$ so we shall abuse notation and assume that $\mathbb{S}_{[\mathbf{t}]_{\equiv}}$ is a collection of subsets of $V_{[\mathbf{t}]_{\equiv}}$.

To simplify notation it will be convenient to use $\mathbb{S}$ as a shorthand for the indexed family $\{\mathbb{S}_{[\mathbf{t}]_{\equiv}} \mid [\mathbf{t}]_{\equiv} \in D^r_{\equiv}\}$. Now let S be $\{S_{[\mathbf{t}]_{\equiv}} \mid [\mathbf{t}]_{\equiv} \in D^r_{\equiv}\}$ satisfying $S_{[\mathbf{t}]_{\equiv}} \in \mathbb{S}_{[\mathbf{t}]_{\equiv}}$ for every $[\mathbf{t}]_{\equiv} \in D^r_{\equiv}$. We associate to S the constraint $(\mathbf{s}, U)$ where the scope $\mathbf{s}$ is constructed as follows. Before defining $\mathbf{s}$ we need some preparation. Recall that every coordinate of U, and hence of $\mathbf{s}$, is associated to a tuple $\mathbf{t} \in D^r$, so we can talk of the class $[\mathbf{t}]_{\equiv}$ to which each coordinate belongs. In particular, there are $k_{[\mathbf{t}]_{\equiv}}$ coordinates in $\mathbf{s}$ of class $[\mathbf{t}]_{\equiv}$. Hence, by fixing some arbitrary ordering we can use $\mathbf{s}^i_{[\mathbf{t}]_{\equiv}}$,

$i = 1, \ldots, k_{[\mathbf{t}]_\equiv}$ to refer to the coordinates in $\mathbf{s}$ of class $[\mathbf{t}]_\equiv$. Then, informally, $S_{[\mathbf{t}]_\equiv}$ describes which variables from $v^0_{[\mathbf{t}]_\equiv}, \ldots, v^{k_{[\mathbf{t}]_\equiv}n-1}_{[\mathbf{t}]_\equiv}$ to use in order to fill coordinates $\mathbf{s}^i_{[\mathbf{t}]_\equiv}$, $i = 1, \ldots, k_{[\mathbf{t}]_\equiv}$. Formally, for every $[\mathbf{t}]_\equiv \in D^r_\equiv$ and each $i = 1, \ldots, k_{[\mathbf{t}]_\equiv}$, $\mathbf{s}^i_{[\mathbf{t}]_\equiv}$ is assigned to the ith element in $S_{[\mathbf{t}]_\equiv}$ in increasing order.

We add such a constraint for each of the $\Pi_{[\mathbf{t}]_\equiv \in D^r_\equiv} n^{k_{[\mathbf{t}]_\equiv}} = n^{(|D|^r)}$ possible choices for S. Therefore, after the first stage we have exactly $n^{(|D|^r)}$ constraints.

In the second stage we add more constraints which will yield the particular symmetry of I_1. Note that every permutation σ on $\{1, \ldots, r\}$ induces a permutation σ' on the coordinates of U in a natural way. Specifically, if coordinate i of U is associated to tuple $\mathbf{t}_i$, then $\sigma'(i) = j$ where $\mathbf{t}_j = (\mathbf{t}_i[\sigma(1)], \ldots, \mathbf{t}_i[\sigma(r)])$. Then, in the second stage, for each permutation σ on $\{1, \ldots, r\}$ and for every constraint $(\mathbf{s}, U)$ added in the first stage we add the constraint $(\mathbf{s}', U)$ where for every $1 \leq i \leq |D|^r$, $\mathbf{s}'[i] = \mathbf{s}[\sigma'(i)]$. Therefore, after the second stage we have a total of $m = r! \cdot n^{(|D|^r)}$ constraints as needed.

We now turn to I_2. The constraints are constructed in a similar way, but instead of using the family $\mathbb{S}$ in the first stage, we use a different family $\mathbb{S}'$. In particular, for each class $[\mathbf{t}]_\equiv$, $\mathbb{S}'_{[\mathbf{t}]_\equiv}$ is obtained by partitioning $V_{[\mathbf{t}]_\equiv}$ in $k_{[\mathbf{t}]_\equiv}$ blocks of consecutive elements, so that each block has exactly n elements. Then, $\mathbb{S}'_{[\mathbf{t}]_\equiv}$ contains the $n^{k_{[\mathbf{t}]_\equiv}}$ sets that can be obtained by selecting one element from each block. The second stage is done exactly as in I_1.

We need to show that I_1 and I_2 are connected. Notice that in both instances each constraint $(\mathbf{s}, U)$ spans all subsets $V_{[\mathbf{t}]_\equiv}$, so in both cases it is sufficient to prove that the subgraph of the factor graph induced by the constraint set together with one of these subsets $V_{[\mathbf{t}]_\equiv}$ is connected. Pick any $\mathbf{t} \in D^r$ with $k_{[\mathbf{t}]_\equiv} \geq 2$. Then, from part 14 of Lemma 14, the hypergraph $(V_{[\mathbf{t}]_\equiv}, \mathbb{S}_{[\mathbf{t}]_\equiv})$ is connected, and since for every $S_{[\mathbf{t}]_\equiv} \in \mathbb{S}_{[\mathbf{t}]_\equiv}$ there is some constraint in I_1 that all the variables in $S_{[\mathbf{t}]_\equiv}$ participate in, $V_{[\mathbf{t}]_\equiv}$ satisfies the aforementioned connectedness condition in I_1, meaning that I_1 is connected. As for I_2, the reasoning is the same except that we are left to show that whenever $k_{[\mathbf{t}]_\equiv} \geq 2$, the hypergraph with vertex set $\{0, 1, \ldots, kn - 1\}$ and edge set $\mathbb{S}'$ is connected. This is immediate as by construction of $\mathbb{S}'$, any two vertices in $\{0, 1, \ldots, kn - 1\}$ either belong to two separate blocks, in which case they share an edge, or they belong to the same block, in which case they both share an edge with any other vertex in any other block. □

Claim 4 I_1 and I_2 have the same iterated degree sequence.

Proof of Claim 4. Let $[\mathbf{t}]_\equiv \in D^r_\equiv$. First, we observe that in both instances after the first stage, every variable of $V_{[\mathbf{t}]_\equiv}$ appears in the same number of constraints. More specifically, every variable in $V_{[\mathbf{t}]_\equiv}$ appears in an n-fraction of the constraints added in stage 1. In the case of instance I_1 this is due to the fact that $\mathbb{S}_{[\mathbf{t}]_\equiv}$ satisfies condition 14 in Lemma 14 and in instance I_2 this follows from the fact that $\mathbb{S}'_{[\mathbf{t}]_\equiv}$ contains all possible sets obtained by choosing an element within each one of the blocks of size n. After the second stage (in both I_1 and I_2 since the second stage is common) every variable in $V_{[\mathbf{t}]_\equiv}$ still participates in an n-fraction of the total number of constraints. In

addition, it follows easily that the positions of the scope in which a variable in $V_{[\mathbf{t}]_\equiv}$ participates distribute evenly among the $k_{[\mathbf{t}]_\equiv}$ positions associated to $\mathbf{t}$. That is, in both instances, we have that for every $[\mathbf{t}]_\equiv \in D^r_\equiv$, every variable $x \in V_{[\mathbf{t}]_\equiv}$, and every position i associated to $[\mathbf{t}]_\equiv$ there are exactly $\frac{m}{nk_{[\mathbf{t}]_\equiv}}$ constraints in which x appears at position i of the scope, where $m = r! \cdot n^{|D|^r}$. Using this fact it is very easy to prove that I_1 and I_2 have the same iterated degree sequence. Formally, one could show by induction on k that for every $[\mathbf{t}]_\equiv \in D^r_\equiv$ and $x_1, x_2 \in V_{[\mathbf{t}]_\equiv}$, $\delta_k^{I_1}(x_1) = \delta_k^{I_2}(x_2)$ and that for any two constraints c_1, c_2 in I_1 and I_2 respectively $\delta_k^{I_1}(c_1) = \delta_k^{I_2}(c_2)$. Here we are using $\delta_k^{I_1}(\cdot)$ and $\delta_k^{I_2}(\cdot)$ to denote the kth degree of a node in the factor graphs of I_1 and I_2 respectively. □

Claim 5 Instance I_1 is unsatisfiable while instance I_2 is satisfiable.

Proof of Claim 5. We start by showing that I_1 is not satisfiable. Assume by contradiction that I_1 has a satisfying assignment ν. For each class $[\mathbf{t}]_\equiv$, consider the values given by ν to the first d variables $v_0, \ldots, v_{d-1}$ in $V_{[\mathbf{t}]_\equiv}$. Since $d = k_{[\mathbf{t}]_\equiv} \cdot |D|$, it follows by the pigeon-hole principle that at least $k_{[\mathbf{t}]_\equiv}$ of these variables are assigned by ν to the same value of D. Let $S_{[\mathbf{t}]_\equiv}$ be a subset of $V_{[\mathbf{t}]_\equiv}$ containing $k_{[\mathbf{t}]_\equiv}$ of these variables (we know that this subset belongs to $\mathbb{S}_{[\mathbf{t}]_\equiv}$ by condition 14 of Lemma 14). Now consider the constraint $(\mathbf{s}, U)$ in I_1 associated to $S := \{S_{[\mathbf{t}]_\equiv} \mid [\mathbf{t}]_\equiv \in D^r_\equiv\}$, which belongs to I_1. If ν is a solution to I_1, then the restriction of ν to $\mathbf{s}$ corresponds to an r-ary polymorphism of Γ. But ν assigns the same value to any two related tuples $\mathbf{t} \equiv \mathbf{t}'$, which implies that ν is symmetric, a contradiction.

We now turn our focus to I_2. Let f be any r-ary polymorphism of Γ (for example the ith $(1 \leq i \leq r)$ projection operation defined as $f(x_1, \ldots, x_r) = x_i$). We shall construct a solution ν of I_2 in the following way. Recall that in the definition of I_2 we have partitioned the elements of $V_{[\mathbf{t}]_\equiv}$ in $k_{[\mathbf{t}]_\equiv}$ consecutive blocks. In the first stage, all the elements in each block are placed in the same coordinate of U. So, if $\mathbf{t}_1, \ldots, \mathbf{t}_{|D|^r}$ are the tuples associated to coordinates $1, \ldots, |D|^r$ and hence blocks $1, \ldots, |D|^r$ respectively, then we only need that all variables in the ith block are assigned to $f(\mathbf{t}_i)$ to satisfy all constraints added in the first stage. This assignment also satisfies the constraints added in the second stage, because if f is an r-ary polymorphism of Γ, then for every permutation σ on $\{1, \ldots, r\}$, the operation g defined as $g(x_1, \ldots, x_r) = f(x_{\sigma(1)}, \ldots, x_{\sigma(r)})$ is also a polymorphism of Γ. □

Proof To sum up, we constructed two connected instances I_1 and I_2, the latter of which is satisfiable while the former is not, which have the same iterated degree sequence. It follows from Corollary 3 that any distributed algorithm will give the same output on both instances, meaning that no algorithm can solve DCSP($\{U\}$). As anticipated at the beginning of the proof then it follows that there are also no algorithms that solve DCSP(Γ). □

4.2 Tractable Languages

In this section we turn our attention to the tractable case. In particular we shall show the following:

Theorem 15 *Let Γ be a constraint language that is invariant under symmetric polymorphisms of all arities. Then there is an algorithm* Alg *that solves* DCSP(Γ). *The total running time, number of rounds, and maximum message size of* Alg *are, respectively, $\mathcal{O}(n^3 m \log n)$, $\mathcal{O}(n^2)$, and $\mathcal{O}(m \log n)$ where n and m are the number of variables and constraints, respectively, of the input instance.*

Note that this implies the "if" part of Theorem 11. Alg is composed of two phases. In the first phase, a distributed version of the colour refinement algorithm allows every process to calculate its iterated degree. Then, thanks to Theorem 10 we can use the degree of a variable as its ID for the second phase, implying that a distributed adapted version of the jpq-consistency algorithm [18] where messages are tagged with a process's iterated degree solves the decision problem for Γ.

Distributed Colour Refinement Let $I = (A, X, D, C, \alpha)$ be an instance of DCSP(Γ) and let $n = |X|$ and $m = |C|$. There is a very natural way to calculate an agent's iterated degree in a distributed way, both for variables and for constraints. This is a mere adaptation of the 1-dimensional Weisfeiler-Leman algorithm, also known as colour refinement, an algorithm that partitions the vertices of a graph into classes by iteratively distinguishing them on the basis of their degree (see for example [15, 16]). The algorithm proceeds in rounds. At round $k = 0$, each agent $\alpha(v)$ for $v \in X \cup C$ computes $\delta_0(v)$ and broadcasts it to all its neighbours. At round $k > 0$, each agent $\alpha(v)$ knows the $(k-1)^{th}$ degrees of its neighbours which it had received in the previous round, uses them to compute $\delta_k(v)$, and broadcasts it to its neighbours. If $k = 2n$ then for every $x, x' \in X$ satisfying $x \sim_\delta^k x'$ we have that $x \sim_\delta x'$, which implies that we can essentially regard the kth iterated degree as the unique common ID for all variables that are iterated degree equivalent. Then in $2n$ rounds each agent $\alpha(v)$ can compute $\delta_\infty(v)$, where we use δ_∞ as a shorthand for δ_{2n}. As we described it, the distributed colour refinement algorithm is not particularly efficient in terms of message complexity. Although this is not necessary to achieve polynomial time, we can reduce the space required to encode $\delta_\infty(v)$.

Lemma 16 *Let $s_{\max}$ denote the size of the encoding of $\delta_\infty(v)$. A modified version of the distributed colour refinement algorithm that runs over $\mathcal{O}(n^2)$ rounds achieves $s_{\max} = \mathcal{O}(\log n)$. The time at each round and the maximum size of a message are both bounded above by $\mathcal{O}(m s_{max})$.*

Proof We describe a variation of the distributed colour refinement algorithm that achieves the required bounds. After computing the kth degree and before

proceeding to compute the $(k+1)^{th}$ degree, all agents broadcast their k^{th} degree to their neighbours. At the next round, every agent broadcasts all the kth degrees received (removing repetitions) to its neighbours so that in $2n$ rounds every agent has received a complete list of all the kth degrees of all nodes. Every agent $\alpha(v)$ orders all k^{th} degrees (this can easily be done in such a way that all agents produce the same order), and sets $\delta_k(v)$ to be the rank of its own degree in the order. Then it proceeds to send out this new encoding of $\delta_k(v)$ and to calculate $\delta_{k+1}(v)$ accordingly.

In this way, we have $s_{\max} = \mathcal{O}(\log(n+m)) = \mathcal{O}(\log n)$. Note that the total number of rounds of this algorithm is $\mathcal{O}(n^2)$ and that, provided every set of degrees is stored as an ordered array, the cost of each computation done locally by an agent at a given round is bounded above by the size, $\mathcal{O}((n+m)s_{\max}) = \mathcal{O}(ms_{\max})$, of the largest message sent. Note that in order to obtain these bounds we have used the standard facts that any instance of CSP over a fixed finite constraint language Γ is such that $m = \mathcal{O}(n^r)$ and $n = \mathcal{O}(rm)$, where r is the largest arity of a relation in Γ. □

As we will see, the price of an increase in the number of rounds (from n to n^2) is compensated by the effect of $s_{\max}$ on both time complexity and the size of the messages.

The Distributed Consistency Algorithm It is well known that if a constraint language Γ has symmetric operations of all arities then it satisfies the so-called *bounded width* property (see [5]). We avoid introducing the formal definition of bounded width as it is not needed in our results and refer the reader to [5] for reference. Informally, a constraint language Γ has bounded width if CSP(Γ) can be solved by a local consistency algorithm. This is a large class of problems which includes many languages that are not solvable by BLP, such as for instance the Boolean inequality relation $\neq_{\mathbb{B}}$ of Example 2. Now, it has been shown in [18] that if Γ has bounded width and $I \in \mathrm{CSP}(\Gamma)$ satisfies a combinatorial condition called jpq-consistency, then I has a solution. Instead of stating literally the result in [18] we shall state a weaker version that uses a different notion of consistency, more suitable to the model of distributed computation introduced in the paper.

A *set system* S is a subset of $X \times D$. We shall use S_x to denote the set $\{d \in D \mid (x,d) \in S\}$. A walk of length ℓ (in instance I) is any sequence $x_0 c_0 \ldots c_{\ell-1} x_\ell$ where $x_0, \ldots, x_\ell$ are variables, $c_0, \ldots, c_{\ell-1}$ are constraints, and $x_i, x_{i+1} \in c_i$ for every $0 \le i < \ell$. Note that walks are precisely the walks in the factor graph G_I (in the standard graph-theoretic sense) starting and finishing in X.

Let S be a set system, p be a walk, and $B \subseteq S_x$ where x is the starting node of p. The *propagation* of B via p under S, denoted $B +_S p$, is the subset of D defined inductively on the length ℓ of p as follows. If $\ell = 0$ then $B +_S p = B$. Otherwise, $p = p' c_{\ell-1} x_\ell$ where p' is a walk of length $\ell - 1$ ending at $x_{\ell-1}$. Let $c_{\ell-1} = (\mathbf{s}, R)$. Then we define $B +_S p$ to contain all $e \in D$ such that there exists $d \in B +_S p'$ and $\mathbf{t} \in R$ such that for every $1 \le i \le arity(R)$, $\mathbf{t}[i]$ satisfies the following conditions:

1. $\mathbf{t}[i] \in S_{\mathbf{s}[i]}$,
2. if $\mathbf{s}[i] = x_{\ell-1}$ then $\mathbf{t}[i] = d$, and
3. if $\mathbf{s}[i] = x_\ell$ then $\mathbf{t}[i] = e$.

We are now ready to state the result from [18] that we shall use.

Theorem 17 (follows from [18]) *Let I be an instance of* CSP(Γ) *where Γ has bounded width and let S be a set system such that $S_x \neq \emptyset$ for every $x \in X$ and such that for every walk p starting and finishing at the same node x and for every $d \in S_x$, d belongs to $\{d\} +_S p$. Then I is satisfiable.*

Our goal is to design a distributed algorithm that either correctly determines that an instance I is unsatisfiable, or produces a set system S verifying the conditions of Theorem 17. This is not possible in general due to the fact that agents are anonymous and hence a hypothetical algorithm that would generate a walk in a distributed way would be unable to determine if the initial and end nodes are the same. However, thanks to the structure established by Theorem 10, this difficulty can be overcome when Γ has symmetric polymorphisms of all arities because, essentially, the iterated degree of a node can act as its unique identifier. To make this intuition precise we will need to introduce a few more definitions.

We say that a pair $(x, d) \in S$ is *S-supported* if for every walk p starting at x and finishing at a node y with $x \sim_\delta y$, we have that $\{d\} +_S p$ contains d.

Remark 18 We note that if $(x, d) \in S$ is not S-supported and $p = x_0 c_0 \dots x_\ell$ is a walk of minimal length among all walks witnessing that (x, d) is not S-supported then $\ell \leq n2^{|D|}$. Indeed if we let $B_i = \{d\} + x_0 c_0 \dots x_i$, $i = 0, \dots, \ell$ then we have that $(x_i, B_i) \neq (x_j, B_j)$ for every $0 \leq i < j \leq \ell$, since otherwise the shorter walk $x_0 c_0 \dots x_i c_j \dots x_\ell$ would contradict the minimality of p. Since there are n choices for each x_i and $2^{|D|}$ choices for B_i, the bound follows.

We say that a set system S is *safe* if for every solution $\nu \in I$ we have

$$\nu(x) = \nu(y) \text{ for all } x, y \in X \text{ with } x \sim_\delta y \implies \nu(x) \in S_x \text{ for all } x \in X.$$

Then, we have

Lemma 19 *Let S be a safe set system and let $(x, d) \in S$ be a pair that is not S-supported. Then $S \setminus \{(x, d)\}$ is safe.*

Proof Let ν be any solution in I satisfying $\nu(y) = \nu(z)$ for every $y, z \in X$ with $y \sim_\delta z$ and let $p = x_0 c_0 \dots x_\ell$ be any walk in S witnessing that (x, d) is not S-supported, (i.e, p is such that $x_0 = x$, $x_0 \sim_\delta x_\ell$, and $d \notin \{d\} +_S p$). Since S is safe we have that $\nu(y) \in S_y$ for every $y \in X$. It remains to see that $\nu(x) \neq d$, so that the safety condition remains unaltered when (x, d) is removed. First, it follows easily by induction that for every $1 \leq i \leq \ell$, $\nu(x_i) \in \{\nu(x)\} +_S p_i$ where $p_i = x_0 c_0 \dots x_i$. Then, since $\nu(x_\ell) \in \{\nu(x)\} +_S p$, $\nu(x) = \nu(x_\ell)$, and $d \notin \{d\} +_S p$, it follows that $\nu(x) \neq d$. □

Our distributed consistency algorithm (that is, the second phase of Alg) works as follows. Every variable agent $\alpha(x)$ maintains a set $S_x \subseteq D$ in such a way that the

set system S is guaranteed to be safe at all times. As a result of an iterative process S is modified. We shall use S^i to denote the content of S at the ith iteration, where an iteration is, in turn, a loop of $T = 2n(2^{|D|} + 1) = \mathcal{O}(n)$ consecutive rounds. The rationale behind this exact value will be made clear later. Initially, S_x^0 is set to D for every $x \in X$. At iteration i for $i \geq 1$, S^i is obtained by removing all the elements in S^{i-1} that are not S^{i-1}-supported. Then, in at most $n|D| = \mathcal{O}(n)$ iterations we shall obtain a fixed point S^∞.

The key observation is that when Γ has symmetric polymorphisms of all arities, the satisfiability of I can be determined from S^∞. Indeed, if $S_x^\infty = \emptyset$ for some $x \in X$ then we can conclude from the fact that S^∞ is safe and Theorem 10 that I has no solution. Otherwise, S^∞ satisfies the conditions of Theorem 17 and, hence, I is satisfiable.

It remains to see how to compute S^{i+1} from S^i. In an initial preparation step for every iteration, every variable agent $\alpha(x)$ sends S_x^i to all its neighbours. To compute S^{i+1} the algorithm proceeds in rounds. All the messages sent are sets containing triplets of the form (δ_∞, d, B) where $d \in D$, $B \subseteq D$, and δ_∞ is the iterated degree of some variable $x \in X$. It follows from the fact that there are at most n possibilities for the degree of a variable that the size of each message is $\mathcal{O}(ns_{\max})$.

The agents controlling variables and constraints alternate. That is, variables perform internal and send events at even rounds and receive messages at odd rounds, while constraints perform internal and send events at odd rounds and receive messages at even rounds. More specifically, in round $j = 0$ of iteration i, every variable agent $\alpha(x)$ sends to its neighbours the message M containing all triplets of the form $(\delta_\infty(x), d, \{d\})$ with $d \in S_x^i$. At round $2j$ for $j > 0$, $\alpha(x)$ computes $M = M_1 \cup \cdots \cup M_r$ where $M_1, \ldots, M_r$ are the messages it received at the end of round $2j - 1$. Subsequently, for every triplet $(\delta_\infty, d, B) \in M$ with $\delta_\infty = \delta_\infty(x)$ and $d \notin B$, $\alpha(x)$ marks d as 'not S^i-supported'. Finally, it sends message M to all its neighbours. This computation can be done in time $\mathcal{O}(rns_{\max}) = \mathcal{O}(mns_{\max})$ provided that each message is stored as an ordered array.

In round $2j + 1$, every constraint agent $\alpha(c)$ computes from the messages M_x (received from each neighbour $\alpha(x)$ in the previous round) the set M'_x, which contains for every variable $y \in c$ and every (δ_∞, d, B) in M_y, the triplet $(\delta_\infty, d, B +_{S^i} p)$ where $p = ycx$. Finally, it sends to each neighbour $\alpha(x)$ the corresponding message M'_x. Note that while $\alpha(c)$ doesn't know the address of $\alpha(x)$ specifically, knowing the label of the channel that connects them is sufficient to calculate M'_x correctly and send the message accordingly. Moreover, for given y and x, $\alpha(c)$ can compute $B +_{S^i} p$ in $\mathcal{O}(1)$ time as $\alpha(c)$ knows both S_y^i and S_x^i. Hence, since the arity of the relations is fixed (as Γ is fixed) the total running time at iteration $2j + 1$ of a constraint agent $\alpha(c)$ is $\mathcal{O}(ns_{\max})$.

Now it is immediate to show by induction that for every $j \geq 0$, every $x \in X$ and $c \in C$ with $x \in c$ the message sent by $\alpha(x)$ to $\alpha(c)$ at the end of round $2j$ is precisely

$$\{(\delta_\infty(y), d, \{d\} + p) \mid y \in X, d \in S_y^i, p \text{ is a walk of length } j \text{ of the form } p = y \ldots x\}$$

and the message sent by $\alpha(c)$ to $\alpha(x)$ at the end of round $2j + 1$ is precisely

$$\{(\delta_\infty(y), d, \{d\} + p) \mid y \in X, d \in S_y^i, p \text{ is a walk of length } j + 1 \text{ of the form } p = y \ldots cx\}.$$

By Remark 18 only $2n2^{|D|} = T - 2n$ iterations are needed to identify all elements in S^i that are not S^i-supported. Hence, after exactly $T - 2n$ rounds every variable agent $\alpha(x)$ computes S_x^{i+1} by removing all the elements in S^i that are marked as "not S^i-supported". If $S_x^{i+1} = \emptyset$, then $\alpha(x)$ initiates a wave, which is propagated by all its neighbours, broadcasting that an inconsistency was detected. In this case, in at most $2n$ additional rounds all agents can correctly declare that I is unsatisfiable. Otherwise, a new iteration begins.

To sum up, the distributed consistency algorithm consists of $\mathcal{O}(n)$ iterations consisting, each, of $\mathcal{O}(n)$ rounds where the total running time for internal events at a given round is $\mathcal{O}(mns_{\max})$ and the maximum size of each message transmitted is $\mathcal{O}(ns_{\max})$. Together with the bounds given by Lemma 16 for the distributed colour refinement phase, this completes the proof of Theorem 15.

4.3 The Search Algorithm

We conclude by presenting the proof of Theorem 12. The hardness part follows immediately from Theorem 11 as the search problem is as difficult as the decision problem. For the positive result we shall present an adaptation of the algorithm solving the decision version. Let I be an instance of DCSP-Search(Γ) where Γ contains symmetric polymorphisms of all arities. In what follows we shall use intensively the fact that $\mathsf{Pol}(\Gamma)$ is closed under composition. Let $J \subseteq D$ be minimal with the property that $f(D) = J$ for some unary polymorphism f in $\mathsf{Pol}(\Gamma)$. It is fairly standard to show that for every $r \geq 0$ there is a r-ary symmetric operation g in $\mathsf{Pol}(\Gamma)$ such that $g(x, \ldots, x) = x$ for every $x \in J$. Indeed, let f satisfy $f(D) = J$ and let g' be any r-ary symmetric polymorphism in $\mathsf{Pol}(\Gamma)$. Then the unary operation h defined by $h(x) = f \circ g'(x, \ldots, x)$ is a unary polymorphism of Γ. By the choice of f we have $h(D) \subseteq J$. We note that $h(J) = J$ since otherwise h^2 would contradict the minimality of f. Consequently, h restricted to J is a (partial) isomorphism, and so h^{-1} preserves all relations of Γ restricted to the subdomain J. Hence, the r-ary operation g defined as $g = h^{-1} \circ f \circ g'$ satisfies the claim. This implies that if we enlarge the constraint language by adding all singletons $\{d\}$, $d \in J$, the resulting constraint language, which we shall denote by Γ', still has symmetric polymorphisms of all arities. For convenience we also include D in Γ'.

The algorithm has two phases. In the first phase it runs the decision algorithm to determine whether the instance is satisfiable. As a byproduct, every variable agent $\alpha(x)$ has computed its iterated degree $\delta_\infty(x)$ and knows as well its rank in a prescribed ordering of all variable degrees $\delta_\infty^1, \ldots, \delta_\infty^r$, $r \leq n$. This (partial) order will be used to coordinate between the agents. An i-agent, $1 \leq i \leq r$ is any agent $\alpha(x)$ with $\delta_\infty(x) = \delta_\infty^i$. We also assume a fixed ordering on the elements in D. If the instance is unsatisfiable nothing else remains to be done so from now on we shall assume that the instance is satisfiable.

In the second phase the algorithm searches for a solution. Every variable agent $\alpha(x)$ maintains a set $F_x \subseteq D$ with the property that there is a solution ν that *falls within* the set system F, i.e, such that $\nu(x) \in F_x$ for every $x \in X$. Initially every agent $\alpha(x)$ sets $F_x = D$ so it is only necessary to make sure that this condition is preserved during the execution of the algorithm. The second phase contains two

nested loops. The outer loop has r iterations and the inner loop consists of at most $|D|$ iterations so that we shall use iteration (i, d) to indicate the run of the algorithm at the $i = 1, \ldots, r$ iteration of the outer loop and at the iteration d of the inner loop.

At the beginning of iteration (i, d) every variable agent $\alpha(x)$ defines $S_x \subseteq D$ to be $S_x = \{d\}$ whenever $\alpha(x)$ is an i-agent and $S_x = F_x$ elsewhere. Then it runs the distributed consistency algorithm starting at S obtaining a fixed point S^∞. We note that since all initial sets S_x belong to Γ' and Γ' contains symmetric polymorphisms of all arities then the obtained fixed point S^∞ correctly determines whether there exists a solution ν that falls within S. Then every i-agent $\alpha(x)$ checks whether $S_x^\infty = \emptyset$. In case of positive answer nothing else is done and round (i, d) finishes. Otherwise, there must be a solution ν that falls within S, so $\alpha(x)$ sets F_x to $\{d\}$ and starts a wave to indicate to all processes that the ith iteration of the outer loop is finished and that the next iteration of the outer loop can start. When the r iterations of the outer loop have been completed the set system F contains only singletons. The assignment that sets every variable $x \in X$ to the only element in F_x is necessarily a solution. This concludes the proof of Theorem 12.

5 Conclusion

We analysed the complexity of the distributed constraint satisfaction problem on a synchronous, anonymous network parametrised by the constraint language. We showed that, depending on the polymorphisms of Γ, DCSP(Γ) is either solvable in polynomial time, or not solvable altogether. A number of natural open questions arise in this context. For instance, it is not clear whether asynchronous networks are strictly more powerful than their synchronous counterpart. Moreover, it would be interesting to explore the role of allowing agents to make random choices - provided this is not used to create and share unique IDs.

In the spirit of [14], one could consider characterizing the structural restrictions on tractable distributed CSPs, or in other words, determining which *classes of networks* are tractable in the DCSP framework, regardless of the constraint language. The starting point for this analysis could be the work on fibrations by Boldi et al. (see for example [8, 9]). In particular, we propose the question of establishing a connection between the universal fibration of a graph and its iterated degree sequence.

Appendix A: The Distributed Formalism

In this section we formalize the definition of the distributed system model which was used throughout the paper. Note that for simplicity, here we describe a distributed system which only allows message broadcast (as opposed to allowing different messages to be sent through channels with different labels). Nonetheless, as pointed out in Section 2, this does not decrease the expressive power of the system.

Let $\mathbb{N}$ denote the set of positive integers and $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. For a (possibly infinite) set S, we use $\mathbb{N}_0^S$ (as an extension of the power set notation 2^S) to denote the set of all multisets containing just elements from S. Let Σ be a finite alphabet.

A *message* is a finite string of symbols from Σ, that is, an element of Σ^*. A *process* (also called an *agent*) is a 7-tuple $(\Sigma, \Lambda, L, Q, q_o, Q_f, F)$ where Λ is a finite set known as the *memory alphabet*, L is a finite set of *labels*, Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $Q_f \subseteq Q$ denotes the subset of *terminal states*, and F is a Turing-computable function

$$F : Q \times \Lambda^* \times \mathbb{N}_0^{L \times \Sigma^*} \to Q \times \Lambda^* \times \Sigma^*$$

which takes as input the current state, the memory tape, and the multiset of labelled messages received at the previous round, and outputs a new state, an updated memory tape, and a message to be broadcast to the process's neighbours. We require that terminal states are fixed points of F in the sense that for all $q \in Q_f$, $\lambda \in \Lambda^*$, and $M \in \mathbb{N}_0^{L \times \Sigma^*}$ it holds that

$$F(q, \lambda, M) = (q, \lambda, \varepsilon)$$

where $\varepsilon \in \Sigma^*$ denotes the empty string. An output of the function F – that is, a triplet $c = (q, \lambda, s) \in Q \times \Lambda^* \times \Sigma^*$ – is called a *configuration* of the process.

In this paper, a *message-passing network* is an edge-labelled graph whose vertex set is a set of identical processes and where the label set of the network corresponds to the label set of the processes. The neighbourhood of a process i in the network is denoted $N(i)$. Given a message-passing network G with vertex set $[p]$, a *run* of F over G with initial memory $(\lambda_0^1, \ldots, \lambda_0^p)$ is a collection of p sequences of configurations $c_0^i, c_1^i, c_2^i \ldots$, $i \in [p]$ where $c_t^i = (q_t^i, \lambda_t^i, s_t^i)$, such that $q_0^i = q_0$ is the initial state, $s_0^i = \varepsilon$ is the empty string, and for all $t \geq 0$ and all $i \in [p]$, $(q_{t+1}^i, \lambda_{t+1}^i, s_{t+1}^i) = F(q_t^i, \lambda_t^i, M_t^i)$, where $M_t^i = \{\{(\ell_{\{i,j\}}, s_t^j) \mid j \in N(i)\}\}$ and $\ell_{\{i,j\}} \in L$ denotes the label of edge $\{i, j\}$. A run is said to be *terminating* if there exists some integer T such that, for each $i \in [p]$, $q_T^i \in Q_f$ (and hence, by definition of F, $q_t^i = q_T^i$ for all $t > T$). The *output* of the run is the set of terminal memory tapes λ_T^i, $i \in [p]$.

We point out that, while for convenience we define runs to be infinite sequences of configurations, for all practical purposes one can assume that a process effectively halts as soon as it reaches a terminating state.

Notice that, given p identical processes, a run of the algorithm is fully determined by the connectivity and labels of the network, and by the choice of initial memory λ_0^i for each of the p processes. This is generally fairly limited as each process acquires new information via message passing. It could, for instance, contain the processes' unique identifiers if these were available (this is clearly not the case for anonymous networks like the one that we address in this paper). In the case of the DCSP message passing network, the connectivity is as described in Section 2. In particular, the network is bipartite and each edge with its corresponding label describes how a variable participates in a constraint. The initial memory is limited to the numbers n and m of variables and constraints respectively in the DCSP instance, and to a bit which specifies if the process is controlling a variable or a constraint. In particular, it contains no identifier nor any other information about the input instance. Moreover, in the case of DCSP the function F can depend on the constraint language, since this is fixed, but not on the specifics of the input instance. Finally, we remark that while F does not depend explicitly on the round t, this can be stored and updated in the process's

memory, and the messages can even be tagged with a time stamp if this is needed for synchronization purposes, for example, due to hardware issues.

Appendix B: Binary Structures

Throughout the paper, we assumed that both variables and constraints are controlled by agents in a distributed network (in this section, we will refer to this as model 1). However, when all the constraint relations in Γ are binary it is also valid and, indeed, more common to assume that only variables are controlled by agents, and there is a communication channel between any two variable agents $\alpha(x)$ and $\alpha(x')$ whenever x and x' share a constraint (model 2) which is labelled with the constraint relation and the direction of the constraint.

It is very easy to see that in the binary case both models are equivalent. Indeed, for every CSP instance (X, D, C), let (A_1, X, D, C, α_1) and (A_2, X, D, C, α_2) be the associated DCSP instances in model 1 and 2 respectively. It is easy to see that every algorithm in model 2 can be easily simulated by an algorithm in model 1. In particular, it is only necessary that at round $2j$ every variable agent $\alpha_1(x)$ replicates the jth round of $\alpha_2(x)$ (while every constraint agent $\alpha_1(c)$ remains idle). Then, round $2j + 1$ is used to replicate the messages sent at round j. That is, whenever $\alpha_2(x)$ sends a message to a neighbour $\alpha_2(x')$ at round j, $\alpha_1(x)$ sends a message to $\alpha_1(c)$ at round $2j$, where c is the constraint shared by x and x'. At round $2j + 1$ then $\alpha_1(c)$ forwards the message to $\alpha_1(x')$.

Similarly, any algorithm in model 1 can be replicated in model 2. In this case, at a given round j, every agent $\alpha_2(x)$ simulates the internal computation done at round j by $\alpha_1(x)$ and all its neighbours.

Acknowledgements We would like to thank Gergely Neu for useful discussions on the weighted majority algorithm.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Angluin, D.: Local and Global Properties in Networks of Processors (Extended Abstract). In: Miller, R.E., Ginsburg, S., Burkhard, W.A., Lipton, R.J. (eds.) Proceedings of the 12th Annual ACM Symposium on Theory of Computing. April 28-30, 1980, Los Angeles, California, USA, pp. 82–93. ACM (1980)

2. Arora, S., Hazan, E., Kale, S.: The Multiplicative Weights Update Method: a meta-Algorithm and Applications. Theory Comput. **8**(1), 121–164 (2012)
3. Barceló, P., Kostylev, E., Monet, M., Pérez, J., Reutter, J.L., Silva, J.P.: The Logical Expressiveness of Graph Neural Networks. In: 8th International Conference on Learning Representations, ICLR 2020 Addis Ababa, Ethiopia. April 26-30, 2020. OpenReview.net (2020)
4. Barto, L.: The constraint satisfaction problem and universal algebra. Bull. Symb. Log. **21**(3), 319–337 (2015)
5. Barto, L., Krokhin, A.A., Willard, R.: Polymorphisms, and how to use them. In: Krokhin, A.A., Zivný, S. (eds.) The Constraint Satisfaction Problem: Complexity and Approximability, vol. 7 of Dagstuhl Follow-Ups. pp. 1–44. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
6. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Flores Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, C., Francis Song, H., Ballard, A.J., Gilmer, J., Dahl, G.E., Vaswani, A., Allen, K.R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., Pascanu, R.: Relational inductive biases, deep learning, and graph networks. CoRR, abs/1806.01261 (2018)
7. Bejar, R., Krishnamachari, B., Gomes, C.arla., G., Selman, B.: Distributed constraint satisfaction in a wireless sensor tracking system. In: Workshop on distributed constraint reasoning, international joint conference on artificial intelligence. vol. 4 (2001)
8. Boldi, P., Shammah, S., Vigna, S., Codenotti, B., Gemmell, P., Simon, J.: Symmetry breaking in anonymous networks: Characterizations. In: Fourth israel symposium on theory of computing and systems, ISTCS 1996, Jerusalem, Israel. June 10-12, 1996, Proceedings. pp. 16–26. IEEE Computer Society (1996)
9. Boldi, P., Vigna, S.: An effective characterization of computability in anonymous networks. In: Welch, J.L. (ed.) Distributed computing, 15th international conference, DISC 2001, Lisbon, Portugal. October 3-5, 2001, Proceedings. vol. 2180 of Lecture Notes in Computer Science, pp. 33–47. Springer (2001)
10. Bulatov, A.A.: A Dichotomy theorem for nonuniform csps. In: Chris Umans (ed.) 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS, Berkeley, CA, USA, October 15-17, 2017, pp. 319–330. IEEE, Computer Society (2017)
11. Duffy, K.R., Bordenave, C., Leith, D.J.: Decentralized Constraint Satisfaction. IEEE/ACM Trans. Netw. **21**(4), 1298–1308 (2013)
12. Fioretto, F., Pontelli, E., Yeoh, W.: Distributed constraint optimization problems and applications: A survey. J. Artif. Intell. Res. **61**, 623–698 (2018)
13. Fokkink, W.: Distributed algorithms: An intuitive approach MIT Press (2013)
14. Grohe, M.: The Complexity of homomorphism and constraint satisfaction problems seen from the Other side. J. ACM **54**(1), 1:1–1:24 (2007)
15. Grohe, M.: word2vec, node2vec, Graph2vec, x2vec: Towards a Theory of Vector Embeddings of Structured data. In: Dan Suciu, Yufei Tao, and Zhewei Wei, editors, Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020. pp. 1–16. ACM (2020)
16. Martin Grohe, M., Kersting, K., Mladenov, M., Schweitzer, P.: Color refinement and its applications. Van den Broeck, G.; Kersting, K.; Natarajan, S (2017)
17. Jonsson, P., Lagerkvist, V., Nordh, G., Zanuttini, B.: Strong partial clones and the time complexity of SAT problems. J. Comput. Syst. Sci. **84**, 52–78 (2017)
18. Kozik, M.: Solving CSPs using weak local consistency. SIAM J. Comput. **50**(4), 1263–1286 (2021)
19. Krokhin, A.A., Živný, S. (eds.): Editors. The Constraint Satisfaction Problem: Complexity and Approximability. vol. 7 of Dagstuhl Follow-Ups Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
20. Kun, G., O'donnell, R., Tamaki, S., Yoshida, Y., Zhou, Y.: Linear programming, width-1 csps, and robust satisfaction. In: Goldwasser, S. (ed.) Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012, pp. 484–495. ACM (2012)
21. Meisels, A.: Distributed Search by Constrained Agents - Algorithms, Performance, Communication. Advanced Information and Knowledge Processing. Springer (2008)
22. Morris, C., Ritzert, M., Fey, M., Hamilton, W.-L., Lenssen, J.-E., Rattan, G., Grohe, M.: Weisfeiler and Leman go Neural: Higher-order graph neural networks. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial

Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pp. 4602–4609. AAAI Press (2019)
23. Rossi, F., Van Beek, P., Walsh, T. (eds.): Handbook of constraint programming. Vol. 2 of Foundations of Artificial Intelligence. Elsevier (2006)
24. Toenshoff, J., Ritzert, M., Wolf, H., Grohe, M.: RUN-CSP: Unsupervised learning of message passing networks for binary constraint satisfaction problems. CoRR, abs/1909.08387 (2019)
25. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: 7th International conference on learning representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019)
26. Yamashita, M., Kameda, T.: Computing on an anonymous network. In: Dolev, D. (ed.) Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada. August 15-17 1988. pp. 117–130. ACM (1988)
27. Yokoo, M., Durfee, E.H., Ishida. T., Kuwabara, K.: Distributed constraint satisfaction for formalizing distributed problem solving. In: Proceedings of the 12th International conference on distributed computing systems, Yokohama, Japan. June 9-12 1992. pp. 614–621. IEEE Computer Society (1992)
28. Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: The distributed constraint satisfaction problem: Formalization and algorithms. IEEE Trans. Knowl. Data Eng. **10**(5), 673–685 (1998)
29. Yokoo, M., Hirayama, K.: Algorithms for distributed constraint satisfaction: A review. Auton. Agents Multi Agent Syst. **3**(2), 185–207 (2000)
30. Zhuk, D.: Berkeley, CA, USA, October 15-17, 2017. pp. 331–342. IEEE, Computer Society. In: Umans, C. (ed.) 58th IEEE Annual symposium on foundations of computer science, FOCS (2017)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.