On Minimal Constraint Networks*

Georg Gottlob

Computer Science Department and Oxford Man Institute,
University of Oxford - Oxford OX1 3QD UK
gottlob@cs.ox.ac.uk

Abstract. In a minimal binary constraint network, every tuple of a constraint relation can be extended to a solution. It was conjectured that computing a solution to such a network is NP hard. We prove this conjecture. We also prove a conjecture by Dechter and Pearl stating that for $k \geq 2$ it is NP-hard to decide whether a constraint network can be decomposed into an equivalent k-ary constraint network, and study related questions.

1 Introduction

In his seminal 1974 paper [11], Montanari introduced the concept of *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 $\{X_1, \ldots, X_v\}$ 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 $1 \leq i < j \leq v$ 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). The minimal network $M(N)$ is unique and its solutions are exactly those of the original network, i.e., $sol(N) = sol(M(N))$.

Obviously, $M(N)$, which can be considered a heavily pruned compiled version of N, is hard to compute. However, with $M(N)$ at hand, we can answer a number of queries in polynomial time that would otherwise be NP hard. Typically, these are queries that involve one or two variables only, for example, "Is there a solution for which $X_5 < 8$?" or "what is the maximal value of X_3 such that X_7 is minimized?". 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. It was even unclear if it is tractable to compute an arbitrary single solution on the basis of $M(N)$. Gaur [7] 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 [4]. Note that this problem deviates in two ways from

* Future improvements and extended versions of this paper will be published in CORR at http://arxiv.org/abs/1103.1604

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 is meant by NP-hardness, 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 NP=P. In the light of this, we can formulate Dechter's conjecture as follows:

Conjecture 1 (Dechter[4]). *Unless P=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" [1], he used results of Cros [2] to show that no backtracking-free algorithm for computing a solution from a minimal network can exist unless the Polynomial Hierarchy collapses to its second level (more precisely, until $\Sigma_2^p = \Pi_2^p$). 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 unfair to some solutions.

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). The minimal networks in the proof, however, have an unbounded number of domain values. 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. We show that even in the bounded domain case, the problem of computing a single solution remains NP-hard (Section 3.3).

In Section 4, we deal with problems of network minimality checking and structure identification. In Section 4.1, we generalize and slightly strengthen a result by Gaur [7] by showing that it is NP hard to determine whether a k-ary network is minimal, even in case of bounded domains. Then, in Section 4.2, we study the complexity of checking whether a network N consisting of a single constraint relation (typically of arity $\geq k$) 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 for N. Dechter and Pearl [5] conjectured that this problem is NP-hard for $k \geq 2$. We prove this conjecture.

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 a hint 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 [4,1], we sometimes use a slightly different notation which is more convenient for our purposes.

Constraints, networks, and solutions. A *k-ary constraint c* is a tuple $(scope(c), rel(c))$. The scope $scope(c)$ of c is a sequence of k variables $scope(c) = (X_{i_1}, \ldots, X_{i_k})$, 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 arity $arity(c)$ of a constraint c is the number of variables in $scope(c)$. The set $\{X_{i_1}, \ldots, X_{i_k}\}$ of all variables occurring in constraint c is denoted by $var(c)$.

A *Constraint Network N* consists of

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

The *domain* $dom(N)$ of a constraint network N as defined above consists of the union of all variable domains: $dom(N) = \bigcup_{X \in var(N)} dom(X)$. The *schema* of N is the set $schema(N) = \{scope(c) | c \in cons(N)\}$ of all scopes of the constraints of N. We call N *binary* (*k-ary*) if $arity(c) = 2$ $(arity(c) = k)$ 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 $W \longrightarrow dom(W)$, 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, in case θ is understood and is consistent, then we 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 $sol(N)$. 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 the variables occurring in $scope(c)$ are precisely those of 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. We call a k-ary constraint network N *complete*, if for each set U of k of variables, there is a constraint c such that $U = var(c)$. 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, for each set $U = \{X_{i_1}, \ldots, X_{i_k}\}$ that is in no scope of N, we may just add the trivial constraint $\top_U$ with $scope(\top_U) = (X_{i_1}, \ldots, X_{i_k})$ and $rel(\top_U) = dom(X_{i_1}) \times dom(X_{i_2}) \times dom(X_{i_k})$. For this reason, we may, without loss of generality, restrict our attention to complete networks. Some authors, such as Montanari [11] who studies binary networks, make this assumption explicitly, others, such as Dechter [4] make it implicitly. We here assume unless otherwise stated, that k-ary networks are complete. In particular, we will assume without loss of generality, that when a binary constraint network N is defined over variables $X_1, \ldots, X_v$, that are given in this order, then the constraints are such that their scopes are precisely all pairs (X_i, X_j) such that $1 \leq i < j \leq v$. For a binary constraint network N over variables $\{X_1, \ldots, X_v\}$, we denote the constraint with scope (X_i, X_j) by c_{ij}^N.

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 consists of all constraints 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 $N \subseteq N'$, 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 $\Pi_S(c) = \{\Pi_s(c) | s \in S\}$. If N is a constraint network and S a schema all of whose variables are contained in $var(N)$, then $\Pi_S(N) = \{\Pi_S(c) | c \in N\}$.

Minimal networks. If c is a constraint over variables $var(c) = \{X_1, \ldots, X_v\}$, we denote by $S_k(c)$ the k-ary schema over $var(c)$ having as scopes precisely all (ordered) lists of k variables from $var(c)$, i.e., all scopes $(X_{i_1}, X_{i_2}, \ldots, X_{i_k})$, where $1 \leq i_1 < i_2 < \cdots < i_{k-1} < i_k \leq v$. Thus $\Pi_{S_k}(c)$ is the constraint network obtained by projecting c over all ordered lists of k variables from $var(C)$. In particular, $\Pi_{S_2}(c)$ consists of all constraints $\Pi_{X_i, X_j}(c)$ such that X_i and X_j are variables from $var(c)$ with $i < j$.

It was first observed in [11] 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' for which $sol(N') = sol(N)$. Minimality here is with respect to the above defined "$\subseteq$"-relation among binary networks. More generally, for $k \geq 2$, each k-ary network there is a unique k-ary minimal network $M_k(N)$ that is the intersection of all k-ary networks N' for which $sol(N') = sol(N)$. (For the special case $k = 2$ we have $M_2(N) = M(N)$.) The following is well-known [11,12,4,1] 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 minimal iff $\Pi_{S_k}(sol(N)) = N$.
- A k-ary network N is minimal iff $M_k(N) = N$.
- A k-ary network N is satisfiable (i.e., has at least one solution) iff $M_k(N)$ is nonempty.

It is obvious that for $k \geq 2$, $M_k(N)$, is hard to compute. In fact, just *deciding* whether for a network N, $M_k(N)$ is the empty network (i.e., has only empty relations as constraint relations) is co-NP 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 [8].) 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., [13]) that each binary constraint network N on n variables can be represented as 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). Gaur [7] gave the following nice characterization of minimal networks: A solvable 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.

3 NP Hardness of Computing Minimal Network Solutions

To show that computing a single solution from 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 SAT 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

Definition 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.*

Lemma 1 (Symmetry Lemma). *For each fixed integer $k \geq 1$, there exists a polynomial-time transformation that transforms each 3SAT instance C into a k-supersymmetric instance C^* which is satisfiable if and only if C is satisfiable.*

Proof. We first prove the lemma for $k = 2$. Consider the given 3SAT instance C. Let us 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 $Disj^-$ 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 $(\bar{p_1} \vee \bar{p_4} \vee \bar{p_5}) \in Disj^-(p)$. 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 $\delta$ be the disjunction of all literals in $K$ different from $p$ and from $\neg p$;[1]
  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.
```

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^*. We can actually replace each clause of C at once and independently by the corresponding 1000 clauses, which –assuming appropriate data

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

structures– can be done in linear time. The entire transformation from C to C^* can thus 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. It is sufficient to show 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. The statement 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 $New(p)$ is 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 it is shown 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 $\bar{p}$ 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 $Newp$. Thus when replacing these atoms by p and assigning p true, the resulting clause K remains true. Now consider a clause $K = \bar{p} \vee \delta$ 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 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 $\bar{p_i} \vee \bar{p_j} \vee \bar{p_k} \vee \delta$. Given that this clause is satisfied by τ'', but τ'' falsifies $\bar{p_i} \vee \bar{p_j} \vee \bar{p_k}$, δ is satisfied by τ'', and since δ contains no $New(p)$-literals, δ is also satisfied by τ'. Therefore, $K = \bar{p} \vee \delta$ 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. $\Diamond$

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. Note that, for any initially fixed pair of propositional variables $p_i, q_j \in propvar(C^*)$, where $1 \leq i, j \leq 5$, the construction of τ^* gives us a large enough degree of freedom so to choose τ^* in order 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 Statement 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. $\diamondsuit$

The proof of 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 $Disj^+$ is now defined as the set of all disjunctions of $k + 1$ positive atoms from $New(p)$. Similarly, $Disj^-$ is now defined as the set of all disjunctions of $k + 1$ negative literals obtained by negating atoms from $New(p)$.
- Whe replace the numbers 2 and 3 by k and $k + 1$, respectively.
- We note that now each 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 observe that any 2-supersymmetric SAT instance is trivially also 1-supersymmetric, which settles the theorem for the case $k = 1$ (that we consider for completeness reasons only). $\square$

3.2 Intractability of Computing Solutions

Let us use the Symmetry Lemma for proving our main result about the intractability of computing solutions from a minimal constraint network.

Theorem 2. *For each fixed constant $k \geq 2$, 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 nonempty 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 each clause set $C = \{K_1, \ldots, K_r\}$ into a binary constraint network $S(C) = N_C$ as follows. The set of variables $var(N_C)$ is defined by $var(N_C) = C = \{K_1, \ldots, K_r\}$. For each variable K_i of N_C, 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 $rel(c_{ij}) = (dom(K_i) \times dom(K_j)) - \{(p, \bar{p}), (\bar{p}, p) \,|\, p \in propvar(C)\}$. It is easy to see that C is satisfiable iff N_C is solvable. Basically, N_C is solvable, iff we can pick one literal per clause such that the set of all picked literal contains no atom together with its negation. But this is just equivalent to the satisfiability of C. Obviously, the transformation S is feasible in polynomial time.

Let us now look at constraint networks $N_{C^*} = S(C^*)$, where C^* is obtained via transformation T as in Lemma 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 1, for every pair ℓ_1, ℓ_2 of distinct non-contradictory literals, there is a satisfying assignment that makes both literals true. In case N_{C^*} is solvable, given our particular construction of N_{C^*}, this means that for every constraint c_{ij}, we may pick each pair (ℓ_1, ℓ_2) in $rel(c_{ij})$ as part of a solution. No such pair is useless. It follows that if N_{C^*} is solvable, then $M(N_{C^*}) = N_{C^*}$. On the other hand, if C^* (and thus C) is not satisfiable, then $M(N_{C^*})$ is the empty network. Thus C is satisfiable iff $M(N_{C^*}) = N_{C^*}$ i.e., iff N_{C^*} is minimal[2] Note that C^*, as constructed in the proof of Theorem 2, is a 9SAT instance, hence the cardinality of the domain of each variable of N_{C^*} is bounded by 9.

We are now ready for specifying our 3SAT-solver A^* that works in polynomial time, and hence witnesses NP=P. To a 3-SAT input instance C, A^* first applies T and computes $C^* = T(C)$ in polynomial time. Then A^* transforms C^* via S in polynomial time to N_{C^*}. If N_{C^*} is empty, then, C^* and C are not satisfiable, and A^* outputs "unsatisfiable" and stops. Otherwise, A^* submits N_{C^*} to A and proceeds as follows:

[2] From this, by the way, it follows that checking whether a given binary network is minimal is NP-hard, and thus NP-complete; see also Section 4.1.

1. If A on input N_{C^*} does not produce any output after $p(|N_{C^*}|)$ steps, then A^* outputs unsatisfiable and stops. (This is justified as follows: if C^* (and hence C) was satisfiable, then, by construction, N_{C^*} would be a satisfiable minimal network, and hence, by definition of A, $A(N_{C^*})$ would output a solution after at most $p(|N_{C^*}|)$ steps. Contradiction.)
2. If A produces an output w, then A^* checks if w is effectively a solution to N_{C^*}.
3. If w is not a solution to N_{C^*}, then N_{C^*} cannot be minimal. Thus A^* outputs ”unsatisfiable” and stops.
4. If w is a solution to N_{C^*}, then N_{C^*} is solvable, and so is C^* and C. Thus A^* outputs ”satisfiable” and stops.

In summary, A^* is a polynomial-time 3SAT checker. The theorem for $k = 2$ follows.

For $k > 2$, the proof is analogous to the one for $k = 2$. The only significant change is that now the transformation S now creates a k-ary constraint c_K for each ordered list of k clauses from C, rather than a binary one. The resulting constraint $N_{C^*} = S(C^*)$, where C^* is as constructed in Lemma 1 then does the job. □

3.3 The Case of Bounded Domains

Theorem 2 says 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. We show that even in case $|dom(N)|$ is bounded, computing a single solution from a minimal network N is hard.

Theorem 3. *For each fixed $k \geq 2$, 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 constant.*

Proof sketch. 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 2 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. Thus for each K_a we have a bijection $f_a : dom(K_a) \longleftrightarrow \{1, 2, \ldots, 9\}$. Of course the same literal ℓ may be represented by different numbers for different variable-domains, i.e., it may well happen that $f_a(\ell) \neq f_b(\ell)$. Similarly, a value i in for X_a may correspond to a completely different literal than the same number i for X_b, i.e., $f_a^{-1}(i)$ may well differ from $f_b^{-1}(i)$. Let us thus simply translate N_{C^*} into a network $N^{\#}_{C^*}$, where each literal ℓ in each column of a variable X_a is replaced by $f_a(\ell)$. It is easy to see that N_{C^*} and $N^{\#}_{C^*}$ are equivalent and that the solutions of N_{C^*} and $N^{\#}_{C^*}$ are in a one-to-one relationship. Obviously, $N^{\#}_{C^*}$ inherits from N_{C^*} the property to be minimal in case it is solvable. Therefore, computing a solution to a network in which only nine values occur in total in the constraint relations is intractable unless NP=P. □

4 Minimal Network Recognition and Structure Identification

In this section we first study the complexity of recognizing whether a k-ary network M is the minimal network of a k-ary network N. We then analyze the problem of deciding whether a k-ary network M is the minimal network of a single constraint.

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 [7] has shown the following for binary networks:

Proposition 1 (Gaur [7]). *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. *For each $k \geq 2$, 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 2 . For each $k \geq 2$, 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. □

4.2 Structure Identification and k-Representability

This section is dedicated to the problem of representing single constraints (or single-constraint networks)through equivalent k-ary minimal networks with smaller relations. By a slight abuse of terminology, we will here identify a single-constraint network $\{\rho\}$ with its unique constraint ρ.

Definition 2. *A complete k-ary network M is a* minimal k-ary network *of ρ iff*

1. *$sol(M) = \rho$, and*
2. *every k-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 2 anyway.

Proposition 2. *Let ρ be a constraint. The following three statements are equivalent: (i) ρ has a minimal k-ary network; (ii) $sol(\Pi_{S_k}(\rho)) = \rho$; (iii) 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 network of ρ, if it exists, already represents all k-ary networks that are equivalent to ρ.

The complexity of deciding whether a minimal network for a relation ρ exists has been stated as an open problem by Dechter and Pearl in [5]. 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* [5]. 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 5 (Dechter and Pearl [5]). *For each positive integer $k \geq 2$, deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$ is NP-hard*[3].

As already observed by Dechter and Pearl in [5], there is a close relationship between the k-representability of constraint relations and some relevant database problems. 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. [4]). The decomposition of relations plays an important role in the database area, in particular in the context of normalization [9]. 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 [10][4].

Proposition 3 (Maier, Sagiv, and Yannakakis [10]). *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 3 is weaker than Conjecture 5 and does not by itself imply it, nor so does its proof given in [10]. In fact, Conjecture 5 speaks about the very specific sets S_k for $k \geq 2$, which are neither mentioned in Proposition 3 nor used in its proof. To prove Conjecture 5 we thus developed an new and independent hardness argument.

[3] Actually, the conjecture stated in [5] 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 part of the input instance. However, from the context and use of this conjecture in [5] it is clear that Dechter and Pearl actually intend NP-hardness for each fixed $k \geq 2$.

[4] As mentioned by Dechter and Pearl [5], Jeff Ullman has proved this result, too. In fact, Ullman, on a request by Judea Pearl, while not aware of the specific result in [10], 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.

Theorem 6. *Given a single constraint ρ and an integer $k \geq 2$, deciding whether $sol(\Pi_{S_k}(\rho)) = \rho$, and thus whether ρ is k-decomposable, is co-NP complete.*

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

Membership. Note that membership already follows from Proposition 3, but let us still give a short proof here for reasons of self-containment. Clearly, $\rho \subset sol(\Pi_{S_k}(\rho))$. Thus $sol(\Pi_{S_k}(\rho)) \neq \rho$ iff the containment is proper, which means that there exists a tuple t in $sol(\Pi_{S_k}(\rho))$ not contained in ρ. One can guess such a tuple t in polynomial time and check in polynomial time that for each k-tuple of variables $X_{i_1}, \ldots, X_{i_k}$ of $var(\rho)$, $i < j$, the projection of t to $(X_{i_1}, \ldots, X_{i_k})$ 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 three-colorable. Let r, g, b be three data values standing intuitively for the three colors red, green, and blue, respectively. Let N_{3COL} be the following constraint network: $var(N_{\text{3COL}}) = \{X_1, \ldots, X_n\}$, $dom(X_i) = \{r, g, b\}$ for $1 \leq i \leq n$, and the schema of N_{3COL} be precisely S_2. Moreover for all $1 \leq i < j \leq n$, N_{3COL} has a constraint c_{ij} with schema (X_i, X_j) and constraint relation r_{ij} defined as follows: if $(i, j) \in E$, then r_{ij} 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_{3COL} is thus a straightforward encoding of 3COL over schema S_2, and obviously G is 3-colorable iff $sol(N_{\text{3COL}}) \neq \emptyset$. Thus the deciding $sol(N_{\text{3COL}}) \neq \emptyset$ is NP hard. Now let us construct from N_{3COL} a single constraint ρ with schema $\{X_1, \ldots, X_n\}$ and constraint relation s as follows. For each constraint c_{ij} of N_{3COL}, ρ contains a tuple t whose X_i and X_j correspond to those of r_{ij}, and whose X_ℓ value, for all $1 \leq \ell \leq n$, $\ell \neq i$, $\ell \neq j$, is a fresh "dummy" constant d_{ij}^t, different from all other used values. We claim that $sol(\Pi_{S_2}(\rho)) \neq \rho$ iff $sol(N_{\text{3COL}}) \neq \emptyset$ (and thus iff G is 3-colorable. This clearly implies the NP-hardness of deciding $sol(\Pi_{S_k}(\rho)) \neq \rho$.

We now show that the claim holds. Trivially, $\rho \subseteq sol(\Pi_{S_2}(\rho))$. It thus follows that $sol(\Pi_{S_2}(\rho)) \neq \rho$ iff there exists a tuple $t_0 \in sol(\Pi_{S_k}(\rho))$ such that $t_0 \notin \rho$. We will argue that t_0 can contain values from $\{r, g, b\}$ only and must be a solution of N_{3COL}. First, assume that t_0 contains two distinct "dummy" constants, say, d, in column X_a and d' in column X_b with $a < b$. This would mean that $\Pi_{X_a X_b}(\rho)$ contains the tuple (d, d'), and thus, that ρ itself contains a tuple with the two *distinct* dummy values d, and d', which clearly contradicts the definition of ρ. It follows that at most one dummy value d may occur in t_0. However, each such dummy value $d = d_{ij}^t$ occurs precisely in one single tuple t of ρ, and thus each relation of $\Pi_{S_k}(\rho))$ has at most one tuple containing d. It follows that there is only one tuple containing d_{ij}^t in $sol(\Pi_{S_k}(\rho))$, which is t itself, but $t \in \rho$. Therefore, t_0 cannot contain any dummy value 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_{3COL}, and vice-versa. Thus $sol(\Pi_{S_k}(\rho)) \neq \rho$ iff $sol(N_{\text{3COL}}) \neq \emptyset$ 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_{3COL}^k 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_{3COL} is extended by use of a distinct dummy value 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 two distinct dummy values cannot jointly occur in a tuple of $sol(\Pi_{S_k}(\rho))$, and that each tuple of $sol(\Pi_{S_k}(\rho))$ that contains a dummy value is already present in ρ because for each dummy 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_{3COL} and thus a valid 3-coloring of G. $\square$

5 Discussion and Future Research

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

- As solution of an open problem posed by Gaur [7] in 1995, and later by Dechter [4], we proved Dechter's conjecture and showed that computing a solution to a minimal constraint network is NP hard.
- We proved a conjecture made in 1992 by Dechter and Pearl [5] by showing that for $k \geq 2$, it is coNP complete to decide whether $sol(\Pi_{S_k}(\rho)) = \rho$, and thus whether ρ is k-decomposable.

We wish to make clear that our hardness results do 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 [6]. 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. In presence of a k-ary minimal constraint network, the satisfiability of queries involving k variables only can be decided in polynomial time. However, our Theorem 2 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 2 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 k-tuples in a k-ary minimal network $M_k(N)$, we may store a full solution t^+ with each k-tuple, where t^+ coincides with t on the k variables of t. Call the so 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 k-tuple t whether it occurs in some solution t^+, why not just memorizing 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 $M_k(N)$.

We currently work on the following problem: Show that deciding if ρ is k-decomposable remains coNP complete for bounded domains, even if $k = 3$ and $dom(\rho)$ is 2-valued, or if $k = 2$ and $dom(\rho)$ is 3-valued. This will hopefully allow us to confirm a further conjecture (Conjecture 3.27 in [5]) about the identification of CNF formulas. Note that for $k = 2$ and two-valued domains, the problem is tractable for reasons similar to those for which binary 2-valued constraint networks can be solved and minimized in polynomial time [3,7]. Another interesting research problem is the following. We may issue queries of the following form against $M_k^+(N)$: `SELECT A SOLUTION WHERE` ϕ. Here ϕ is 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.

Acknowledgments. Work funded by EPSRC Grant EP/G055114/1 "Constraint Satisfaction for Configuration: Logical Fundamentals, Algorithms, and Complexity". We thank C. Bessiere, R. Dechter, D. Gaur, M. Vardi, M. Yannakakis, S. Živný, and the referees for useful comments and/or pointers to earlier work.

References

1. Bessiere, C.: Constraint propagation. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, ch. 3, pp. 29–83 (2006)
2. Cros, H.: Compréhension et apprentissage dans les résaux de contraintes, Université de Montpellier, "PhD thesis, cited in [1], currently unavailable" (2003)
3. Dechter, R.: From local to global consistency. Artif. Intell. 55(1), 87–108 (1992)
4. Dechter, R.: Constraint processing. Morgan Kaufmann, San Francisco (2003)
5. Dechter, R., Pearl, J.: Structure identification in relational data. Artif. Intell. 58, 237–270 (1992)
6. Fleischanderl, G., Friedrich, G., Haselböck, A., Schreiner, H., Stumptner, M.: Configuring large systems using generative constraint satisfaction. IEEE Intell. Systems 13(4), 59–68 (1998)
7. Gaur, D.R.: Algorithmic complexity of some constraint satisfaction problems, Master of Science (MSc) Thesis, Simon Fraser University (April 1995), Currently available at: `http://ir.lib.sfu.ca/bitstream/1892/7983/1/b17427204.pdf`
8. Mackworth, A., Freuder, E.: The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. Artif. Intelligence 25(1), 65–74 (1985)
9. Maier, D.: The theory of relational databases. Computer Science Press, Rockville (1983)
10. Maier, D., Sagiv, Y., Yannakakis, M.: On the complexity of testing implications of functional and join dependencies. J. ACM 28(4), 680–695 (1981)
11. Montanari, U.: Networks of constraints: Fundamental properties and applications to picture processing. Information Sciences 7, 95–132 (1974)
12. Montanari, U., Rossi, F.: Fundamental properties of networks of constraints: A new formulation. In: Kanal, L., Kumar, V. (eds.) Search in Artificial Intelligence, pp. 426–449 (1988)
13. Tsang, E.: Foundations of constraint satisfaction. Academic Press, London (1993)