Artificial Intelligence 191–192 (2012) 42–60

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

On minimal constraint networks ☆

Georg Gottlob

Department of Computer Science and Oxford Man Institute, University of Oxford, Oxford OX1 3QD, UK

ARTICLE INFO

Article history:
Received 12 May 2012
Received in revised form 26 July 2012
Accepted 28 July 2012
Available online 31 July 2012

Keywords:
Constraints
Minimal network
Complexity
Join decomposition
Structure identification
Database theory
Knowledge compilation

ABSTRACT

In a minimal binary constraint network, every tuple of a constraint relation can be extended to a solution. The tractability or intractability of computing a solution to such a minimal network was a long standing open question. Dechter conjectured this computation problem to be NP-hard. We prove this conjecture. We also prove a conjecture by Dechter and Pearl stating that for $k \geqslant 2$ it is NP-hard to decide whether a single constraint can be decomposed into an equivalent k-ary constraint network. We show that this holds even in case of bi-valued constraints where $k \geqslant 3$, which proves another conjecture of Dechter and Pearl. Finally, we establish the tractability frontier for this problem with respect to the domain cardinality and the parameter k.

1. Introduction

This paper deals with problems related to minimal constraint networks. First, the complexity of computing a solution to a minimal network is determined. Then, the problems of recognizing network minimality and network-decomposability are studied.

1.1. Minimal constraint networks

In his seminal 1974 paper [26], Montanari introduced the concept of a *minimal constraint network*. Roughly, a minimal network is a constraint network where each partial instantiation corresponding to a tuple of a constraint relation can be extended to a solution. Each arbitrary binary network N having variables $\{X_1,\dots,X_v\}$ can be transformed into an equivalent binary minimal network $M(N)$ by computing the set $sol(N)$ of all solutions to N and creating for $1 \leqslant i < j \leqslant v$ a constraint c_{ij} whose scope is (X_i, X_j) and whose constraint relation consists of the projection of $sol(N)$ to (X_i, X_j), and for $1 \leqslant i \leqslant v$ a unary constraint c_i whose scope is (X_i) and whose constraint relation is the projection of $sol(N)$ over (X_i). The minimal network $M(N)$ is unique, and its solutions are exactly those of the original network, i.e., $sol(N) = sol(M(N))$.

An example of a binary constraint network N is given in Fig. 1(a). This network has four variables $X_1,\dots,X_4$ which, for simplicity, all range over the same numerical domain $\{1,2,3,4,5\}$. Its solution, $sol(N)$, which is the join of all relations of N, is shown in Fig. 1(b). The minimal network $M(N)$ is shown in Fig. 1(c).

Obviously, $M(N)$, which can be regarded as an optimally pruned version of N, is hard to compute. But computing $M(N)$ may result in a quite useful *knowledge compilation* [21,5]. In fact, with $M(N)$ at hand, we can answer a number of queries

☆ This paper is a significantly extended version of a paper with the same title presented at the 17th International Conference on Principles and Practice of Constraint Programming (Gottlob, 2011, [17]). The present paper contains new results in addition to those of Gottlob (2011) [17]. Possible future updates will be made available on CORR at http://arxiv.org/abs/1103.1604.

E-mail address: georg.gottlob@cs.ox.ac.uk.

0004-3702/$ – see front matter
http://dx.doi.org/10.1016/j.artint.2012.07.006

X_1	X_2
1	1
1	2
1	3
1	5
2	1
2	5
3	4

X_2	X_3
1	1
1	2
2	2
3	1
4	1
4	3
4	4

X_1	X_3
1	2
2	1
3	1
3	2
4	1
4	2
4	3

X_3	X_4
1	2
2	1
3	1
3	2
4	1
4	2
4	3

(a) Binary constraint network N

X_1	X_2	X_3	X_4
1	1	2	1
1	2	2	1
2	1	1	2
3	4	1	2

(b) Solution $sol(N)$ of N

X_1	X_2
1	1
1	2
2	1
3	4

X_2	X_3
1	1
1	2
2	2
4	1

X_1	X_3
1	2
2	1
3	1

X_1	X_4
1	1
2	2
3	2

X_2	X_4
1	1
1	2
2	1
4	2

X_3	X_4
1	2
2	1

X_1
1
2
3

X_2
1
2
4

X_3
1
2

X_4
1
2

(c) Minimal network $M(N)$

Fig. 1. A binary constraint network N, it solution $sol(N)$, and its minimal network $M(N)$.

in polynomial time that would otherwise be NP-hard. Typically, these are queries that involve one or two variables only, for example, the queries "*is there a solution for which* $X_4 \leqslant 3$?" or "*does N have a solution for which* $X_2 < X_1$?" are affirmatively answered by a simple lookup in the relevant tables of $M(N)$. For the latter query, for example, one just has to look into the first relation table of $M(N)$, whose tuple $\langle 2, 1\rangle$ constitutes a witness. In contrast, in our example, the query "is there a solution for which $X_1 < X_4$?" is immediately recognized to have a negative answer, as the fourth relation of $M(N)$ has no tuple witnessing this inequality. An example of a slightly more involved non-Boolean two-variable query that can be polynomially answered using $M(N)$ is: "*what is the maximal value of* X_2 *such that* X_4 *is minimum over all solutions*?". Again, one can just "read off" the answer from the single relation of $M(N)$ whose variables are those of the query. In our example in Fig. 1, it is the penultimate relation of $M(N)$, that can be easily used to deduce that the answer is 2.

1.2. Computing solutions to minimal constraint networks

In applications such as computer-supported interactive product configuration, such queries arise frequently, but it would be useful to be able to exhibit at the same time a full solution together with the query answer, that is, an assignment of values to all variables witnessing this answer. However, it was even unclear whether the following problem is tractable: Given a non-empty minimal network $M(N)$, compute an arbitrary solution to it. Gaur [11] formulated this as an open problem. He showed that a stronger version of the problem, where solutions restricted by specific value assignments to a pair of variables are sought, is NP-hard, but speculated that finding arbitrary solutions could be tractable. However, since the introduction of minimal networks in 1974, no one came up with a polynomial-time algorithm for this task. This led Dechter to conjecture that this problem is hard [8]. Note that this problem deviates in two ways from classical decision problems: First, it is a search problem rather than a decision problem, and second, it is a *promise problem*, where it is "promised" that the input networks, which constitute our problem instances, are indeed minimal – a promise whose verification is itself NP-hard (see Section 4.1). We therefore have to clarify what NP-hardness means, when referring to such problems. The simplest and probably cleanest definition is the following: The problem is NP-hard if any polynomial algorithms solving it would imply the existence of a polynomial-time algorithm for NP-hard decision problems, and would thus imply $\mathrm{P} = \mathrm{NP}$. In the light of this, Dechter's conjecture reads as follows:

Conjecture 1.1. *(See Dechter [8].) Unless $\mathrm{P} = \mathrm{NP}$, computing a single solution to a non-empty minimal constraint network cannot be done in polynomial time.*

While the problem has interested a number of researchers, it has not been solved until recently. Some progress was made by Bessiere in 2006. In his well-known handbook article "Constraint Propagation" [4], he used results of Cros [6] to show that no backtrack-free algorithm for computing a solution from a minimal network can exist unless the Polynomial Hierarchy collapses to its second level (more precisely, unless $\Sigma_2^p = \Pi_2^p$). However, this does not mean that the problem is intractable. A backtrack-free algorithm according to Bessiere must be able to recognize *each* partial assignment that is extensible to a solution. In a sense, such an algorithm, even if it computes only one solution, must have the potential to compute all solutions just by changing the choices of the variable-instantiations made at the different steps. In more colloquial terms, backtrack-free algorithms according to Bessiere must be *fair to all solutions*. Bessiere's result does not preclude the existence of a less general algorithm that computes just one solution, while being unable to recognize all partial assignments, and thus being unfair to some solutions.

The simple example in Fig. 1, by the way, shows that the following naïve backtrack-free strategy is doomed to fail: Pick an arbitrary tuple from the first relation of $M(N)$, expand it by a suitable tuple of the second relation, and so on. In fact, if we just picked the first tuple $\langle 1, 1\rangle$ of the first relation, we could combine it with the first tuple $\langle 1, 1\rangle$ of the second relation and obtain the partial instantiation $X_1 = X_2 = X_3 = 1$. However, this partial instantiation is not part of a solution, as it cannot be expanded to match any tuple of the third relation. While this naïve strategy fails, one may still imagine the existence of a more sophisticated backtrack-free strategy, that pre-computes in polynomial time some helpful data structure before embarking on choices. However, as we show in this paper, such a strategy cannot exist unless $\mathrm{NP} = \mathrm{P}$.

In the first part of this paper, we prove Dechter's conjecture by showing that every polynomial-time search algorithm A that computes a single solution to a minimal network can be transformed into a polynomial-time decision algorithm A^* for the classical satisfiability problem 3SAT. The proof is carried-out in Section 3. We first show that each SAT instance can be transformed in polynomial time into an equivalent one that is highly symmetric (Section 3.1). Such symmetric instances, which we call *k-supersymmetric*, are then polynomially reduced to the problem of computing a solution to a minimal binary constraint network (Section 3.2). We further consider the case of bounded domains, that is, when the input instances are such that the cardinality of the overall domain of all values that may appear in the constraint relation is bounded by some fixed constant c. By a simple modification of the proof of the general case, it is easily seen that even in the bounded domain case, the problem of computing a single solution remains NP-hard (Section 3.3).

Our hardness results for computing relations can be reformulated in terms of database theory. Every constraint network N can be seen as a relational database instance, where each constraint of N corresponds to a single relation instance. The set $sol(N)$ of all solutions to a binary constraint network (or database instance) N is identical to the relation obtained by performing the natural join of all relation instances of N. The minimal network $M(N)$ is then a lossless decomposition of $sol(N)$ according to the join dependency $*[S]$, where S is the schema of $M(N)$. Our main hardness result thus implies that it is coNP-hard to recover an arbitrary single tuple of a relation instance R (called a *universal relation*) from its lossless decomposition according to a given single join dependency, when only this decomposition is given. Lossless decompositions and universal relations have been studied for many decades, and they were recently related to hidden variable models in quantum mechanics [1,2].

1.3. Minimality checking and structure identification

In Section 4.1, we generalize and slightly strengthen a result by Gaur [11] by showing that it is NP-hard to determine whether a k-ary network is minimal, even in case of bounded domains.

In Section 4.2, we study the complexity of checking whether a network N consisting of a single constraint relation (typically of arity $\geqslant k$) can be represented by an equivalent k-ary constraint network. Note that this is precisely the case iff there exists a k-ary *minimal* network M equivalent to N, i.e., one such that $sol(M) = sol(N)$. Dechter and Pearl [9] regarded this problem as a relevant complexity problem of *structure identification for relational data*, i.e., of checking whether an element of a general class of objects (in this case, data relations) belongs to a structurally simpler subclass (in this case, k-decomposable relations). This problem is equivalent to the database problem of testing whether a given instance of a data relation satisfies a specific join dependency. Dechter and Pearl conjectured that the problem is NP-hard for $k \geqslant 2$. We prove this conjecture by showing the problem to be coNP-complete for each $k \geqslant 2$.

A special case considered in [9] is the one of bi-valued constraints, that is, constraints over the Boolean domain. For bi-valued constraints, the above structure identification problem is equivalent to testing whether a Boolean formula represented by the explicit list of all its models is equivalent to a k-CNF. For $k = 2$ this problem is known to be tractable (see [7,11]). Dechter and Pearl [9] conjectured it to be NP-hard for every fixed $k > 2$. In Section 4.3 we prove this conjecture and show that deciding whether bi-valued relations are k-decomposable is coNP-complete for each fixed $k > 2$. Moreover, we show in Section 4.4 that the representability of tri-valued constraints (and more generally r-valued constraints for $r \geqslant 3$) as a k-ary network is coNP-hard for each fixed $k \geqslant 2$. Put together, our results allow us to trace the precise tractability frontier for the problem of relational structure identification in terms of the domain cardinality and the parameter k. This is visualized in Fig. 3 in Section 4.4.

The paper is concluded in Section 5 by a brief discussion of the practical significance of our main result, a proposal for the enhancement of minimal networks, and some hints at possible future research.

2. Preliminaries and basic definitions

While most of the definitions in this section are adapted from the standard literature on constraint satisfaction, in particular [8,4], we sometimes use a slightly different notation which is more convenient for our purposes.

Constraints, networks, and solutions We assume a totally ordered infinite set $(\mathbf{X}, \prec)$ of variables. For $X_i, X_j \in \mathbf{X}$, $X_i \prec X_j$ means that X_i is smaller than X_j according to the "$\prec$" ordering. We assume that all variables of constraint networks are from this set. A *k-ary constraint* c is a pair $(scope(c), rel(c))$. The scope $scope(c)$ of c is a sequence $(X_{i_1}, X_{i_2}, \ldots, X_{i_k})$ of k distinct variables from $\mathbf{X}$, where $X_{i_1} \prec X_{i_2} \prec \cdots \prec X_{i_k}$, and where each variable X_{i_j} has an associated finite domain $dom(X_{i_j})$. The relation $rel(c)$ of c is a subset of the Cartesian product $dom(X_{i_1}) \times dom(X_{i_2}) \times \cdots \times dom(X_{i_k})$. The set $\{X_{i_1}, \ldots, X_{i_k}\}$ of all variables occurring in $scope(c)$ is denoted by $var(c)$. Given that each set of variables is totally ordered by $\prec$, we shall identify each set U of variables, whenever convenient, with the list $\vec{U}$ of its elements ordered according to $\prec$. We thus may write $scope(c) = U$ instead of $scope(c) = \vec{U}$. More generally, since the concepts of lists of distinct elements and ordered sets coincide, we may use set-theoretic notation to express facts about such lists. For example, if s denotes a scope, we may write $X_i \in s$ to express that X_i is a variable in this scope. If, moreover, U denotes a list (or even an unordered set) of variables, we may write $s \subseteq U$ to say that each variable of s is also an element of U, and so on.

A *constraint network* N consists of a finite set $var(N) = \{X_1, \ldots, X_v\}$ of variables with associated domains $dom(X_i)$ for $1 \leqslant i \leqslant v$, and a set of constraints $cons(N) = \{c_1, \ldots, c_m\}$, where for $1 \leqslant i \leqslant m$, $var(c_i) \subseteq var(N)$.

If $U \subseteq var(N)$ is a set of variables, then $dom(U) = \bigcup_{X \in U} dom(X)$. The *domain* $dom(N)$ of a constraint network N is defined by $dom(N) = dom(var(N))$. The *schema* of N is the set $schema(N) = \{scope(c) \mid c \in cons(N)\}$ of all scopes of the constraints of N. If S is a schema, then $var(S)$ denotes the set of all variables in the scopes of S. In particular, if S is the schema of network N, we have $var(S) = var(N)$. We call N *binary* (*k-ary*) if $arity(c) \leqslant 2$ ($arity(c) \leqslant k$) for each constraint $c \in cons(N)$.

Let N be a constraint network. An *instantiation mapping* for a set of variables $W \subseteq var(N)$ is a mapping $\theta : W \longrightarrow dom(W)$, such that for each $X \in var(N)$, $\theta(X) \in dom(X)$. We call $\theta(W)$ an instantiation of W. An instantiation of a proper subset W of $var(N)$ is called a *partial instantiation* while an instantiation of $var(N)$ is called a *full instantiation* (also *total instantiation*). A constraint c of N is *satisfied* by an instantiation mapping $\theta : W \longrightarrow dom(W)$ if whenever $var(c) \subseteq W$, then $\theta(scope(c)) \in rel(c)$. An instantiation mapping $\theta : W \longrightarrow dom(W)$ is *consistent* if it is satisfied by all constraints. By abuse of terminology, if θ is understood and is consistent, then we may also say that $\theta(W)$ is consistent. A *solution* to a constraint network N is a consistent full instantiation for N. The set of all solutions of N is denoted by $sol(N)$. N is *solvable* iff $sol(N) \neq \emptyset$. Whenever useful, we will identify the solution set $sol(N)$ with a single constraint whose scope is $var(N)$ and whose relation consists of all tuples in $sol(N)$. We assume without loss of generality, that for each set of variables $W \subseteq var(N)$ of a constraint network, there exists at most one constraint c such that $var(c) = W$. (In fact, if there are two or more constraints with exactly the same variables in the scope, an equivalent single constraint can always be obtained by intersecting the constraint relations.)

Complete networks The *complete schema* S_k^U over a set of variables U denotes the schema consisting of all non-empty constraint scopes of arity at most k contained in U. For example, if $U = \{X_1, X_2\}$, then $S_k^U = \{(X_1), (X_2), (X_1, X_2)\}$. If the set of variables U is understood, we will write S_k instead of S_k^U. A k-ary constraint network N is *complete*, if its schema is $S_k^{var(N)}$. For each fixed constant k, each k-ary constraint network N can be transformed by a trivial polynomial reduction into an equivalent complete k-ary network N^+ with $sol(N) = sol(N^+)$. In fact, if $\ell \leqslant k$, then for each (ordered) set of variables $W = \{X_{i_1}, \ldots, X_{i_\ell}\}$ that is in no scope of N, we may just add the trivial constraint $\top_W$ with $scope(\top_W) = (X_{i_1}, \ldots, X_{i_\ell})$ and $rel(\top_W) = dom(X_{i_1}) \times dom(X_{i_2}) \times \cdots \times dom(X_{i_\ell})$. For this reason, we may, whenever useful, restrict our attention to complete networks. Some authors, such as Montanari [26] who studies binary networks, assume by definition that all networks are complete, others, such as Dechter [8] make this assumption implicitly.

Intersections of networks, containment, and projections Let N_1 and N_2 be two constraint networks defined over the same schema S (that is, the same set S of constraint scopes). The *intersection* $M = N_1 \cap N_2$ of N_1 and N_2 is the network having $var(M) = var(N_1) = var(N_2)$, and having a constraint c^s, for each $s \in S$, such that $scope(c^s) = s$ and $rel(c^s) = rel(c_1^s) \cap rel(c_2^s)$, where c_1 and c_2 are the constraints having scope s of N_1 and N_2, respectively. The intersection of arbitrary families of constraint networks defined over the same schema is defined in a similar way. For two networks N_1 and N_2 over the same schema S, we say that c_1 is *contained in* c_2, and write $N_1 \subseteq N_2$, if for each $s \in S$, and for $c_1 \in cons(N_1)$ and $c_2 \in cons(N_2)$

with $scope(c_1) = scope(c_2) = s$, $rel(c_1) \subseteq rel(c_2)$. If c is a constraint over a set of variables $W = \{X_1, \ldots, X_v\}$ and $V \subseteq W$, then the projection $\Pi_V(c)$ is the constraint whose scope is V, and whose relation is the projection over V of $rel(c)$. Let c be a constraint and S a schema consisting of one or more scopes contained in $scope(c)$, then $\Pi_S(c) = \{\Pi_s(c) \mid s \in S\}$.

Minimal networks Let c be a constraint with $var(c) = U$. The projection $\Pi_{S_k^U}(c)$ will henceforth just denote by $\Pi_{S_k}(c)$. Thus $\Pi_{S_k}(c)$ is the constraint network obtained by projecting c over all scopes in the schema S_k^U (simply denoted by S_k), i.e., over all non-empty ordered lists of at most k variables from $var(c)$. In particular, the constraints of $\Pi_{S_2}(c)$ are precisely all $\Pi_W(c)$ such that $W \subseteq var(c)$ is a unary or binary scope.

It was first observed in [26] that for each binary constraint network N, there is a unique binary *minimal network* $M(N)$ that consists of the intersection of all binary networks N' over schema S_2 for which $sol(N') = sol(N)$. Minimality here is with respect to the above defined "⊆"-relation among binary networks. More generally, for each k-ary network N there is a unique k-ary minimal network $M_k(N)$ that is the intersection of all k-ary networks N' over schema S_k for which $sol(N') = sol(N)$. (For the special case $k = 2$ we have $M_2(N) = M(N)$.) The following is well known [26,27,8,4,19] and easy to see:

- $M_k(N) = \Pi_{S_k}(sol(N))$.
- $M_k(N) \subseteq N'$ for all k-ary networks N' with $sol(N') = sol(N)$.
- A k-ary network N is satisfiable (i.e., has at least one solution) iff $M_k(N)$ is non-empty.
- A k-ary network N is minimal iff $\Pi_{S_k}(sol(N)) = N$.
- A k-ary network N is minimal iff $M_k(N) = N$.
- A network N over schema S_k is minimal iff there exists a *universal relation* ρ for N, that is, a single constraint ρ such that $N = \Pi_{S_k}(\rho)$. In this case N is said to be *join consistent* (see [19]).

It is obvious that for $k \geqslant 2$, $M_k(N)$ is hard to compute. In fact, just *deciding* whether for a network N, $M_k(N)$ is the empty network is coNP-complete, because this decision problem is equivalent to deciding whether N has no solution. (Recall that deciding whether a network N has a solution is NP-complete [23].) In this paper, however, we are not primarily interested in computing $M_k(N)$, but in computing a single solution, in case $M_k(N)$ has already been computed and is known.

Graph-theoretic characterization of minimal networks An *n-partite graph* is a graph whose vertices can be partitioned into n disjoint sets so that no two vertices from the same set are adjacent. It is well known (see, e.g., [31]) that each binary constraint network N on n variables can be represented as an n-partite graph G_N. The vertices of G_N are possible instantiations of the variables by their corresponding domain values. Thus, for each variable X_i and possible domain value $a \in dom(X_i)$, there is a vertex X_i^a. Two vertices X_i^a and X_j^b are connected by an edge in G_N iff the relation of the constraint c_{ij}^N with scope (X_i, Y_j) contains the tuple (a, b).[1] Gaur [11] gave the following nice characterization of minimal networks: A solvable[2] complete binary constraint network N on n variables is minimal iff each edge of N is part of a clique of size n of G_N. Note that by definition of G_N as an n-partite graph, there cannot be any clique in G_N with more than n vertices, and thus the cliques of n vertices are precisely the maximum cliques of G_N.

Satisfiability problems An instance C of the *satisfiability* (*SAT*) problem is a conjunction of clauses (often just written as a *set* of clauses), each of which consists of a disjunction (*often written as set*) of literals, i.e., of positive or negated *propositional variables*. Propositional variables are also called (*propositional*) *atoms*. If α is a set of clauses or a single clause, then we denote by $propvar(\alpha)$ the set of all propositional variables occurring in α.

A 3SAT instance is a SAT instance each clause of which is a disjunction of at most three literals. 3SAT is the problem of deciding whether a 3SAT instance is satisfiable.

3. NP-hardness of computing minimal network solutions

To show that computing a single solution to a minimal network is NP-hard, we will do exactly the contrary of what people – or automatic constraint solvers – usually do whilst solving a constraint network or a SAT instance. While everybody aims at breaking symmetries, we will actually *introduce additional symmetry* into a 3SAT instance and its corresponding constraint network representation. This will be achieved by the *Symmetry Lemma* to be proved in the next section.

3.1. The Symmetry Lemma

The following lemma shows that, for each fixed $k \geqslant 1$, one can transform an arbitrary 3SAT instance C in polynomial time into a satisfiability-equivalent highly symmetric SAT instance C^* such that, whenever C (and thus C^*) is satisfiable, each truth value assignment to any k variables of C^* can be extended to a truth value assignment satisfying C^*. Before stating the lemma, let us formally define this notion of symmetry, which we refer to as *supersymmetry*.

[1] We disregard unary relations of N here; in fact, each unary relation of a constraint network can be eliminated by appropriately restricting the domain of its scope variable.

[2] We here refer to solvability according to our definition; Gaur uses a different definition of this term.

Definition 3.1. For $k \geqslant 1$, a SAT instance C is *k-supersymmetric* if C is either unsatisfiable or if for each set of k propositional variables $\{p_1, \ldots, p_k\} \subseteq propvar(C)$ and for each arbitrary truth value assignment η to $\{p_1, \ldots, p_k\}$, there exists a satisfying truth value assignment τ for C that extends η. A SAT instance that is 2-supersymmetric is also called *supersymmetric*.

Assume $k < k'$. By the above definition, if a SAT instance C is k'-supersymmetric, then C is also k-supersymmetric. However, a k-supersymmetric SAT instance C is not necessarily also k'-supersymmetric.

Lemma 3.1 *(Symmetry Lemma). For each fixed integer $k \geqslant 1$, there is a polynomial-time transformation T that transforms each 3SAT instance C into a k-supersymmetric SAT instance C^* such that C is satisfiable iff C^* is satisfiable.*

We illustrate the proof of Lemma 3.1 by an example. A full proof is given in Appendix A.

Proof. (*Illustration by Example*) Consider the 3SAT instance $C = C_1 \wedge C_2 \wedge C_3$, where

$$C_1 = \quad p \quad \vee \quad \neg q \quad \vee \quad r$$

$$C_2 = \neg p \quad \vee \quad \neg q$$

$$C_3 = \qquad\qquad q$$

Clearly, the above 3SAT instance C, while satisfiable, is not even 1-supersymmetric, and therefore, a fortiori, not k-supersymmetric for any $k \geqslant 1$. To see this, observe that the partial truth value assignment assigning *false* to q always falsifies clause C_3, and can thus not be extended to a satisfying truth value assignment for C. In the sequel, we illustrate how C can be transformed by a polynomial-time transformation T into a satisfiable supersymmetric SAT instance $C^* = T(C)$. To this aim we introduce to each propositional variable v of C a set $New(v)$ of five new propositional variables. In particular, we have

$$New(p) = \{p_1, p_2, p_3, p_4, p_5\},$$

$$New(q) = \{q_1, q_2, q_3, q_4, q_5\}, \quad \text{and}$$

$$New(r) = \{r_1, r_2, r_3, r_4, r_5\}.$$

We now create C^* from C by taking the conjunction of all clauses obtained by replacing in each clause of C each positive literal v in all possible ways by the disjunction $v_i \vee v_j \vee v_k$ of three elements $v_i, v_j, v_k \in New(v)$, and by replacing each negative literal $\neg v$ in all possible ways by the disjunction $\neg v_i \vee \neg v_j \vee \neg v_k$, where v_i, v_j, v_k are elements of $New(v)$. Each clause is thus replaced by a multitude of other clauses that are all taken in conjunction. In particular, in our example, clause C_1 will actually be replaced by the conjunction of the following 1000 clauses $C_1^1 \ldots C_1^{1000}$:

$$\begin{array}{llllll}
C_1^1: & p_1 \vee p_2 \vee p_3 & \vee & \neg q_1 \vee \neg q_2 \vee \neg q_3 & \vee & r_1 \vee r_2 \vee r_3; \\
C_1^2: & p_1 \vee p_2 \vee p_3 & \vee & \neg q_1 \vee \neg q_2 \vee \neg q_3 & \vee & r_1 \vee r_2 \vee r_4; \\
C_1^3: & p_1 \vee p_2 \vee p_3 & \vee & \neg q_1 \vee \neg q_2 \vee \neg q_3 & \vee & r_1 \vee r_2 \vee r_5; \\
\ldots & \ldots\ldots & . & \ldots\ldots & . & \ldots\ldots \\
C_1^{10}: & p_1 \vee p_2 \vee p_3 & \vee & \neg q_1 \vee \neg q_2 \vee \neg q_3 & \vee & r_3 \vee r_4 \vee r_5; \\
C_1^{11}: & p_1 \vee p_2 \vee p_3 & \vee & \neg q_1 \vee \neg q_2 \vee \neg q_4 & \vee & r_1 \vee r_2 \vee r_3; \\
\ldots & \ldots\ldots & . & \ldots\ldots & . & \ldots\ldots \\
C_1^{1000}: & p_3 \vee p_4 \vee p_5 & \vee & \neg q_3 \vee \neg q_4 \vee \neg q_5 & \vee & r_3 \vee r_4 \vee r_5.
\end{array}$$

Similarly, clause $C_2 = \neg p \vee \neg q$ is replaced by the following 100 clauses $C_2^1 \ldots C_2^{100}$:

$$\begin{array}{llll}
C_2^1: & \neg p_1 \vee \neg p_2 \vee \neg p_3 & \vee & \neg q_1 \vee \neg q_2 \vee \neg q_3; \\
C_2^1: & \neg p_1 \vee \neg p_2 \vee \neg p_3 & \vee & \neg q_1 \vee \neg q_2 \vee \neg q_4; \\
\ldots & \ldots\ldots & . & \ldots\ldots \\
C_2^{100}: & \neg p_3 \vee \neg p_4 \vee \neg p_5 & \vee & \neg q_3 \vee \neg q_4 \vee \neg q_5.
\end{array}$$

Finally, clause $C_3 = p$ is replaced by the following 10 clauses $C_3^1, \ldots, C_3^{10}$:

$$\begin{array}{ll}
C_3^1: & q_1 \vee q_2 \vee q_3; \\
C_3^2: & q_1 \vee q_2 \vee q_4; \\
\ldots & \ldots\ldots \\
C_3^{10}: & q_3 \vee q_4 \vee q_5.
\end{array}$$

The SAT instance $C^* = T(C)$ then consists of the conjunction of all these clauses:

$$C^* = C_1^1 \wedge \cdots \wedge C_1^{1000} \wedge C_2^1 \wedge \cdots \wedge C_2^{100} \wedge C_3^1 \wedge \cdots \wedge C_3^{10}.$$

We claim – and formally prove in Appendix A – that the above transformation from a 3SAT instance C to a SAT instance C^* satisfies the following two key facts:

Fact 1: *C^* is satisfiable iff C is satisfiable (in our example, C^* is thus satisfiable).* In fact, each satisfiable truth value assignment τ to the propositional variables of C can be transformed to a satisfying truth value assignment τ^* to C^* as follows: If $\tau(v) = \mathit{true}$, then let τ^* assign *true* to at least three propositional variables in $\mathit{New}(v)$, and *false* to all others, and if $\tau(v) = \mathit{false}$, then let τ^* assign *false* to at least three propositional variables in $\mathit{New}(v)$, and *true* to all others. In our example, for instance, consider the truth value assignment τ satisfying C, where $\tau(p) = \mathit{false}$ and $\tau(q) = \tau(r) = \mathit{true}$. This truth value assignment satisfies r and therefore C_1. The assignment τ^* to C^* thus assigns *true* to at least three atoms from $\mathit{New}(r) = \{r_1, r_2, r_3, r_4, r_5\}$, assume, for example to $\{r_1, r_4, r_5\}$. But each 3-element subset of $\mathit{New}(r)$ has a non-empty intersection with each other non-empty three element subset of $\mathit{New}(r)$, and thus with the set of atoms of each and every clause $C_1^i \in \{C_1^1, \ldots, C_1^{1000}\}$. Therefore, each such clause C_1^i is satisfied. For example, C_1^1 has an r_1 in common with the set $\{r_1, r_4, r_5\}$, and so must be satisfied by τ^* for $\tau^*(r_1) = \mathit{true}$. A similar argument holds for negative literals. Applying the same type of reasoning to all clauses C_i of C, given that each such C_i has at least one literal satisfied by τ, all clauses C_i^j of C^* are satisfied by τ^*. In summary, τ^* satisfies C^*. Vice versa, we show in the full proof that if C^* is satisfiable, then so must be C.

Fact 2: *C^* is supersymmetric.* Intuitively, this is due to the great choice of truth value assignments to the propositional variables in $\mathit{New}(v)$, when constructing a satisfying assignment τ^* for C^*, as above, from an assignment τ for C. Imagine, for illustration, we'd like to construct a truth value assignment τ^* satisfying our example-instance C^*, such that $\tau^*(p_1) = \mathit{true}$ and $\tau^*(q_3) = \mathit{false}$. Note that no truth value assignment to instance C can actually satisfy p or falsify q. Notwithstanding, we are able to find an appropriate τ^* with the desired properties. We start with an arbitrary satisfying truth value assignment τ to C, for example, the one where $\tau(p) = \mathit{false}$ and $\tau(q) = \tau(r) = \mathit{true}$. To construct τ^*, let us first define τ^* on the elements of $\mathit{New}(p) = \{p_1, \ldots, p_5\}$. According to the construction rules for τ^* in the previous paragraph, given that $\tau(p) = \mathit{false}$, τ^* must assign *false* to at least three elements of $\mathit{New}(p)$, but not necessarily to all elements of $\mathit{New}(p)$. This leaves us the freedom of assigning *true* to p_1. So, we can, for example, assign *false* to p_2, p_3, p_4, and p_5, and *true* to p_1. Similarly, given that $\tau(q) = \mathit{true}$, τ^* must assign *true* to at least three elements of $\mathit{New}(q)$, which can be done while fulfilling at the same time our requirement that $\tau^*(q_3) = \mathit{false}$. For example, let $\tau^*(q_1) = \tau^*(q_2) = \tau^*(q_5) = \mathit{true}$ and $\tau^*(q_3) = \tau^*(q_4) = \mathit{false}$. Finally, the only requirement regarding the truth values assigned by τ^* to the elements of $\mathit{New}(r)$ is that at least three of these propositional variables be assigned *true*. Thus, for example, let $\tau^*(r_1) = \tau^*(r_2) = \cdots = \tau^*(r_5) = \mathit{true}$. In summary, it is easy to see (and actually follows from Fact 1) that the truth value assignment τ^* constructed this way satisfies C^*. Moreover, τ^* extends the initially given partial truth value assignment $\tau^*(p_1) = \mathit{true}$ and $\tau^*(q_3) = \mathit{false}$. More generally, for every pair v, w of propositional variables of C^*, and for every truth value assignment η to $\{v, w\}$, one can construct a truth value assignment τ^* that extends η and satisfies C^*. This shows that C^* is 2-supersymmetric, i.e., supersymmetric.

As easily seen, the transformation from an arbitrary 3SAT instance C to the corresponding C^* is polynomial-time computable. Together with Facts 1 and 2, this informally proves the Symmetry Lemma for $k = 2$. For $k > 2$, the proof is analogous. □

Remark. The concept of supersymmetry is somewhat related to the notions of *quadrangle* and *subquadrangle* defined in [30] and further discussed in [20]. A quadrangle is a single constraint c that is satisfied for all value assignments that assign any arbitrary value from $\mathit{dom}(X)$ to each variable X in $\mathit{scope}(c)$. Thus, the constraint relation $\mathit{rel}(c)$ of a quadrangle c simply consists of a Cartesian product of domains. An n-ary constraint c is a subquadrangle if each projection of c to $n-1$ or fewer variables from $\mathit{scope}(c)$ is a quadrangle. Generalizing this notion, we define a *k-subquadrangle* to be a constraint, all of whose projections to k variables are quadrangles. In this context, Lemma 3.1 may be reformulated as follows: For each $k \geqslant 1$, every satisfiable 3SAT instance C can be transformed to a satisfiable SAT instance C^* whose solution relation $\mathit{sol}(C^*)$ is a k-subquadrangle.

3.2. Intractability of computing solutions

The Symmetry Lemma is used for proving our main intractability result.

Theorem 3.1. *For each fixed constant $k \geqslant 2$, unless NP = P, computing a single solution from a minimal k-ary constraint network N cannot be done in polynomial time. The problem remains intractable even if the cardinality of each variable-domain is bounded by a fixed constant.*

Proof. We first prove the theorem for $k = 2$. Assume A is an algorithm that computes in time $p(n)$, where p is some polynomial, a solution $A(N)$ to each non-empty minimal binary constraint network N of size n. We will construct a polynomial-time 3SAT-solver A^* from A. The theorem then follows.

Let us first define a simple transformation S from SAT instances to equivalent binary constraint networks. S transforms conjunctions $K = K_1 \wedge \cdots \wedge K_r$ of at least two clauses into binary constraint networks $S(K) = N_K$ as follows. The set of variables $var(N_K)$ is defined by $var(N_K) = \{K_1, \ldots, K_r\}$. For each variable K_i of N_K, the domain $dom(K_i)$ consists exactly of all literals appearing in K_i. For each distinct pair of clauses (K_i, K_j), $i < j$, there is a constraint c_{ij} having $scope(c_{ij}) = (K_i, K_j)$ and $rel(c_{ij}) = (dom(K_i) \times dom(K_j)) - \{(p, \bar{p}), (\bar{p}, p) \mid p \in propvar(K)\}$. Moreover, for each variable K_i there is a unary constraint c_i whose scope contains the single variable K_i, such that $rel(c_i) = dom(K_i)$. It is easy to see that K is satisfiable iff N_K is solvable. Basically, N_K is solvable, iff we can pick one literal per clause such that the set of all picked literals contains no atom together with its negation. But this is just equivalent to the satisfiability of K. The transformation S is clearly polynomial-time computable.

Now consider constraint networks $N_{C^*} = S(C^*)$, where C^* is obtained via transformation T as in Lemma 3.1 from some 3SAT instance C, i.e., $C^* = T(C)$. In a precise sense, N_{C^*} inherits the high symmetry present in C^*. In fact, if C^* is satisfiable, then, by Lemma 3.1, for every pair ℓ_1, ℓ_2 of non-contradictory literals, there is a satisfying assignment that makes both literals true. Thus, if C^* (and thus C) is satisfiable, for every constraint c_{ij}, we may pick each pair (ℓ_1, ℓ_2) in $rel(c_{ij})$ as part of a solution, and thus no such pair is useless. Moreover, if C^* is satisfiable, then, clearly, each value in the relations of each unary constraint c_i is part of a solution. It follows that if C^* – and thus C – is satisfiable, then $M(N_{C^*}) = N_{C^*}$, which means that N_{C^*} is minimal. We thus have:

$(*)$ If C is satisfiable then N_{C^*} is non-empty and minimal.

We are now ready for specifying our 3SAT-solver A^* that works in polynomial time, and hence witnesses NP = P. Algorithm A^* is also illustrated by the flowchart in Fig. 2. The input of A^* is a 3SAT input instance C. We here assume without loss of generality that C has at least two clauses. A^* works as follows:

1. Apply transformation T to C and get $C^* := T(C)$.
   Note: *C^* is supersymmetric and C^* is satisfiable iff C is.*
2. Apply transformation S to C^* and get $N_{C^*} := S(C^*)$.
   Note: *N_{C^*} solvable $\Leftrightarrow$ C^* satisfiable $\Leftrightarrow$ C satisfiable.*
3. Run A on input N_{C^*} for $p(|N_{C^*}|)$ steps; denote by w the output at this point.
   Note: *If C (and thus C^*) is satisfiable, then N_{C^*} is a solvable minimal network, and thus w is a solution to N_{C^*}; otherwise N_{C^*} is unsolvable, and w is the empty string or any string other than a solution to N_{C^*}.*
4. Check if w is a solution to N_{C^*}.
5. If w is not a solution to N_{C^*} then output "C unsatisfiable" and stop.
   Note: *In fact, if w is not a solution to N_{C^*} then N_{C^*} is either empty or non-minimal. By the contrapositive of Fact $(*)$, C must then be unsatisfiable.*
6. If w is a solution to N_{C^*} then output "C satisfiable" and stop.
   Note: *If w is a solution, then N_{C^*} is solvable, and thus C^* and C are satisfiable.*

Each step of A^* requires polynomial time only. The polynomial runtime of step 3 depends parametrically on the *fixed* polynomial p. A^* is thus a polynomial-time 3SAT solver. The theorem for $k = 2$ follows.

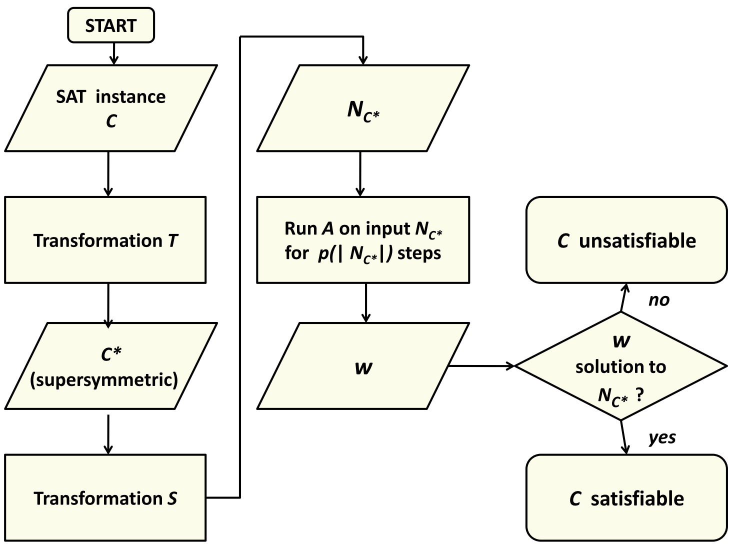

Fig. 2. Flowchart of the 3SAT-solver A^*.

Note that C^*, as constructed in the proof of Theorem 3.1, is a 9SAT instance, hence the cardinality of the domain of each variable of N_{C^*} is bounded by 9.

For $k > 2$, the proof is analogous, the main change being that the transformation S now creates an ℓ-ary constraint c_L for each (ordered) set L of $\ell \leqslant k$ clauses from C. The resulting constraint network $N_{C^*} = S(C^*)$, where C^* is as constructed in Lemma 3.1 then does the job. □

3.3. The case of bounded domains

Theorem 3.1 states that the problem of computing a solution from a non-empty minimal binary network is intractable even in case the cardinalities of the domains of all variables are bounded by a constant. However, if we take the total domain $dom(N)$, which is the set of *all* literals of C^*, its cardinality is unbounded. This notwithstanding, the following simple corollary to Theorem 3.1 shows that even in case $|dom(N)|$ is bounded, computing a single solution from a minimal network N is hard.

Corollary 3.1. *For each fixed $k \geqslant 2$, unless NP = P, computing a single solution from a minimal k-ary constraint network N cannot be done in polynomial time, even in case $|dom(N)|$ is bounded by a constant.*

Proof. We prove the result for $k = 2$; for higher values of k, the proof is totally analogous. The key fact we exploit here is that each variable K_a of N_{C^*} in the proof of Theorem 3.1 has a domain of exactly nine elements, corresponding to the nine literals occurring in clause K_a of C^*. We "standardize" these domains by simply renaming the nine literals for each variable by the numbers 1 to 9. We thus get an equivalent minimal constraint network with a total domain of cardinality 9. □

4. Minimal network recognition and structure identification

In this section we first deal with the complexity of recognizing whether a k-ary network M is the minimal network of a k-ary network N (Section 4.1). We then study the problem of deciding whether a k-ary network M is the minimal network of a single constraint (Section 4.2).

4.1. Minimal network recognition

An algorithmic problem of obvious relevance is recognizing whether a given network is minimal. Using the graph-theoretic characterization of minimal networks described in Section 2, Gaur [11] has shown the following for binary networks:

Proposition 4.1. *(See Gaur [11].) Deciding whether a complete binary network N is minimal is NP-complete under Turing reductions.*

We generalize Gaur's result to the k-ary case and slightly strengthen it by showing NP-completeness under the standard notion of polynomial-time many-one reductions:

Theorem 4.1. *For each $k \geqslant 2$, deciding whether a complete k-ary network N is minimal is NP-complete, even in case of bounded domain sizes.*

Proof. Membership in NP is easily seen: We just need to guess a candidate solution s_t from $sol(N)$ for each of the polynomially many tuples t of each constraint c of N, and check in polynomial time that s_t is effectively a solution and that the projection of s_t over $scope(c)$ yields t. For proving hardness, revisit the proof of Theorem 3.1. For each $k \geqslant 2$, from a 3SAT instance C, we there construct in polynomial time a highly symmetric k-ary network with bounded domain sizes N_{C^*}, such that N_{C^*} is minimal (i.e., $M_k(N_{C^*}) = N_{C^*}$) iff C is satisfiable. This is clearly a standard many-one reduction from 3SAT to network minimality. □

A result in database theory similar to Theorem 4.1 was shown in [19], where it was proven that determining whether a set of database relations is join consistent (i.e., admits a universal relation) is NP-complete. This was actually proven for sets of binary relations, however not over schema S_k. Here we showed that this also holds for *complete k-ary networks*, i.e., for sets of relations over the specific schemas S_k, for each $k \geqslant 2$.

4.2. Structure identification and k-representability

This section as well as Sections 4.3 and 4.4 are dedicated to the problem of representing single constraints (or single-constraint networks) through equivalent k-ary minimal networks. By a slight abuse of terminology, when there is no danger of confusion, we will often identify a single-constraint network $\{\rho\}$ with its unique constraint ρ, and for tuples t of the relation $rel(\rho)$ of the constraint ρ, we may write $t \in \rho$ instead of $t \in rel(\rho)$.

Definition 4.1. A complete k-ary network M is a *minimal k-ary network* of ρ iff

1. $sol(M) = \rho$, and
2. every tuple occurring in some constraint r of M is the projection of some tuple t of ρ over $scope(r)$.

We say that a constraint relation ρ is *k-representable* if there exists a (not necessarily complete) k-ary constraint network M such that $sol(M) = \rho$. The following proposition seems to be well known and follows very easily from Definition 4.1 anyway.

Proposition 4.2. *Let ρ be a constraint. The following three statements are equivalent:*

1. *ρ has a minimal k-ary network;*
2. *$sol(\Pi_{S_k}(\rho)) = \rho$;*
3. *ρ is k-representable.*

Note that the equivalence of ρ being k-representable and of ρ admitting a minimal k-ary network emphasizes the importance and usefulness of minimal networks. In a sense this equivalence means that the minimal k-ary network of ρ, if it exists, already represents all k-ary networks that are equivalent to ρ.

The complexity of deciding whether a minimal k-ary network for a relation ρ exists has been stated as an open problem by Dechter and Pearl in [9]. More precisely, Dechter and Pearl consider the equivalent problem of deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$ holds, and refer to this problem as a problem of *structure identification in relational data* [9]. The idea is to identify the class of relations ρ that have the structural property of being equivalent to the k-ary network $\Pi_{S_k}(\rho)$, and thus, of being k-representable. Dechter and Pearl formulated the following conjecture:

Conjecture 4.1. *(See Dechter and Pearl [9].) For each fixed positive integer $k \geqslant 2$, deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$ is NP-hard.*[3]

As already observed by Dechter and Pearl in [9], there is a close relationship between the k-representability of constraint relations and a relevant database problem. Let us briefly digress on this. It is common knowledge that a single constraint ρ can be identified with a *data relation* in the context of relational databases (cf. [8]). The decomposition of relations plays an important role in the database area, in particular in the context of normalization [24]. It consists of decomposing a relation ρ without loss of information into smaller relations whose natural join yields precisely ρ. If ρ is a concrete data relation (i.e., a relational instance), and S is a family of subsets (subschemas) of the schema of ρ, then the decomposition of ρ over S consists of the projection $\Pi_S = \{\Pi_s(\rho) \mid s \in S\}$ of ρ over all schemes in S. This decomposition is *lossless* iff the natural join of all $\Pi_s(\rho)$ yields precisely ρ, or, equivalently, iff ρ satisfies the *join dependency* $*[S]$. We can thus reformulate the concept of k-decomposability in terms of database theory as follows: A relation ρ is k-decomposable iff it satisfies the join dependency $*[S_k]$, i.e., iff the decomposition of ρ into schema S_k is lossless. The following complexity result was shown as early as 1981 in [25].[4]

Proposition 4.3. *(See Maier, Sagiv, and Yannakakis [25].) Given a relation ρ and a family S of subsets of the schema of ρ, it is coNP-complete to determine whether ρ satisfies the join dependency $*[S]$, or equivalently, whether the decomposition of ρ into schema S is lossless.*

Proposition 4.3 is weaker than Conjecture 4.1 and does not by itself imply it, nor so does its proof given in [25]. In fact, Conjecture 4.1 speaks about the very specific sets S_k for $k \geqslant 2$, which are neither mentioned in Proposition 4.3 nor used in its proof. Actually, the NP-hardness proof in [25] transforms 3SAT into the problem of checking a join dependency $*[S]$ over schema $S = (S_1, \ldots, S_{m+1})$, where one of the relation schemas, namely S_{m+1} is of unbounded arity (depending on the input 3SAT instance), while the others are of arity 4. To prove Conjecture 4.1, that refers to the specific schema S_k in which all relations have arity at most k, we thus needed to develop a new and independent hardness argument.

Theorem 4.2. *For each fixed integer $k \geqslant 2$, deciding for a single constraint ρ whether $sol(\Pi_{S_k}(\rho)) = \rho$, that is, whether ρ is k-decomposable, is coNP-complete.*

[3] Actually, the conjecture stated in [9] is somewhat weaker: Given a relation ρ and an integer k, deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$ is NP-hard. Thus k is not fixed and is part of the input instance. However, from the context and use of this conjecture in [9] it is clear that Dechter and Pearl actually intend NP-hardness for each fixed $k \geqslant 2$.

[4] As mentioned by Dechter and Pearl [9], Jeff Ullman has proved this result, too. In fact, Ullman, on a request by Judea Pearl, while not aware of the specific result in [25], has produced a totally independent proof in 1991, and sent it as a private communication to Pearl. The result is also implicit in Moshe Vardi's 1981 PhD thesis.

Proof. We show that deciding whether $sol(\Pi_{S_k}(\rho)) \neq \rho$ is NP-complete.

Membership. Membership in NP already follows from Proposition 4.3, but we give a short proof of it here for sake of self-containment. Clearly, $\rho \subseteq sol(\Pi_{S_k}(\rho))$. Thus $sol(\Pi_{S_k}(\rho)) \neq \rho$ iff the containment is proper, which means that there exists a tuple t_0 in $sol(\Pi_{S_k}(\rho))$ not contained in ρ. One can guess such a tuple t_0 in polynomial time and check in polynomial time that for each $\ell \leqslant k$, each ℓ-tuple of variables $X_{i_1}, \ldots, X_{i_\ell}$ of $var(\rho)$, the projection of t_0 to $(X_{i_1}, \ldots, X_{i_\ell})$ is indeed a tuple of the corresponding constraint of S_k. Thus determining whether $sol(\Pi_{S_k}(\rho)) \neq \rho$ is in NP.

Hardness. We first show hardness for the binary case, that is, the case where $k = 2$. We use the NP-hard problem 3COL of deciding whether a graph $G = (V, E)$ with set of vertices $V = \{v_1, \ldots, v_n\}$ and edge set E is 3-colorable. Let G be given as input instance. We assume without loss of generality that G has at least three vertices. Let r, g, b be three data values standing for the three colors red, green, and blue, respectively. Let $N_{3\mathrm{COL}}$ be the constraint network defined as follows. The set of variables $var(N_{3\mathrm{COL}}) = \{X_1, \ldots, X_n\}$. The schema S_2^+ of $N_{3\mathrm{COL}}$ consists of all exactly binary scopes (X_i, X_j) where $X_i \prec X_j$, and $dom(X_i) = \{r, g, b\}$ for $1 \leqslant i \leqslant n$. Moreover, for all $1 \leqslant i < j \leqslant n$, the constraint c_{ij} with schema (X_i, X_j) has the following constraint relation $rel(c_{ij}) = r_{ij}$: if $(i, j) \in E$, then r_{ij} is the set of pairs representing all legal vertex colorings, i.e., $r_{ij} = \{(r, g), (g, r), (r, b), (b, r), (g, b), (b, g)\}$; otherwise $r_{ij} = \{r, g, b\}^2$. $N_{3\mathrm{COL}}$ is thus a straightforward encoding of 3COL over schema S_2^+, and obviously G is 3-colorable iff $sol(N_{3\mathrm{COL}}) \neq \emptyset$. Thus, deciding whether $sol(N_{3\mathrm{COL}}) \neq \emptyset$ is NP-hard.

We construct from $N_{3\mathrm{COL}}$ a single constraint ρ with schema $\{X_1, \ldots, X_n\}$ as follows. The domain $dom(\rho)$ contains the color constants r, g, and b, as well as special "tuple identifiers" to be detailed below. For each constraint c_{ij} of $N_{3\mathrm{COL}}$, and for each tuple $(a, b) \in r_{ij}$, ρ contains a tuple t whose X_i and X_j values are a and b, respectively, and whose X_ℓ value, for all $1 \leqslant \ell \leqslant n$, $\ell \neq i$, $\ell \neq j$, is a constant d^t_{ij}, different from all values used in other tuples, whose purpose is to act as a tuple identifier. This concludes the description of the transformation from a 3COL instance $G = (V, E)$ to a constraint network $N_{3\mathrm{COL}}$ and further to a constraint ρ. Clearly, this transformation is feasible in polynomial time. We claim the following:

Claim. *$sol(\Pi_{S_2}(\rho)) \neq \rho$ iff $sol(N_{3\mathrm{COL}}) \neq \emptyset$ (and thus iff G is 3-colorable).*

This claim clearly implies the NP-hardness of deciding $sol(\Pi_{S_k}(\rho)) \neq \rho$. Let us prove that the claim holds.

We start with the *if* direction. Assume $sol(N_{3\mathrm{COL}}) \neq \emptyset$. Then $G = (V, E)$ is 3-colorable and hence there exists a function $f : V \longrightarrow \{r, g, b\}$ such that for each edge $\langle v_i, v_j \rangle \in E$, $f(v_i) \neq f(v_j)$. Let t be the tuple defined by $\forall 1 \leqslant i \leqslant n$, $t[X_i] = f(v_i)$. Then, by definition of t and ρ, for each $1 \leqslant i < j \leqslant n$, $t[X_i, X_j] \in \Pi_{X_i, X_j}(\rho)$ and fort each $1 \leqslant i \leqslant n$, $t[X_i] \in \Pi_{X_i}(\rho)$. Therefore, $t \in sol(\Pi_{S_2}(\rho))$. However, $t \notin \rho$, because each tuple of ρ, unlike t, has some tuple identifiers as components. It thus follows that $sol(\Pi_{S_2}(\rho)) \neq \rho$.

Let us now show the *only-if* direction of the claim. Assume $sol(\Pi_{S_2}(\rho)) \neq \rho$. Given that, as already noted, $\rho \subseteq sol(\Pi_{S_2}(\rho))$, there must exist a tuple $t_0 \in sol(\Pi_{S_2}(\rho))$ such that $t_0 \notin \rho$. We show that t_0 can contain values from $\{r, g, b\}$ only, and must, moreover, be a solution to $N_{3\mathrm{COL}}$. Assume a tuple identifier $d = d^t_{ij}$ occurs as a component of t_0. By construction of ρ, d occurs in precisely one single tuple t of ρ. It follows that each relation of $\Pi_{S_2}(\rho)$ has at most one tuple containing d, and therefore the join of all relations of $\Pi_{S_2}(\rho)$ contains a single tuple only in which d occurs as data value, namely t itself. Therefore, $t_0 = t$, and hence $t_0 \in \rho$, which contradicts our assumption that $t_0 \notin \rho$. We have thus shown that t_0 cannot contain any tuple identifier at all, and can be made of "color elements" from $\{r, g, b\}$ only. However, by definition of ρ, each tuple $t_{ij} \in \{r, g, b\}^2$ occurring in a relation with schema (X_i, X_j) of $\Pi_{S_2}(\rho)$ also occurs in the corresponding relation of $N_{3\mathrm{COL}}$, and vice versa. Thus $sol(\Pi_{S_2}(\rho)) \neq \rho$ iff $sol(N_{3\mathrm{COL}}) \neq \emptyset$ iff G is 3-colorable, which proves our claim.

For each fixed $k > 2$ we can apply exactly the same line of reasoning. We define $N^k_{k\mathrm{COL}}$ as the complete network on variables $\{X_1, \ldots, X_n\}$ of all k-ary correct "coloring" constraints, where the relation with schema $X_{i_1}, \ldots, X_{i_k}$ expresses the correct colorings of vertices $v_{i_1}, \ldots, v_{i_k}$ of graph G. We then define ρ in a similar way as for $k = 2$: each k-tuple of a relation of $N^k_{k\mathrm{COL}}$ is extended by use of (possibly multiple occurrences of) a tuple identifier to an n-tuple of ρ. Given that k is fixed, ρ can be constructed in polynomial time, and so $\Pi_{S_k}(\rho)$. It is readily seen that each tuple of $sol(\Pi_{S_k}(\rho))$ that contains a tuple identifier is already present in ρ because for each tuple identifier value d, each relation of $\Pi_{S_k}(\rho)$ contains at most one tuple involving d. Hence, any tuple in $sol(\Pi_{S_k}(\rho)) - \rho$ involves values from $\{r, g, b\}$ only, and is a solution to $N^k_{k\mathrm{COL}}$ and thus a valid 3-coloring of G. $\square$

4.3. The case of bi-valued relations

Let us now turn our attention to bi-valued relations ρ, that is, relations ρ over a binary domain. As explained in Section 3.2 of [9], such bi-valued relations are of special interest, as that they correspond to Boolean formulas. For example, a 3CNF can be seen as a bi-valued constraint network of ternary relations, and a single bi-valued relation ρ corresponds to a DNF. The problem of structure identification in the bi-valued case thus corresponds to relevant identification and learnability questions about Boolean formulas; we refer the reader to [9] for details. In this context, it would be interesting to know whether, or for which parameter k, Theorem 4.2 carries over to the bi-valued case. While the coNP-membership clearly applies to the special case of a bi-valued ρ, the hardness part of that proof uses a multiple-valued relation ρ and does not allow us to derive a hardness result for the bi-valued case. In fact, the relations ρ constructed in the proof of Theorem 4.2 from arbitrary 3COL instances are not bi-valued and actually have unbounded domains $dom(\rho)$ containing $|\rho| + 3$ elements[5]: the "color constants" r, g, b, and the $|\rho|$ tuple identifiers $d^t_{i,j}$.

As noted in [7,11], for $k = 2$ and bi-valued domains, the problem of deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$ is tractable. It can actually be reduced to 2SAT. But what about for values $k \geqslant 3$? Dechter and Pearl made the following conjecture (Conjecture 3.27 in [9]):

Conjecture 4.2. *(See Dechter and Pearl [9].) For each fixed positive integer $k \geqslant 3$, deciding for a bi-valued relation ρ whether $sol(\Pi_{S_k}(\rho)) = \rho$ is NP-hard.*[6]

We are able to confirm this conjecture.

Theorem 4.3. *For each fixed integer $k \geqslant 3$, deciding for a single bi-valued constraint ρ whether $sol(\Pi_{S_k}(\rho)) = \rho$, that is, whether ρ is k-decomposable, is coNP-complete.*

The rather involved proof of this theorem is given in Appendix B. The hardness part is similar in spirit to the one of Theorem 4.2, except for two important changes that are due to the requirement of a two-valued domain. First we encode 3SAT rather than 3COL, in order to achieve a binary domain. However, there is still the problem of the tuple identifiers (the values $d^t_{i,j}$ in the proof of Theorem 4.2). They are values from an unbounded domain. We therefore use a specific bit-vector encoding that allows us to represent tuple identifiers in binary format. This is, however, not totally trivial. The difficulty lies in the fact that in the relations of the projection $\Pi_{S_k}(\rho)$ we do no longer have full bit vectors at our disposal, but only k-bit projections of such bit vectors. Sophisticated coding tricks are used for coping with this problem, and for obtaining a correct reduction.

Theorem 4.3 has a corollary, which we here formulate in the terminology of Dechter and Pearl [9].

Corollary 4.1. *For fixed $k \geqslant 3$, the class of k-CNFs is not identifiable relative to all CNFs (unless $P = NP$).*

The above means the following. If a CNF ϕ (or, more generally, a Boolean function ϕ) is given by the set of all its models (i.e., by a bi-valued relation, each tuple of which corresponds to a model), then it is NP-hard to decide whether ϕ is equivalent to a k-CNF. We refer the reader to [9] for a more detailed account of k-CNF identification and its equivalence to the problem of whether a bi-valued relation ρ is k-decomposable. To conclude this topic, let us note that the representation of a Boolean function ϕ by the explicit set of all its models, i.e., by all satisfying truth value assignments, is also known as the *onset* of ϕ [32]. The above corollary thus states that, for fixed $k \geqslant 3$, it is NP-hard to decide whether a Boolean function specified by its onset is equivalent to a k-CNF.[7]

4.4. Further strengthening and tractability frontier

The technique used to prove Theorem 4.3 can be used to strengthen Theorem 4.2, and to show that it actually also holds for tri-valued constraints ρ.

Theorem 4.4. *For each fixed integer $k \geqslant 2$, deciding for a single tri-valued constraint ρ whether $sol(\Pi_{S_k}(\rho)) = \rho$, that is, whether ρ is k-decomposable, is coNP-complete.*

The proof of this theorem is given in Appendix C. We there use a transformation from 3COL from a graph $G = (V, E)$ as described in the proof of Theorem 4.2 by applying, in addition, similar vectorization techniques as in the proof of Theorem 4.3.

The above result, together with Theorem 4.3, and with the fact that the 2-representability of binary networks is feasible in polynomial time (see [7]), and with the facts that the 0-representability and 1-representability of each network and the k-representability of 1-valued networks are trivially tractable, gives us the following precise characterization of the tractability of deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$:

Theorem 4.5. *For the class of i-valued relations ρ, deciding $sol(\Pi_{S_k}(\rho)) = \rho$ is tractable iff $i = 1$ or ($i = 2$ and $k \leqslant 2$). In all other cases, the problem is coNP-complete.*

Fig. 3 illustrates this tractability frontier.

---

[5] Here $|\rho| = |rel(\rho)|$ designates the number of tuples in the constraint relation of the constraint ρ.

[6] Note that in [9], the parameter k is not explicitly required to be fixed, however, from the context it is clear that the present stronger version of the conjecture was actually intended. Moreover, Conjecture 3.27 in [9] was formulated in terms of k-CNFs rather than in a purely relational setting. To avoid additional definitions and terminology, we have restated an equivalent relational formulation here. In particular, we have replaced the term $M(\Gamma_{S_k}(\rho))$ in the original formulation by the equivalent term $sol(\Pi_{S_k}(\rho))$.

[7] While we have not found this result in the literature on Boolean functions, we cannot totally exclude that it has been independently derived, maybe in a different context or using a different formalism.

	$k \leq 1$	$k = 2$	$k \geq 3$
$i \geq 3$	tractable	coNP-complete	coNP-complete
$i = 2$	tractable	tractable	coNP-complete
$i \leq 1$	tractable	tractable	tractable

Fig. 3. Tractability Frontier for the k-decomposability of i-valued relations ρ.

5. Summary, discussion, and future research

In this paper we have tackled and solved long standing complexity problems related to minimal constraint networks:

- We solved an open problem posed by Gaur [11] in 1995, and later by Dechter [8], by proving Dechter's conjecture and showing that computing a solution to a minimal constraint network is NP-hard.
- We proved a conjecture on structure identification in relational data made in 1992 by Dechter and Pearl [9]. In particular, we showed that for $k \geqslant 2$, it is coNP-complete to decide whether for a single constraint (or data relation) ρ, $sol(\Pi_{S_k}(\rho)) = \rho$, and thus whether ρ is k-decomposable.
- We also proved a refined conjecture of Dechter and Pearl [9], showing that the above problem remains coNP-hard even if ρ is a bi-valued constraint, in case $k \geqslant 3$. A consequence of this is the NP-hardness of identifying k-CNFs relative to the class of all CNFs (when represented by the explicit enumeration of their models).
- We finally proved that deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$ is coNP-complete for tri-valued relations and $k \geqslant 2$. Together with our other results on structure identification, this allowed us to trace the precise tractability frontier for this problem.

We wish to make clear that our hardness result about computing solutions to minimal networks does not mean that we think minimal networks are useless. To the contrary, we are convinced that network minimality is a most desirable property when a solution space needs to be efficiently represented for applications such as computer-supported configuration [10]. For example, a user interactively configuring a PC constrains a relatively small number of variables, say, by specifying a maximum price, a minimum CPU clock rate, and the desired hard disk type and capacity. The user then wants to quickly know whether a solution exists, and if so, wants to see it. For a k-ary minimal constraint network, the satisfiability of queries involving k variables only can be decided in polynomial time. However, our Theorem 3.1 states that, unless NP = P, in case the query is satisfiable, there is no way to witness the satisfiability by a complete solution (in our example, by exhibiting a completely configured PC satisfying the user requests).

Our Theorem 3.1 thus unveils a certain deficiency of minimal networks, namely, the failure of being able to exhibit full solutions. However, we have a strikingly simple proposal for redressing this deficiency. Rather than just storing ℓ-tuples (where $\ell \leqslant k$) in a k-ary minimal network $M_k(N)$, we may store a full solution t^+ with each ℓ-tuple, where t^+ coincides with t on the ℓ variables of t. Call this extended minimal network $M_k^+(N)$. Complexity-wise, $M_k^+(N)$ is not harder to obtain than $M_k(N)$. Moreover, in practical terms, given that the known algorithms for computing $M_k(N)$ from N require to check for each ℓ-tuple t whether it occurs in some solution t^+, why not just memorize t^+ on the fly for each "good" tuple t? Note also that the size of $M_k^+(N)$ is still polynomial, and at most by a factor $|var(N)|$ larger than the size of $M_k(N)$. One may even go further and store not just a single solution, but the K *best* solutions (according to some predefined preference ordering) whose values coincide with those of t with each tuple t of $M_k(N)$. This allows one to answer *top K queries* with at most k variables in polynomial time, once $M_k(N)$ has been compiled. An example would be: *show me the* 5 *cheapest laptops fulfilling* ϕ, where ϕ constrains k variables only.

For practical applications it is not always optimal to consider the complete schemas S_k as defined here. In the conference version [17] of this paper, S_k was defined to contain only all *exactly* k-ary relations over a given set of variables, rather than all *at most* k-ary relations. It is easy to see that the relations with scopes of fewer than k variables are redundant and can indeed be omitted (they can always be obtained via projections from the exactly k-ary relations). The only reason why we used the complete schemas in the present journal version is that we wanted to use exactly the same definition as in the standard references [26,8]. However, yet more liberal definitions are possible. For example, Lecoutre [22] defines a constraint network N over an arbitrary schema S to be minimal if $N = \Pi_S(sol(N))$. Clearly, all our complexity bounds carry over to this more liberal setting: the lower bounds are directly inherited, as all instances in our settings are also instances of the more liberal setting, and the upper bounds are obtained by a trivial adaptation of the proofs of our existing upper bounds.

An interesting problem for future research is the following. We may issue queries of the following form against $M_k^+(N)$: `SELECT A SOLUTION WHERE` ϕ. Here ϕ is some Boolean combination on constraints on the variables of N. Queries, where ϕ is a simple combination of range restrictions on k variables can be answered in polynomial time. But there are much more complicated queries that can be answered efficiently, for example, queries that involve aggregate functions and/or re-use of quantified variables. It would thus be nice and useful to identify very large classes of queries to $M_k^+(N)$ for which a single solution – if it exists – can be found in polynomial time.

Another relevant research problem is related to the projection $\Pi_{S^*}(sol(N))$ of the solution $sol(N)$ of a (not necessarily binary) constraint network N to a user-defined schema S^*, and to the further use of the schema S^* for distributed constraint solving. The projection of the solution space to specific sets of variables is used in the context of system configuration, when a system is jointly configured by a number of engineers, each having access to a projection of the solution space only [33]. The problem of computing a solution $\Pi_{S^*}(sol(N))$ is generally NP-hard, and remains NP-hard in many special cases, e.g. if N is binary and, at the same time $S^* = S_2$ (see Theorem 3.1). We would like to investigate relevant restrictions that make this problem tractable. For some restrictions, this is already known. For example, if S^* has bounded hypertree width [16,15,3], then this problem becomes tractable. If S^* has bounded hinge width, then computing a solution can even be done in a backtrack-free manner, see Section 3 of [18]. Note that bounded hinge width is a stronger restriction than bounded hypertree width; for a comparison of these and other hypergraph restrictions, see [14]. Other decompositions that lead to backtrack solution search are the *world-set decompositions* discussed in [28] and further generalized in [29]. These decompositions are based on Cartesian products rather than on joins, therefore, computing solutions is easier than with project-join decompositions.

There is also the problem of computing the desired projections without computing the possibly very large relation $sol(N)$, and, as a special case, computing the minimal constraint network $M(N)$ from a given network N. More formally, we would like to compute $\Pi_{S^*}(sol(N))$ from N in polynomial space as efficiently as possible, assuming the relations of S^* are all of bounded arity. There are already promising approaches to this problem in the literature. In [12,13], conflict-driven answer set programming (ASP) techniques are used for this task. In [33], projections of $sol(N)$ are computed via a SAT solver, and it is shown that this method is feasible for large datasets stemming from the automotive industry. However, we expect that a structural analysis of the original network N and of the desired projection schema S^* could further help to speed up this computation.

Acknowledgements

This research was originally stimulated by discussions on various aspects of constraint solving and pruning with Donald Knuth, to whom this paper is dedicated with my warmest congratulations on his 75th birthday.

Work funded by EPSRC Grant EP/G055114/1 “Constraint Satisfaction for Configuration: Logical Fundamentals, Algorithms, and Complexity”. The author is a James Martin Senior Research Fellow. He thanks V. Bárány, C. Bessiere, D. Cohen, R. Dechter, D. Gaur, J. Petke, M. Vardi, M. Yannakakis, S. Živný, and the referees of both the conference and the journal version for useful comments and/or pointers to earlier work.

Appendix A. Proof of the Symmetry Lemma

Lemma 3.1 *(Symmetry Lemma). For each fixed integer $k \geqslant 1$, there is a polynomial-time transformation T that transforms each 3SAT instance C into a k-supersymmetric instance C^* such that C is satisfiable iff C^* is satisfiable.*

Proof. We first prove the lemma for $k = 2$. Consider the given 3SAT instance C. Create for each propositional variable $p \in propvar(C)$ a set $New(p) = \{p_1, p_2, p_3, p_4, p_5\}$ of fresh propositional variables. Let $Disj^+(p)$ be the set of all disjunctions of three distinct positive atoms from $New(p)$ and let $Disj^-(p)$ be the set of all disjunctions of three distinct negative literals corresponding to atoms in $New(p)$. Thus, for example $(p_2 \vee p_4 \vee p_5) \in Disj^+(p)$ and $(\overline{p_1} \vee \overline{p_4} \vee \overline{p_5}) \in Disj^-(p)$. Note that $Disj^+(p)$ and $Disj^-(p)$ each have exactly $\binom{5}{3} = 10$ elements (we do not distinguish between syntactic variants of equivalent clauses containing the same literals).

Consider the following transformation T, which eliminates all original literals from C, yielding C^*:

Function T:
BEGIN $C' := C$.
WHILE $propvar(C) \cap propvar(C') \neq \emptyset$ DO
  {pick any $p \in propvar(C) \cap propvar(C')$; $C' := elim(C', p)$};
Output(C')
END.

Here $elim(C', p)$ is obtained from C' and p as follows:

FOR each clause K of C' in which p occurs positively or negatively DO

BEGIN
let δ be the disjunction of all literals in K different from p and from $\neg p$[8];
if p occurs positively in K, replace K in C' by the conjunction $\Gamma^+(K)$ of all clauses of the form $\alpha \vee \delta$, where $\alpha \in Disj^+(p)$;
if p occurs negatively in K, replace K in C' by the conjunction $\Gamma^-(K)$ of all clauses of the form $\alpha \vee \delta$, where $\alpha \in Disj^-(p)$;
END.

[8] An empty δ is equal to *false*, and it is understood that $\alpha \vee false$ is simply α.

Let $C^* = T(C)$ be the final result of T. C^* contains no original variable from *propvar*(C). Note that C^* can be computed in polynomial time from C. In fact, note that every clause of three literals of C gives rise to exactly $\binom{5}{3}^3 = 10^3 = 1000$ clauses of 9 literals each in C^*. While computing C^* from C, we can thus replace each three-literal clause of C at once and independently by the corresponding 1000 clauses. Similar direct replacements (but with fewer result clauses) are, of course, possible for two-literal and one-literal clauses of C. Assuming appropriate data structures, the transformation from C to C^* can thus actually be done in linear time.

We now need to prove (1) that C^* is satisfiable iff C is and (2) that C^* is 2-supersymmetric.

Fact 1: *C^* is satisfiable iff C is.* We will actually prove more than we need here. In fact, our proof of Fact 1 below also shows that a satisfying assignment to C can be transformed into *many* satisfying assignments to C^*. We will use this, when we come to prove supersymmetry in Fact 2. We prove Fact 1 by showing that, when at each step of algorithm T, C' is transformed into its next value $C'' = elim(C', p)$, then C' and C'' are satisfaction-equivalent. Fact 1 then follows by induction. Assume C' is satisfied via a truth value assignment τ'. Then let τ'' be any truth value assignment to the propositional variables of C'' with the following properties:

- for each propositional variable q of C'' different from p, $\tau''(q) = \tau'(q)$,
- if $\tau'(p) = true$, then at least 3 of the variables in *New*(p) are set true by τ'', and
- if $\tau'(p) = false$, then at most two of the variables in *New*(p) are set true by τ'' (and at least three are thus set false).

By definition of C'', τ'' must satisfy C''. In fact, assume first $\tau'(p) = true$. Let K be a clause of C in which p occurs positively. Then, given that at least three variables in *New*(p) are set true by τ'', each element of *Disj*$^+(p)$ must have at least one atom made true by τ'', and thus each of the clauses of $\Gamma^+(K)$ of C'' evaluates to true via τ''. All other clauses of C'' stem from clauses of C' that were made true by literals corresponding to an atom q different from p. But, by definition of τ, these literals keep their truth values, and hence make the clauses true. In summary, all clauses of C'' are satisfied by τ''. In a very similar way it is shown that τ'' satisfies C'' if, $\tau'(p) = false$.

Vice versa, assume some truth value assignment τ'' satisfies C''. Then it is not hard to see that C' must be satisfied by the truth value assignment τ' to C' defined as follows: If a majority (i.e. 3 or more) of the five atoms in *New*(p) are made true via τ'', then let $\tau'(p) = true$, otherwise let $\tau'(p) = false$; moreover, for all propositional variables $q \notin New(p)$, let $\tau'(q) = \tau''(q)$.

To see that τ' satisfies C', consider first the case that three or more of the propositional variables of *New*(p) are assigned *true* by τ''. Note that all clauses of C' that neither contain p nor $\overline{p}$ are trivially satisfied by τ', as τ' and τ'' coincide on their atoms. Now let us consider any clause K of C' in which p occurs positively. Then the only clauses that contain positive occurrences of elements of *New*(p) of C'' are the sets $\Gamma^+(K)$. If τ'' is such that it makes at least three of the five atoms in *New*(p) true, then any clause in $\Gamma^+(K)$ is made true by atoms of *New*(p). Thus when replacing these atoms by p and assigning p true, the resulting clause K remains true. Now consider a clause $K = \overline{p} \vee \delta$ of C' in which p occurs negatively. The only clauses containing negative *New*(p)-literals in C'' are, by definition of C'', those in $\Gamma^-(K)$. Recall we assumed that τ'' satisfies at least three distinct atoms from *New*(p). Let three of these satisfied atoms be p_i, p_j, and p_k. By definition, $\Gamma^-(K)$ contains a clause of the form $\overline{p_i} \vee \overline{p_j} \vee \overline{p_k} \vee \delta$. Given that this clause is satisfied by τ'', but τ'' falsifies $\overline{p_i} \vee \overline{p_j} \vee \overline{p_k}$, δ is satisfied by τ'', and since δ contains no *New*(p)-literals, δ is also satisfied by τ'. Therefore, $K = \overline{p} \vee \delta$ is satisfied by τ'. This concludes the case where three or more of the propositional variables of *New*(p) are assigned *true* by τ''. The case where three or more of the propositional variables of *New*(p) are assigned *false* by τ'' is completely symmetric, and can thus be settled in a totally similar way. □

Fact 2: *Proof that C^* is 2-supersymmetric*. Assume C^* is satisfiable by some truth value assignment η. Then C is satisfiable by some truth value assignment τ, and thus C^* is satisfiable by some truth value assignment τ^* constructed inductively as described in the proof of Fact 1. Let us have a closer look at the inductive construction used to obtain τ^* in Fact 1. For any initially fixed pair of propositional variables $p_i, q_j \in propvar(C^*)$, where $1 \leqslant i, j \leqslant 5$, the construction of τ^* gives us a very large degree of freedom for choosing τ^*. Actually, the construction is so general, that it allows us to let p_i, q_j take on any arbitrary truth value assignment among of the four possible joint truth value assignments. In fact, however we choose the truth value assignments for two among the variables in $\{p_1, \ldots, p_5, q_1, \ldots, q_5\}$, there is always enough flexibility for assigning the remaining variables in this set some truth values that ensure that the majority of variables has the truth value required by the proof of Fact 1 for representing the original truth value of p via τ'. (This holds even in case p and q are one and the same variable, and we thus want to force two elements from $\{p_1, \ldots, p_5\}$ to take on some truth values, see the second example below.) Let us give two examples that illustrate the two characteristic cases to consider. First, assume p and q are distinct and τ satisfies p and falsifies q. We would like to construct, for example, a truth value assignment τ^* that falsifies p_2 and simultaneously satisfies q_4. In constructing τ^*, the only requirements on *New*(p) and *New*(q) are that more than three variables from *New*(p) need to be satisfied by τ^*, but no more than two from *New*(q) need to be satisfied by τ^*. For instance, we may then set $\tau^*(p_1) = \tau^*(p_3) = \tau^*(p_4) = \tau^*(p_5) = true$ and $\tau^*(p_2) = false$ and $\tau^*(q_1) = \tau^*(q_2) = \tau^*(q_3) = \tau^*(q_5) = false$ and $\tau^*(q_4) = true$. This achieves the desired truth value assignment to p_2 and q_4. An extension to a full satisfying truth value assignment τ^* for C^* is guaranteed. Now, as a second example, assume that $\tau(p) = false$, but we would like $\tau(p_1)$ and $\tau(p_2)$ to be simultaneously true in a truth value assignment satisfying C^*. Note that in this case, the only requirement on $New(p)$ in the construction of τ^* is that at most two atoms from $New(p)$ must be assigned *true*. Here we have a single option only: set $\tau^*(p_1) = \tau^*(p_2) = true$ and $\tau^*(p_3) = \tau^*(p_4) = \tau^*(p_5) = false$. This option works perfectly, and assigns the desired truth values to p_1 and p_2. In summary, C^* is 2-supersymmetric. □

The proof for $k > 2$ is totally analogous, except for the following modifications:

- Instead of creating for each propositional variable $p \in propvar(C)$ a set $New(p) = \{p_1, p_2, \ldots, p_5\}$ of five new variables, we now create a set $New(p) = \{p_1, p_2, \ldots, p_{2k+1}\}$ of $2k + 1$ new propositional variables.
- The set $Disj^+(p)$ is now defined as the set of all disjunctions of $k + 1$ positive atoms from $New(p)$. Similarly, $Disj^-(p)$ is now defined as the set of all disjunctions of $k + 1$ negative literals obtained by negating atoms from $New(p)$.
- We replace the numbers 2 and 3 by k and $k + 1$, respectively.
- We note that now each three-literal clause of C is replaced no longer by $\binom{5}{3}^3$ clauses but by $\binom{2k+1}{k+1}^3$ clauses.
- We note that the resulting clause set C^* is now a $3(k + 1)$-SAT instance.

It is easy to see that the proofs of Fact 1 and Fact 2 above go through with these modifications.

Finally, let us recall that any 2-supersymmetric SAT instance is trivially also 1-supersymmetric, which settles the theorem for $k = 1$. □

Appendix B. Proof of Theorem 4.3

Theorem 4.3. *For each fixed integer $k \geqslant 3$, deciding for a single bi-valued constraint ρ whether $sol(\Pi_{S_k}(\rho)) = \rho$, that is, whether ρ is k-decomposable, is coNP-complete.*

Proof. It suffices to show coNP-hardness, as membership in coNP already follows from Theorem 4.2. We first prove coNP-hardness for the case $k = 3$.

Consider a non-empty 3SAT instance $C = \{C_1, \ldots, C_m\}$ over a set $propvar(C) = \{p_1, \ldots, p_n\}$ of propositional variables, where each C_i is a clause containing precisely 3 literals whose corresponding atoms are mutually distinct.

Let us first define two numbers r_0 and r from C, whose meaning and use will become clear later on. Let r_0 denote the number of 3-element sets $\{p_a, p_b, p_c\}$ of mutually distinct propositional variables $p_a, p_b, p_c \in propvar(C)$ that do not all three jointly appear in any clause of C. Note that $r_0 \leqslant 8\binom{n}{3}$. Let, moreover, $r = 7m + r_0$. Clearly, r is polynomially bounded in the size of C, as $r \leqslant 7m + 8\binom{n}{3}$.

We construct in polynomial time a bi-valued constraint ρ of r elements, such that $sol(\Pi_{S_k}(\rho)) \neq \rho$ iff C is satisfiable. The scope $scope(\rho)$ of ρ contains for each $p_i \in propvar(C)$ a list of $r + 1$ variables $X_i^0, X_i^1, \ldots, X_i^r$. Intuitively, in each tuple t of $rel(\rho)$, for each $1 \leqslant i \leqslant n$, the values assigned to the variables $X_i^0, X_i^1, \ldots, X_i^r$ either shall encode a truth value assignment to p_i, in which case all variables of this list will be assigned the same value, zero or one, or these values shall encode a tuple identifier for the tuple in which they occur. A tuple identifier for the sth tuple of $rel(\rho)$ assigns the value zero to all X_i^j where $j \leqslant s$ and the value one to all X_i^j where $j \geqslant s$. This will be made more formal below.

The constraint relation $rel(\rho)$ consists of two groups of tuples:

Clause-induced tuples These are $7m$ tuples, namely, seven for each clause C_h, $1 \leqslant h \leqslant m$. These tuples are numbered from 1 to $7m$. Each of these tuples describes one of the 7 legal truth value assignments (out of 8 possible) to the three propositional variables of a clause $C_h \in C$. For each clause C_h, $1 \leqslant h \leqslant m$, and each truth value assignment $\tau_j \in propvar(C_h) \longrightarrow \{0, 1\}$, among all 7 permitted truth value assignments to the propositional variables of C_h, where $1 \leqslant j \leqslant 7$, $rel(\rho)$ contains precisely one tuple t_h^j, whose components are described as follows. For each $p_i \in propvar(C_h)$, $t_h^j[X_i^0] = t_h^j[X_i^1] = t_h^j[X_i^2] = \cdots = t_h^j[X_i^r] = \tau_j(p_i)$. Moreover, for each $p_i \in propvar(C) - propvar(C_h)$, $t_h^j[X_i^0] = 0$, and the assignments to $X_i^1 \ldots X_i^r$ jointly constitute a unique tuple identifier that exclusively appears in the tuple t_h^j, and that encodes the tuple number s of the tuple t_h^j (namely $s = 7(h-1) + j$) in a very simple way: It assigns 0 to all $X_i^{s'}$ where $0 \leqslant s' < s$ and 1 to all variables $X_i^{s'}$ where $s \leqslant s' \leqslant r$.

Auxiliary tuples These are no more than $8\binom{n}{3}$ tuples: one for each 3-element set $\{p_a, p_b, p_c\}$ of mutually distinct propositional variables $p_a, p_b, p_c \in propvar(C)$ that do not all three jointly appear in any clause of C. These auxiliary tuples are numbered from $7m + 1$ to r, where $r \leqslant 7m + 8\binom{n}{3}$ is the total number of tuples in ρ. Essentially, the eight auxiliary tuples associated with the above sets $\{p_a, p_b, p_c\}$ each encode one of the eight truth value assignments $\sigma_1, \ldots, \sigma_8$ to the propositional variables p_a, p_b, and p_c. These tuples thus do not encode effective constraints, as they reflect any arbitrary truth value assignment p_a, p_b, and p_c, but they will be needed for technical reasons. More formally, for each set $S = \{p_a, p_b, p_c\}$ as above, and each truth value assignment σ to $\{p_a, p_b, p_c\}$, $rel(\rho)$ contains a tuple t_S^σ, whose components are described as follows. For each $p_i \in S$, $t_S^\sigma[X_i^0] = t_S^\sigma[X_i^1] = t_s^\sigma[X_i^2] = \cdots = t_S^\sigma[X_i^r] = \sigma(p_i)$. Moreover, for each $p_i \in \mathit{propvar}(C) - S$, $t_S^\sigma[X_i^0] = 0$, and the assignments to $X_i^1 \dots X_i^r$, just as before, jointly constitute a unique tuple identifier that exclusively appears in the tuple t_S^σ, and that encodes the tuple number s of the tuple t_S^σ by assigning 0 to all $X_i^{s'}$ where $s' < s$ and 1 to all variables $X_i^{s'}$ where $s' \geqslant s$.

This concludes the definition of ρ.

Claim. *$\mathit{sol}(\Pi_{S_3}(\rho)) \neq \rho$ iff C is satisfiable.*

We first prove the *if-part* of the claim. Assume C is satisfiable. Thus there exists a truth value assignment τ to $\mathit{propvar}(C)$ satisfying C. We show that $\mathit{sol}(\Pi_{S_3}(\rho))$ then must contain the tuple $t \notin \mathit{rel}(\rho)$ defined as follows. For each $1 \leqslant i \leqslant n$, $t[X_i^0] = t[X_i^1] = t[X_i^2] = \cdots = t[X_i^r] = \tau(p_i)$. To see this, it suffices to observe that the projection $t[S]$ of t to any set $S = \{X_a^u, X_b^v, X_c^w\}$ of three distinct variables from $\mathit{scope}(\rho)$ is contained in the corresponding relation $\Pi_S(\rho)$ of $\Pi_{S_3}(\rho)$.

In fact, if the atoms p_a, p_b and p_c jointly occur in a clause C_h of C, then, the tuple t' in $\mathit{rel}(\rho)$ induced by C_h for truth value assignment $\tau[p_a, p_b, p_c]$ coincides in its S-components with the tuple t, in other terms, $t'[S] = t[S]$. Hence $t[S]$ is contained in the relation $\Pi_S(\rho)$ of $\Pi_{S_3}(\rho)$. Moreover, in case p_a, p_b and p_c do not jointly occur in a clause of C, then there must exist an auxiliary tuple t' such that $t'[S] = t[S]$, and thus, again, $t[S]$ is contained in the relation $\Pi_S(\rho)$ of $\Pi_{S_3}(\rho)$. In summary, t is contained in the join of the exactly ternary relations of $\Pi_{S_3}(\rho)$. As is easily verified, the binary and unary relations of $\Pi_{S_3}(\rho)$ are weaker than the ternary ones, and actually redundant; the join of all constraints with precisely three variables is in fact equal to $\mathit{sol}(\Pi_{S_3}(\rho))$. It follows that t is contained in the join of $\Pi_{S_3}(\rho)$, which is $\mathit{sol}(\Pi_{S_3}(\rho))$. However, t is not in $\mathit{rel}(\rho)$ because t does not contain any tuple identifier, whereas each tuple of $\mathit{rel}(\rho)$ does.

It now remains to show that, whenever $\mathit{sol}(\Pi_{S_3}(\rho))$ contains a tuple $t \notin \mathit{rel}(\rho)$, then t corresponds to a satisfying truth value assignment for C, and C is thus satisfiable. Let t be such a tuple. We first show that for each $1 \leqslant i \leqslant n$ and each $1 \leqslant v \leqslant r$ and $1 \leqslant w \leqslant r$ it must hold that $t[X_i^v] = t[X_i^w]$, thus all bits of $t[X_i^0, X_i^1, \dots, X_i^r]$ must be equal. We prove this by showing that this bit-vector cannot have two consecutive bits of different value.

- Assume that for some $0 < \ell \leqslant r$, $t[X_i^{\ell-1}] = 0$ while $t[X_i^\ell] = 1$. By construction, $\mathit{rel}(\rho)$ contains only a single tuple t' for which $t'[X_i^{\ell-1}] = 0$ but $t'[X_i^\ell] = 1$, namely the tuple numbered ℓ. Therefore, in each relation $\mathit{rel}(c)$ of any constraint c of $\Pi_{S_3}(\rho)$ where $\mathit{scope}(c)$ contains $X_i^{\ell-1}$, X_i^ℓ and any other variable X_j^u, there is thus a single tuple f_c having $f_c[X_i^{\ell-1}] = 0$ and $f_c[X_i^\ell] = 1$. It follows that $\mathit{sol}(\Pi_{S_3}(\rho))$ contains a unique tuple whose $X_i^{\ell-1}$-value is zero and whose X_i^ℓ-value is one, namely the tuple t'. Therefore $t = t'$, which contradicts our assumption that $t \notin \mathit{rel}(\rho)$.
- Assume that for some $0 < \ell \leqslant r$, $t[X_i^{\ell-1}] = 1$ while $t[X_i^\ell] = 0$. Observe that, by construction, $\mathit{rel}(\rho)$ does not contain a single tuple t' for which $t'[X_i^{\ell-1}] = 1$ while $t'[X_i^\ell] = 0$. In fact, $\mathit{rel}(\rho)$ was carefully constructed so that the bit values in the sequences $t'[X_i^0, X_i^1, \dots, X_i^r]$ never decrease in any of its tuples. Therefore, in no relation $\mathit{rel}(c)$ of any constraint c of $\Pi_{S_3}(\rho)$ where $\mathit{scope}(c)$ contains $X_i^{\ell-1}$, X_i^ℓ and any other variable X_j^u, there is thus a tuple f having $f[X_i^{\ell-1}] = 1$ and $f[X_i^\ell] = 0$. It follows that the join $\mathit{sol}(\Pi_{S_3}(\rho))$ contains no tuple whose $X_i^{\ell-1}$-value is one and whose X_i^ℓ-value is zero. Contradiction.

We have thus established that for $1 \leqslant i \leqslant n$, all bits of $t[X_i^0, X_i^1, \dots, X_i^r]$ must be equal. Let τ be the truth value assignment that for $1 \leqslant i \leqslant n$ associates to each p_i the truth value $t[X_i^0] = t[X_i^1] = \cdots = t[X_i^r]$. Let C_h be any clause of C. Let the atoms of C_h be p_a, p_b and p_c. Define

$$X_{(a)} := X_a^0 \quad \text{if } \tau(p_a) = 1 \quad \text{and} \quad X_{(a)} := X_a^r \quad \text{if } \tau(p_a) = 0;$$
$$X_{(b)} := X_b^0 \quad \text{if } \tau(p_b) = 1 \quad \text{and} \quad X_{(b)} := X_b^r \quad \text{if } \tau(p_b) = 0;$$
$$X_{(c)} := X_c^0 \quad \text{if } \tau(p_c) = 1 \quad \text{and} \quad X_{(c)} := X_c^r, \quad \text{if } \tau(p_c) = 0.$$

Consider the constraint q of $\Pi_{S_3}(\rho)$ having $\langle X_{(a)}, X_{(b)}, X_{(c)} \rangle$ as scope. This constraint must have a tuple $t_q = \langle \tau(p_a), \tau(p_b), \tau(p_c) \rangle$, which is obviously identical to $t[X_{(a)}, X_{(b)}, X_{(c)}]$. There is, therefore, a tuple $t' \in \mathit{rel}(\rho)$ such that

$$t'[X_{(a)}, X_{(b)}, X_{(c)}] = \bigl\langle \tau(p_a), \tau(p_b), \tau(p_c) \bigr\rangle.$$

Given the specific values and positions of $X_{(a)}$, $X_{(b)}$, and $X_{(c)}$ in t', it is easily seen that the tuple t' must belong to the group of *clause-induced tuples*, and more specifically, t' is induced by precisely clause C_h and truth value assignment $\tau[p_a, p_b, p_c]$. To see this, let us first recall that in our encoding of a tuple identifier the first bit (i.e., bit 0) is always 0 and the last bit (i.e., bit r) is always 1, which is *never* the case for the encoding of a truth value. Now consider $\tau(p_a)$. If $\tau(p_a) = 0$, then $t(X_{(a)}) = t'(X_{(a)}) = t'(X_a^r) = 0$. If $X_{(a)} = X_a^r$ were part of a tuple identifier, then $t[X_{(a)}]$, which is identical to $t[X_a^r]$, could never have value zero, because, bit r of a tuple identifier is always 1. Therefore, $X_{(a)}$ must be part of a (representation of a) truth value assignment. Similarly, if $\tau(p_a) = 1$, then $t(X_{(a)}) = t'(X_{(a)}) = t'(X_a^0) = 1$. If $X_{(a)} = X_a^0$ were part of a tuple identifier, $t[X_{(a)}]$ could never have value one, because all tuple identifiers have value zero at their bit position of index zero. Therefore, again, $X_{(a)}$ must be part of a (representation of a) truth value assignment. Exactly the same reasoning applies to $X_{(b)}$ and $X_{(c)}$. In summary, t' is a tuple of ρ that exactly describes truth value assignment τ restricted to the three propositional variables p_a, p_b, and p_c. Given that these propositional variables jointly occur in clause C_h, t' is a clause-induced tuple, and τ is a "legal" truth value assignment that satisfies C_h. Given that C_h was an arbitrary clause of C, τ satisfies all clauses of C, and thus C is satisfiable. We are done for $k = 3$. The proof is easily modified to hold for any larger fixed value k. It suffices, for example, to start with kSAT instead of 3SAT. The proof goes through with the obvious adjustments to the numeric parameters. □

Appendix C. Proof of Theorem 4.4

Theorem 4.4. *For each fixed integer $k \geqslant 2$, deciding for a single tri-valued constraint ρ whether $sol(\Pi_{S_k}(\rho)) = \rho$, that is, whether ρ is k-decomposable, is coNP-complete.*

Proof. For all constants k, the membership in coNP of our decision problem is already covered by (the upper bound in) Theorem 4.2. Moreover, the coNP-hardness for $k \geqslant 3$ is already proven in Theorem 4.3, as bi-valued relations are trivially also k-valued relations (where the additional $k-2$ values appear in the domains but not in the actual constraint relations). Thus, what remains to be done is to prove coNP-hardness for $k = 2$.

We use a transformation from 3COL from a graph $G = (V, E)$ as described in the proof of Theorem 4.2 by applying similar vectorization techniques as in the proof of Theorem 4.3. In particular, consider the relation ρ obtained from the 3-colorability network $N_{3\mathrm{COL}}$ in the hardness part of the proof of Theorem 4.2, and let $s = |\rho|$ be the cardinality of ρ. Rather than transforming $N_{3\mathrm{COL}}$ (and thus the graph G) to ρ, we will transform it to a tri-valued constraint ρ^* of the same cardinality s, that closely resembles ρ. To this aim, for $1 \leqslant i \leqslant n$, every scope variable X_i of $N_{3\mathrm{COL}}$ (and thus of ρ) is replaced by a block of $s+1$ variables $X_i^0, \ldots, X_i^s$, which either encodes a color from $\{r, g, b\}$, or a tuple identifier. We here use the following encoding:

- Color *red* is encoded as a block consisting of $s+1$ consecutive positions having value r.
- Color *green* is encoded as a block consisting of $s+1$ consecutive positions having value g.
- Color *blue* is encoded by a leading b (as an assignment to X_i^0) followed by a block containing s consecutive positions having value r.
- The tuple identifier for tuple number d is a block of length $s+1$ starting with a sequence of one or more r elements, having a b in the position corresponding to X_i^d, followed by g elements. In other terms, this tuple identifier is a sequence of length $s+1$ of the form $r, \ldots, r, b, g, \ldots, g$, whose $d+1$st component is b.

The new relation ρ^* thus has $dom(\rho^*) = \{r, g, b\}$ and

$$scope(\rho^*) = (X_1^0, X_1^1, \ldots, X_1^s, X_2^0, X_2^1, \ldots, X_2^s, \ldots, X_n^0, X_n^1, \ldots, X_n^s).$$

Claim. *G is 3-colorable iff $sol(\Pi_{S_2}(\rho^*)) - rel(\rho^*)$ is non-empty.*

The *if-part* is not hard to see from our construction. In fact, each correct graph coloring τ gives rise to a tuple t in $sol(\Pi_{S_2}(\rho^*)) - rel(\rho^*)$ whose vectorized component $t[X_i^0, \ldots, X_i^s]$ representing vertex v_i consists of the encoding of the color $\tau(v_i)$.

Let us now prove the *only-if* part. Assume there exists a tuple t in $sol(\Pi_{S_2}(\rho^*)) - rel(\rho^*)$. We can show by similar arguments as in the proof of Theorem 4.3 that G must be 3-colorable. This is shown by the following successively derived facts:

1. Tuple t can never have value b in an X_i^j-component with $j \neq 0$. In fact, if it had a b assigned to a variable X_i^ℓ with $\ell \neq 0$, this assignment would occur in a single tuple t' of $rel(\rho^*)$ only. Therefore in each relation $rel(c)$ of any constraint c of $\Pi_{S_2}(\rho^*)$ where $scope(c)$ contains X_i^ℓ would contain a single tuple having $X_i^\ell = b$. But this means that the join of all relations $\Pi_{S_2}(\rho^*)$ contains a single tuple having $X_i^\ell = b$, namely t' itself. But would imply $t = t' \in rel(\rho^*)$ which is a contradiction.
2. No pair of consecutive values of any block $t[X_i^1, \ldots, X_i^s]$, for $1 \leqslant i \leqslant n$ can coincide with rg or gr. In fact, by construction, neither rg or gr occur as consecutive values in two consecutive columns labeled X_i^ℓ, $X_i^{\ell+1}$, of $rel(\rho^*)$, where $\ell \geqslant 1$. Therefore, no relation $rel(c)$ of any constraint c of $\Pi_{S_2}(\rho^*)$, whose scope is X_i^ℓ, $X_i^{\ell+1}$, where $\ell \geqslant 1$, contains tuple rg or tuple gr. It follows that the join $sol(\Pi_{S_2}(\rho^*))$ cannot contain any tuple having rg or tuple gr in consecutive components corresponding to the variables (attributes) X_i^ℓ, $X_i^{\ell+1}$, where $\ell \geqslant 1$. Given that $t \in sol(\Pi_{S_2}(\rho^*))$, the same follows for tuple t.
3. For each $1 \leqslant i \leqslant n$, the block $t[X_i^1, \ldots, X_i^s]$ is made entirely of the same value, namely, either r or g. This follows immediately from the above Facts 1 and 2.

4. For $1 \leqslant i \leqslant n$, each block of values $t[X_i^0, \ldots, X_i^s]$ precisely encodes one of the colors *red*, *green*, or *blue*, according to our encoding scheme. To show this, it is sufficient to show that for $1 \leqslant i \leqslant n$, if $t[X_i^1] = r$ then $t[X_i^0] \in \{r, b\}$, and if $t[X_i^1] = g$ then $t[X_i^0] = g$. This is shown just in the same way as Fact 2 above. By construction of ρ^*, the same property holds for each tuple of ρ^*, and thus for all the constraints with scope $\{X_i^0, X_i^1\}$ of $\Pi_{S_2}(\rho^*)$. Therefore, the property must also hold for each tuple of the join $sol(\Pi_{S_2}(\rho^*))$ of $\Pi_{S_2}(\rho^*)$, and thus, in particular, for t.
5. For each edge $\langle v_a, v_b\rangle \in E$, the blocks $t[X_a^0, \ldots, X_a^s]$ and $t[X_b^0, \ldots, X_b^s]$ represent *different* colors. To show this, define $X_{(a)} := X_a^s$ if $X_a^0 = r$ and $X_{(a)} := X_a^0$ otherwise. Similarly, define $X_{(b)} := X_b^s$ if $X_b^0 = r$ and $X_{(b)} := X_b^0$ otherwise. Let q be the constraint of $\Pi_{S_2}(\rho^*)$ with $scope(q) = \{X_{(a)}, X_{(b)}\}$. Clearly $t[X_{(a)}, X_{(b)}] = q[X_{(a)}, X_{(b)}]$. Thus there is a tuple $t' \in rel(\rho^*)$ such that $t[X_{(a)}, X_{(b)}] = t'[X_{(a)}, X_{(b)}]$. However, due to the particular value-position combinations, neither $t'[X_{(a)}]$ nor $t'[X_{(b)}]$ can be part of a tuple identifier, and they thus jointly represent a legal coloring of the edge $\langle v_a, v_b\rangle$ of G. Since this is true for all edges $\langle v_a, v_b\rangle$ of G, all edges of G are correctly colored by the coloring expressed by tuple t.

Therefore, G is 3-colorable. This concludes the proof of the *only-if* part of our claim, and thus the proof of our theorem. □

References

[1] Samson Abramsky, Relational databases and Bell's theorem, submitted for publication.
[2] Samson Abramsky, Relational hidden variables and non-locality, Studia Logica, in press.
[3] Isolde Adler, Georg Gottlob, Martin Grohe, Hypertree width and related hypergraph invariants, European J. Combin. 28 (8) (2007) 2167–2181.
[4] Christian Bessiere, Constraint propagation, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Chapter 3, 2006.
[5] Marco Cadoli, Francesco M. Donini, A survey on knowledge compilation, AI Commun. 10 (3–4) (1997) 137–150.
[6] Hervé Cros, Compréhension et apprentissage dans les résaux de contraintes, PhD thesis, Université de Montpellier, 2003, cited in [4], currently unavailable.
[7] Rina Dechter, From local to global consistency, Artificial Intelligence 55 (1) (1992) 87–108.
[8] Rina Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[9] Rina Dechter, Judea Pearl, Structure identification in relational data, Artificial Intelligence 58 (1992) 237–270.
[10] Gerhard Fleischanderl, Gerhard Friedrich, Alois Haselböck, Herwig Schreiner, Markus Stumptner, Configuring large systems using generative constraint satisfaction, IEEE Intell. Syst. 13 (4) (1998) 59–68.
[11] Daya Ram Gaur, Algorithmic complexity of some constraint satisfaction problems, Master of Science (MSc) Thesis, Simon Fraser University, April 1995, currently available at http://summit.sfu.ca/system/files/iritems1/6666/b17427204.pdf.
[12] Martin Gebser, Benjamin Kaufmann, André Neumann, Torsten Schaub, Conflict-driven answer set enumeration, in: Chitta Baral, Gerhard Brewka, John S. Schlipf (Eds.), LPNMR, in: Lecture Notes in Comput. Sci., vol. 4483, Springer, 2007, pp. 136–148.
[13] Martin Gebser, Benjamin Kaufmann, Torsten Schaub, Solution enumeration for projected boolean search problems, in: Willem Jan van Hoeve, John N. Hooker (Eds.), CPAIOR, in: Lecture Notes in Comput. Sci., vol. 5547, Springer, 2009, pp. 71–86.
[14] Georg Gottlob, Nicola Leone, Francesco Scarcello, A comparison of structural CSP decomposition methods, Artificial Intelligence 124 (2) (2000) 243–282.
[15] Georg Gottlob, Nicola Leone, Francesco Scarcello, Hypertree decompositions: A survey, in: Jiri Sgall, Ales Pultr, Petr Kolman (Eds.), MFCS, in: Lecture Notes in Comput. Sci., vol. 2136, Springer, 2001, pp. 37–57.
[16] Georg Gottlob, Nicola Leone, Francesco Scarcello, Hypertree decompositions and tractable queries, J. Comput. System Sci. 64 (3) (2002) 579–627.
[17] Georg Gottlob, On minimal constraint networks, in: Jimmy Ho-Man Lee (Ed.), CP, in: Lecture Notes in Comput. Sci., vol. 6876, Springer, 2011, pp. 325–339.
[18] Marc Gyssens, Peter Jeavons, David A. Cohen, Decomposing constraint satisfaction problems using database techniques, Artificial Intelligence 66 (1) (1994) 57–89.
[19] Peter Honeyman, Richard E. Ladner, Mihalis Yannakakis, Testing the universal instance assumption, Inform. Process. Lett. 10 (1) (1980) 14–19.
[20] Chris Houghton, David A. Cohen, Solution equivalent subquadrangle reformulations of constraint satisfaction problems, in: Peter van Beek (Ed.), CP, in: Lecture Notes in Comput. Sci., vol. 3709, Springer, 2005, p. 851.
[21] Henry A. Kautz, Bart Selman, A general framework for knowledge compilation, in: Harold Boley, Michael M. Richter (Eds.), PDK, in: Lecture Notes in Comput. Sci., vol. 567, Springer, 1991, pp. 287–300.
[22] Christophe Lecoutre, Constraint Networks – Techniques and Algorithms, John Wiley and Sons, 2009.
[23] Alan Mackworth, Eugene Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, Artificial Intelligence 25 (1) (1985) 65–74.
[24] David Maier, The Theory of Relational Databases, Computer Science Press, 1983.
[25] David Maier, Yehoshua Sagiv, Mihalis Yannakakis, On the complexity of testing implications of functional and join dependencies, J. ACM 28 (4) (1981) 680–695.
[26] Ugo Montanari, Networks of constraints: Fundamental properties and applications to picture processing, Inform. Sci. 7 (1974) 95–132.
[27] Ugo Montanari, Francesca Rossi, Fundamental properties of networks of constraints: A new formulation, in: L. Kanal, V. Kumar (Eds.), Search in Artificial Intelligence, 1988, pp. 426–449.
[28] Dan Olteanu, Christoph Koch, Lyublena Antova, World-set decompositions: Expressiveness and efficient algorithms, Theoret. Comput. Sci. 403 (2–3) (2008) 265–284.
[29] Dan Olteanu, Jakub Zavodny, Factorised representations of query results, in: Proc. International Conference on Database Theory ICDT 2012, Berlin, Germany, March 26–30, 2012.
[30] Robert Rodošek, A new approach on solving 3-satisfiability, in: Jacques Calmet, John Campbell, Jochen Pfalzgraf (Eds.), Artificial Intelligence and Symbolic Mathematical Computation, in: Lecture Notes in Comput. Sci., vol. 1138, Springer, Berlin, Heidelberg, 1996, pp. 197–212.
[31] Edward Tsang, Foundations of Constraint Satisfaction, Academic Press, 1993.
[32] Christopher Umans, Tiziano Villa, Alberto L. Sangiovanni-Vincentelli, Complexity of two-level logic minimization, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 25 (7) (2006) 1230–1246.
[33] Alexey Voronov, Knut Åkesson, Fredrik Ekstedt, Enumeration of valid partial configurations, in: K. Shchekotykhin, D. Jannach, M. Zanker (Eds.), Proceedings IJCAI 2011 Workshop on Configuration, Barcelona, Spain, July 16, 2011, in: CEUR Workshop Proc., vol. 755, 2011, paper 04, available at http://ceur-ws.org/Vol-755/paper04.pdf.