DOI:10.1145/1810891.1810914

Constraint Satisfaction Problems and Global Cardinality Constraints

By Andrei A. Bulatov and Dániel Marx

Abstract

In a constraint satisfaction problem (CSP) the goal is to find an assignment of a given set of variables subject to specified constraints. A global cardinality constraint is an additional requirement that prescribes how many variables must be assigned a certain value. We study the complexity of the problem CCSP(Γ), the CSP with global cardinality constraints that allows only relations from the set Γ. The main result of this paper characterizes sets Γ that give rise to problems solvable in polynomial time, and states that the remaining such problems are NP-complete.

1. CONSTRAINT PROBLEMS

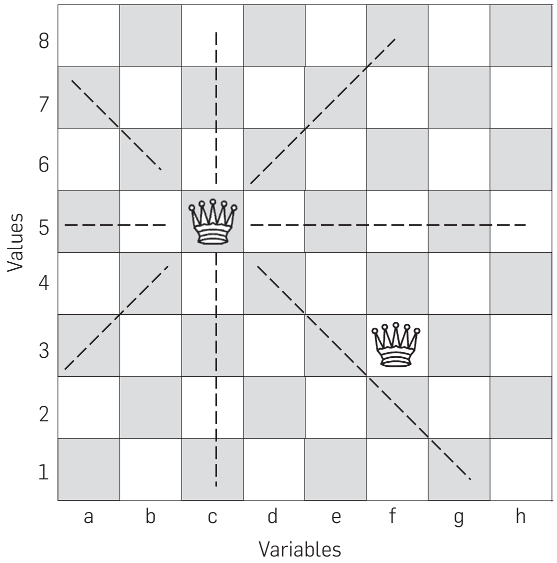
1.1. Constraint satisfaction problem

Among formalisms unifying and classifying various combinatorial problems the *Constraint Satisfaction Problem* (or CSP) is one of the most successful ones. In this problem, we are given a set of variables and a collection of restrictions—constraints—on the allowed combinations of values of the variables; the goal is to find an assignment to the variables so that all constraints are satisfied. Usually constraints are imposed on small sets of variables; thus, the CSP formalizes the idea of finding a global solution bound by local restrictions. The Sudoku puzzle gives a popular toy example of CSP. We need to assign values—numbers from 1 to 9—to variables—entries of the puzzle so that the values of variables in a row, column, or 3 × 3 block are different. Another toy example whose CSP encoding is less obvious is the 8-Queen problem: place eight queens on a 8 × 8 chessboard so that they do not hit each other.[15] To represent it as a CSP we consider the columns $\{a, b, c, d, e, f, g, h\}$ (see Figure 1) as variables that can be assigned values from the set of rows, and the assigned value shows the position of a queen in this column.

Many combinatorial problems readily fall into this framework. For example, in the Graph 3-Coloring problem, the vertices of a given graph are variables to receive one of the three colors, and assignments are constrained by the requirement that adjacent vertices receive different colors. Thus, this problem is a CSP. The list of examples can be extended by other combinatorial problems like Satisfiability, problems in scheduling, temporal and spatial reasoning, and many others.

CSPs have been studied from both practical and theoretical perspectives. On the practical side, the expressive power of the CSP allows to model a wide range of real-world problems from planning[24] and scheduling,[35] frequency assignment problems,[17] to image processing,[32] to programming language analysis,[33] to natural language understanding.[1] A number of commercial and freeware solvers exist capable of solving a wide range of CSPs of nearly industrial scale, and methods of solving constraint problems are developing rapidly.[15] On the theoretical side, researchers focus on several directions such as the complexity of CSPs problems, efficient algorithms for CSPs, where such algorithms exist, and connections of CSPs with other combinatorial problems.[3, 8, 10, 13, 18, 21, 22, 26, 31, 34]

Figure 1. The 8-Queen problem.



1.2. Global constraints

The 'pure' CSP described above is sometimes not enough to model practical problems, as some constraints that have to be satisfied are not 'local' in the sense that they cannot be viewed as applied to only a limited number of variables. Constraints of this type are called *global*. Global constraints are very diverse; the current Global Constraint Catalog (see http://www.emn.fr/x-info/sdemasse/gccat/) lists 313 types of such constraints. In this paper we focus on *global cardinality constraints*.[6, 14]

Some of the global constraints such as the surjectivity of a solution, that is, the requirement that all variables take distinct values (cf. the Sudoku puzzle), allow simulation by local

The original version of this paper was published in the *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science* (Los Angeles, CA, Aug. 11–14, 2009), 419–428.

constraints. Surjectivity can be enforced by requiring that every two variables receive distinct values. However, sometimes it is not possible. In this paper we focus on one type of such 'truly' global constraints, *cardinality constraints*, that impose restrictions on the number of variables assigned certain values, see Figure 2. For instance, in the 3-Coloring problem, a cardinality constraint may require that at least half of the vertices of the graph are colored red.

1.3. Complexity of constraints

As the general CSP is NP-hard, the study of its complexity focuses on considering restricted versions of the problem. There are two principal ways to restrict the CSP, both of them can be applied to CSPs with cardinality constraints as well.

The first approach restricts the way constraints interact. The interaction of constraints can be represented by the *primal graph* whose vertices are variables, and two vertices are connected if and only if they belong to the scope of a constraint. This approach was motivated by the observation that if the primal graph is acyclic or close to acyclic in a well-defined sense (has bounded treewidth), then CSP becomes polynomial-time solvable.[20] Interestingly, attempts to characterize conjunctive queries to databases that can be processed efficiently led to the same question.[26] After a series of recent breakthrough results[21, 31] the structure of polynomial-time solvable CSPs of this type is largely understood.

The second approach to restrict the CSP is to limit the allowed types of constraints. It can be expressed formally as follows. Let the possible values of variables in the problem be taken from a set D (the *domain*). In this paper, we always assume D to be finite. Then every constraint that can be imposed on a set of k variables is a list of all allowed combinations of values these variables can take simultaneously, that is, a k-ary relation on D. If now we fix a set Γ of such relations on D and allow constraints to be chosen only from Γ, we arrive to the problem denoted CSP(Γ). In this context, Γ is often called a *constraint language*. Same restrictions can be applied to problems with cardinality constraints. We use CCSP(Γ) to denote such problem.

Figure 2. Formal definition of CSP and CCSP.

CSP
Let D be a (finite) set (the *domain*). Every instance $I = (V,\mathcal{C})$ of the problem CSP consists of:

- a set V of variables, and
- a set $\mathcal{C}$ of *constraints*. Every constraint is a pair $\langle \mathbf{s}, R\rangle$, where
  - $\mathbf{s} = (v_1, \dots, v_k)$ is a tuple of variables from V, not necessarily distinct, and
  - R is a k-ary relation over D.

A *solution* of $I = (V, \mathcal{C})$ is a mapping φ: $V \to D$ such that for any constraint $\langle \mathbf{s}, R\rangle$, we have $\varphi(\mathbf{s}) \in R$.

CCSP
A global cardinality constraint for an instance $I = (V,\mathcal{C})$ is a mapping π: $D \to \mathbb{N}$ such that $\Sigma_{a\in D}\ \pi(a) = |V|$. Solution φ *satisfies* π if $|\varphi^{-1}(a)| = \pi(a)$ for every $a \in D$. The question is whether or not there is a solution satisfying one of the given cardinality constraints.

CSP(Γ) and CCSP(Γ)
Let Γ be a set (finite or infinite) of relations on D, called a *constraint language*. The problems CSP(Γ) and CCSP(Γ) include those instances of CSP and CCSP, respectively, that use only relations from Γ.

Problems of the form CSP(Γ) and CCSP(Γ) span a wide range of combinatorial problems such as ones in Figure 3, and many others.

Graph 3-Coloring. Let $R_{\neq_3}$ denote the disequality relation on a 3-element set, that is, the binary relation containing all pairs (a, b) of elements from the set such that $a \neq b$:

$$R_{\neq_3} = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 2 & 3 & 1 & 3 & 1 & 2 \end{pmatrix}.$$

(Observe that we write pairs, and later longer tuples of elements vertically, so members of the relation are the columns of the matrix.) Then the 3-Coloring problem equals CSP($\Gamma_{3\text{–Col}}$) where $\Gamma_{3\text{–Col}} = \{R_{\neq_3}\}$.

2-Satisfiability. Recall that a literal is a propositional variable or its negation. A disjunction of literals (of 2 literals) is called a clause (a 2-clause). A propositional formula that is a conjunction of clauses (2-clauses) is said to be a conjunctive normal form, or a CNF (2-CNF) for short. In the 2-Satisfiability problem, given a 2-CNF, the goal is to find an assignment to its variables that makes the formula true. If the set of variables of the CNF is V then every clause defines a constraint on a pair of variables that forbids exactly one combination of values. Let $\Gamma_{2\text{–SAT}}$ be the following set of 4 binary relations, each of which omits a certain pair:

$$R_{x\vee y} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad R_{\bar{x}\vee y} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$
$$R_{x\vee \bar{y}} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad R_{\bar{x}\vee \bar{y}} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Then CSP($\Gamma_{2\text{–SAT}}$) represents 2-Satisfiability and it is known to be polynomial-time solvable.

3-Satisfiability. Analogously to 2-SAT, let $\Gamma_{3\text{–SAT}}$ be the set consisting of eight ternary relations on $\{0, 1\}$, each of which omits a certain triple. Then CSP($\Gamma_{3\text{–SAT}}$) represents 3-Satisfiability and it is NP-complete.

Independent Set. An independent set in a graph is a set of vertices, no two of which are connected with an edge. In the Independent Set problem, given a graph and a natural number k, the question is whether or not there exists an independent set of size k. Let

$$R_{\text{IS}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

that is, R_{IS} = excludes only (1, 1), and $\Gamma_{\text{IS}} = \{R_{\text{IS}}\}$. Now, to reduce Independent Set to the CSP the vertices of a given graph are treated as variables and the constraint R_{IS} is imposed on every pair of adjacent vertices. For any solution of such CSP the variables (vertices) assigned 1 form an independent set in the graph. To express the restriction on the size of an independent set we can use a cardinality constraint that requires that exactly k variables are assigned 1. Therefore Independent Set is equivalent to CCSP(Γ_{IS}). The Independent Set problem is well known to be NP-complete.

Despite such expressive power, problems of the form CSP(Γ) probably cannot capture all combinatorial problems. As is easily seen, all CSPs belong to the class NP. Some of them, such as 3-Coloring or 3-SAT are NP-complete, while others, for example, 2-SAT, belong to the class P, that is, solvable in polynomial time. If P≠NP, there is an infinite hierarchy of complexity classes between P and NP such that problems from different classes are not reducible to each other in a natural sense.[28] However, all known problems CSP(Γ) turn out to be either in P or NP-complete. This phenomenon is known as complexity dichotomy.[18] The dichotomy phenomenon was first discovered by Schaefer[34] for CSPs with 2-element domain, and was later confirmed in many particular cases.[3, 7, 9] This caused Feder and Vardi to pose a conjecture, called the *Dichotomy Conjecture*, that every problem CSP(Γ) is either solvable in polynomial time or is NP-complete. The Dichotomy Conjecture remains open till now.

Remarkably, the phenomenon of complexity dichotomy extends inside P, although a weaker notion of reduction is needed for this. To date, only four complexity classes and a series of very similar classes inside P are known such that CSP(Γ) can be complete in.[2, 29] In some cases the lack of problems CSP(Γ) of intermediate complexity is shown.[29]

Figure 3. More examples of CSPs and CCSPs.

Bipartite Independent Set. We say that a graph is bipartite if the vertices can be partitioned into two classes *X* and *Y* such that every edge connects a vertex of *X* and a vertex of *Y*. In the Bipartite Independent Set problem, we are looking for a independent set containing exactly k_X vertices of *X* and k_Y vertices of *Y*. This problem is equivalent to a CCSP over the domain $\{0_X, 0_Y, 1_X, 1_Y\}$ where each edge is represented by the binary relation

$$R_{\mathrm{BIS}} = \begin{pmatrix} 0_X & 0_X & 1_X \\ 0_Y & 1_Y & 0_Y \end{pmatrix}$$

and we require k_X variables with value 1_X and k_Y variables with value 1_Y in the solution. Bipartite Independent Set is known to be NP-hard. The variant of the problem, where we require an independent set of size *k* in a bipartite graph (without specifying the number of vertices in each class) is polynomial-time solvable; however, this variant cannot be expressed as a CCSP.

Linear Equations. In the regular Linear Equations problem the question is, given a system of linear equations over a finite field, decide whether it is consistent or not. The version of this problem allowing global cardinality constraints asks whether such a system has a solution that assigns each of the elements from the field to a prescribed number of variables. While Linear Equations without cardinality constraints is polynomial-time solvable, cardinality constraints make it NP-complete,[5] even if the variables are over the two element field and every equation is of the form $x + y + z = 1$. This means that CCSP($\{R_{\mathrm{ODD-3}}\}$) is NP-complete, where

$$R_{\mathrm{ODD\text{-}3}} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

is the ternary relation satisfied by an odd number of 1s.

In this paper, we report on a dichotomy theorem for CSPs with cardinality constraints. The next section describes a dynamic programming algorithm that solves CCSPs whenever it can be solved efficiently. In Section 3, we outline the algebraic approach to the CSP and CCSP and show how it can be used to formulate the dichotomy theorem for the CCSP. Finally, in Section 4 we present the main ideas behind the hardness result. A longer version of the paper can be found in.[12]

2. EASY CASES OF CCSP

2.1. Boolean CCSP

To gain some intuition we start with the Boolean CSP and CCSP, in which values are taken from the set $\{0, 1\}$. The dichotomy result for Boolean CSPs[34] identifies six types of tractable relations, that is, those which give rise to a CSP solvable in polynomial time. Among these relations are those representable by a 2-CNF, solution spaces of systems of linear equations over the 2-element field, and some others. If a constraint language Γ is not composed from relations of one of these six types, CSP(Γ) is NP-complete. For CCSPs, a dichotomy result was proved in Creignou et al.[14] The structure of tractable CCSPs is much simpler. Let $R_{=2}$ and $R_{\neq 2}$ denote the equality and disequality relations on $\{0, 1\}$. Then CCSP(Γ) is solvable in polynomial time if and only if every relation from Γ can be expressed by a conjunction of $R_{=2}$ and $R_{\neq 2}$ clauses, and the two constant constraints 0 and 1. Otherwise the Bipartite Independent Set or Linear Equations problems can be reduced to CCSP(Γ), and the problem is NP-complete.

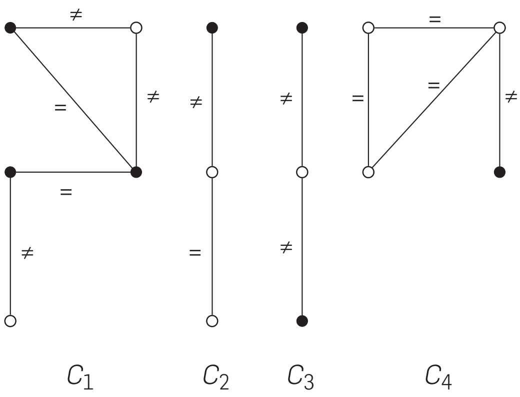
The polynomial-time solvable cases can be handled by a standard application of dynamic programming. Suppose that the instance is given by a set of binary equality/disequality clauses (see Figure 4 for a concrete example). Consider the graph formed by the binary clauses. There are at most two possible assignments for each connected component of the graph: setting the value of a variable uniquely determines the values of all the other variables in the component. Thus the problem is to select one of the two assignments for each component. Trying all possibilities would be exponential in the number of components. Instead, for $i = 1, 2, \ldots$, we compute the set Π_i of all possible pairs (x, y) such that there is a partial solution on the first *i* components containing exactly *x* zeros and exactly *y* ones. It is not difficult to see that Π_{i+1} can be efficiently computed if Π_i is already known.

2.2. Generalizations

We generalize the results of Creignou et al.[14] for arbitrary finite sets and arbitrary constraint languages. As usual, the characterization for arbitrary finite domains is significantly more complex and technical than for the 2-element domain. As a straightforward generalization of the 2-element case, we can observe that the

Figure 4. Using dynamic programming to solve Boolean CCSP with binary equalities and disequalities.

Example 1. Let $\Gamma = \{=_2, \neq_2\}$ contain the binary equality and disequality relations. Consider the following instance of CCSP(Γ) with 15 variables and 13 constraints:



Each component has exactly two satisfying assignments: either the "black" variables have value 0 and the "white" variables have value 1, or vice versa. Let set Π_i contain all possible pairs (x, y) such that the union of the first i components have a solution with x 0's and y 1's. Then

$$\Pi_1 = \{(2, 3), (3, 2)\}$$
$$\Pi_2 = \{(3, 5), (4, 4), (5, 3)\}$$
$$\Pi_3 = \{(4, 7), (5, 6), (6, 5), (7, 4)\}$$
$$\Pi_4 = \{(5, 10), (6, 9), (7, 8), (8, 7), (9, 6), (10, 5)\}$$

If component C_i has b_i black and w_i white vertices, then clearly a pair (x, y) is in C_i if and only if either $(x - b_i, y_i - w_i) \in \Pi_{i-1}$ or $(x - w_i, y_i - b_i) \in \Pi_{i-1}$. This gives us an efficient way of computing Π_i if Π_{i-1} has been computed.

problem is polynomial-time solvable if every relation can be expressed by graphs of bijective mappings. For a mapping $\varphi: A \rightarrow A$, the *graph* of φ is the binary relation consisting of pairs of the form $(a, \varphi(a))$, $a \in A$. In this case, setting a single value in a component uniquely determines all the values in the component. Therefore, if the domain is D, then there are at most $|D|$ possible assignments in each component, and the same dynamic programming technique can be applied (but this time the set Π_i contains $|D|$-tuples instead of pairs).

One might be tempted to guess that the class described in the previous paragraph is the only class where CCSP is polynomial-time solvable. However, it turns out that there are more general tractable classes. First, suppose that the domain is partitioned into equivalence classes, and the binary constraints are mappings between the sets of equivalence classes. This means that the values in the same equivalence class are completely interchangeable. Thus it is sufficient to keep one representative from each class, and then the problem can be solved by the algorithm sketched in the previous paragraph. Again, one might believe that this construction gives all the tractable classes, but the example in Figure 5 shows that there are more complicated constraint languages, where CCSP is polynomial-time solvable, but we have to do two-level dynamic programming on the subcomponents of each component. It is not difficult to make this example more complicated in such a way that we have to look at sub-subcomponents and perform multiple levels of dynamic programming. This suggests that it would be difficult to characterize the tractable relations in a simple combinatorial way.

2.3. Algorithm for the tractable CCSP problems

In this section, we present a general algorithm for solving CCSP. We prove our dichotomy theorem by showing that for every finite constraint language Γ, either this algorithm solves CCSP(Γ) in polynomial time, or CCSP(Γ) is NP-complete. In this section, we cannot give a full characterization of those constraint languages Γ for which the algorithm works: we postpone it to Section 3.3, as it can be done most conveniently using the algebraic tools introduced in the next section.

The first condition that we require is that every relation in Γ is defined by its binary projections. Formally, we say that r-ary relation R is 2-*decomposable,* if there are binary relations R_{ij} $(1 \leq i < j \leq r)$ such that $(a_1, \ldots, a_r) \in R$ if and only if $(a_i, a_j) \in R_{ij}$ for every $1 \leq i < j \leq r$. For example, the relation R in Figure 5 is 2-decomposable, as it is shown by the relations

$$R_{12} = \begin{pmatrix} 1 & 1 & a & d \\ 2 & 4 & b & e \end{pmatrix} \quad R_{13} = \begin{pmatrix} 1 & 1 & a & d \\ 3 & 5 & c & c \end{pmatrix}$$
$$R_{23} = \begin{pmatrix} 2 & 4 & b & e \\ 3 & 5 & c & c \end{pmatrix}.$$

On the other hand, relation $R_{\text{ODD-3}}$ of Figure 3 is *not* 2-decomposable: all three of the corresponding relations R_{12}, R_{13}, R_{23} contain the pair $(0, 0)$, but tuple $(0, 0, 0)$ is not in R.

If a constraint is 2-decomposable, then it can be expressed by a set of binary constraints. Thus in the following, we can assume that every constraint of the CCSP instance is binary.

The algorithm finds all cardinality constraints that are satisfied by solutions of the instance. First, given an instance, we make sure that every variable v is associated with a domain D_v that contains all the values that are useful for this variable. That is, if $\langle(v, w), R\rangle$ is a constraint, then D_v is exactly $\{x \mid (x, y) \in R\}$, or in other words, D_v is exactly the set of values that the pairs of R contain at the position corresponding to v. This is achieved by the standard propagation algorithm, see, e.g., Freuder[19].

A binary constraint $\langle(v, w), R\rangle$ is *trivial* if $R = D_v \times D_w$, allowing any combination of values from the domains of v and w. Let G be the graph formed by the *nontrivial* binary constraints of the problem. If graph G is disconnected, then arbitrary satisfying assignments for the connected components can be combined to obtain a satisfying assignment for the instance. Therefore, the algorithm recurses on the problems induced by connected components, and then merges the solutions using the same dynamic programming approach as for Boolean CCSP (Figure 4). If G is connected, the algorithm chooses an arbitrary variable v and tries to substitute every possible value of D_v into v. This way, we get $|D_v|$ new instances and it is clear that the original problem has a solution satisfying a cardinality

Figure 5. A two-level dynamic programming algorithm for CCSP.

Example 2. We claim that CCSP($\{R\}$) is polynomial-time solvable for the relation

$$R = \begin{pmatrix} 1 & 1 & a & d \\ 2 & 4 & b & e \\ 3 & 5 & c & c \end{pmatrix}.$$

Consider the graph on the variables where two variables are connected if and only if they appear together in a constraint. As in Figure 4, for each component, we compute a set containing all possible cardinality vectors, and then use dynamic programming. In each component, we have to consider only two cases: either every variable is in $\{1, 2, 3, 4, 5\}$ or every variable is in $\{a, b, c, d, e\}$. If every variable of component K is in $\{1, 2, 3, 4, 5\}$, then R can be expressed by the unary constant relation 1, and the binary relation $R' = \{(2, 3), (4, 5)\}$. The binary relations partition component K into sub-components $K_1, \dots, K_t$. Since R' is the graph of a mapping, there are at most 2 possible assignments for each sub-component. Thus we can use dynamic programming to compute the set of all possible cardinality vectors on K that use only the values in $\{1, 2, 3, 4, 5\}$. If every variable of K is in $\{a, b, c, d, e\}$, then R can be expressed as the unary constant relation c and the binary relation $R'' = \{(a, b), (d, e)\}$. Again, binary relation R'' partitions K into sub-components, and we can use dynamic programming on them. Observe that the sub-components formed by R' and the sub-components formed by R'' can be different: in the first case, u and v are adjacent if they appear in the second and third coordinates of a constraint, while in the second case, u and v are adjacent if they appear in the first and second coordinates of a constraint.

constraint if and only if one of the new instances has such a solution. Thus in this case, the problem can be solved by recursively solving $|D_v|$ instances and taking the union of the set of cardinality constraints satisfied by these instances.

There is no question that the scheme described above finds every cardinality constraint satisfied by the instance. The only issue is whether the running time is polynomial: branching into $|D_v|$ directions in the case when G is connected can create an exponentially large recursion tree. We identify a useful special case that guarantees a polynomial bound on the size of the recursion tree. After substituting a value into v, we can rerun the propagation algorithm to reduce the domains of the variables by throwing away those values that are no longer useful. The key property that we require is the following:

> **Key Property:** If G is connected, then no matter what value we substitute, propagation strictly decreases the domain of *every* variable.

If this property is true, then the algorithm has to terminate after at most $|D|$ substitutions, and therefore the height of the recursion tree is at most $|D|$, which is constant for a fixed constraint language. This gives us a polynomial bound on the size of the recursion tree.

Are there constraint languages Γ for which the key property described above holds? Yes, there are, for example, if every binary relation is the graph of a bijective mapping and G is connected, then substituting any value to a variable v decreases the domain of every other variable to a single element. As mentioned earlier, it is not easy to give a simple combinatorial characterization of those sets Γ for which the algorithm works (in the next section, we characterize them in a more algebraic way). We can at least give some necessary conditions that show what kind of generalizations of mappings should we deal with.

Let R be a binary relation from a set A to set B, that is, $R \subseteq A \times B$. Relation R is said to be a *thick mapping* if whenever pairs (a, c), (a, d), (b, c) belong to R, the pair (b, d) also belongs to R. As is easily seen, any thick mapping R has two associated equivalence relations α and β on A and B, respectively, such that R can be thought of as a mapping from the set of equivalence classes of α to that of β.

To give some intuition why it is a problem if a relation is not a thick mapping, consider the relation $R = \{(a, c), (a, d), (b, c)\}$. Suppose that there are only two variables v, w and there is a single constraint $\langle (v, w), R\rangle$. In this case, the domains are $D_v = \{a, b\}$ and $D_w = \{c, d\}$. The constraint is nontrivial, thus the graph G is connected. But if we assign value a to variable v, then the domain size of w does not decrease: b and d are both possible. Thus for this relation, the algorithm does not have the property that every substitution decreases every domain, and we cannot guarantee a polynomial bound on the recursion tree.

Unfortunately, requiring that every relation is a thick mapping is not sufficient for tractability, as thick mappings can interact with each other in a way that makes CCSP hard. Therefore in order to the problem CCSP(Γ) for a set Γ of thick mappings to be easy, more restrictions have to be imposed on Γ. Such a condition called *noncrossing* requires that if two thick mappings induce equivalence relations α and β on a certain set, then for any equivalence class C of α and a class D of β that are not disjoint, either $C \subseteq D$ or $D \subseteq C$. We need even stronger conditions: not only relations from Γ must be noncrossing thick mappings, but also certain relations derived from them. A detailed explanation is given in the next section.

3. ALGEBRAIC APPROACH

One of the main difficulties in studying problems CSP(Γ) and CCSP(Γ) is: How can one describe or characterize a constraint language (possibly infinite)? A combinatorial characterization is very often impossible, so two alternative approaches have been widely used, one through logic and another one through algebra. Here we use the algebraic one.

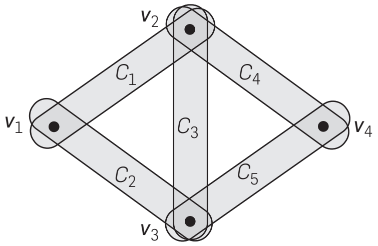
3.1. Primitive positive definitions

In a CSP, possible combinations of values of certain variables can be constrained even if there is no explicit constraint imposed on them, see Figure 6. That is, we can use the constraints in Γ to build "gadgets" that enforce a constraint relation on a certain set of variables. Note that, as in Figure 6, the constraint relation expressed by the gadget does not necessarily belong to Γ. This means that for every constraint language Γ, there is a set of *implicit* constraints that do not belong to Γ, but can still be expressed by instances of CSP(Γ).

How can we characterize all the implicit constraints of a constraint language Γ? It turns out that the implicit constraints that can be expressed in instances of CSP(Γ) admit a simple logic representation. Treating relations in Γ as predicates, one can construct logic formulas from them, and use

Figure 6. Implicit constraints.

Example 3.[8] Let Γ be a constraint language containing a single binary relation R over the set $D = \{0, 1, 2\}$, where R is given by $R = \{(0, 0), (0, 1), (1, 0), (1, 2), (2, 1), (2, 2)\}$. Consider the instance of CSP(Γ) with the set of variables $\{v_1, v_2, v_3, v_4\}$ and set of constraints $\{C_1, C_2, C_3, C_4, C_5\}$, where $C_1 = \langle (v_1, v_2), R\rangle$, $C_2 = \langle (v_1, v_3), R\rangle$, $C_3 = \langle (v_2, v_3), R\rangle$, $C_4 = \langle (v_2, v_4), R\rangle$, $C_5 = \langle (v_3, v_4), R\rangle$. There is no explicit constraint on the pair (v_1, v_4). However, by considering all solutions to the instance, it can be shown that the possible pairs of values which can be taken by this pair of variables are precisely the elements of the relation $R' = R \cup \{(1, 1)\}$. Thus this instance can be considered as a "gadget" implementing R' using only the relations R.



The relation R' can be expressed as the following primitive positive (pp-) definition:

$$R'(x, y) = \exists z, t (R(x, z) \wedge R(x, t) \wedge R(z, t) \wedge R(z, y) \wedge R(t, y)).$$

these formulas to express other predicates (relations). The type of formulas that is just right for representing implicit constraints is called *primitive positive*. Primitive positive (pp-) formulas include predicates from Γ (atomic formulas) and the equality, conjunctions of atomic formulas, and existential quantifiers. Relations (or predicates) that can be expressed by using pp-formulas with predicates from Γ are said to be *pp-definable* in Γ.

Jeavons et al.[23] proved that pp-definitions give rise to reductions between CSPs: If Γ and Δ are constraint languages on the same set such that Δ is finite and every relation in Δ is pp-definable in Γ, then CSP(Δ) is polynomial-time reducible to CSP(Γ) (can be improved to logarithmic-space reducibility). Thus, when proving hardness of CSPs one can use any relations pp-definable in the given constraint language. Very often 'gadgets' used in complexity proofs can be expressed as pp-definitions, so primitive positive definitions generalize and unify gadget reductions.

In CSPs with cardinality constraints, it is not obvious that adding pp-definable relations to the constraint language does not increase hardness. The difficulty is that introducing gadgets (like the one in Figure 6) means adding auxiliary variables, and the values appearing on these variables can affect the cardinality constraints. Nevertheless, we can show that adding a new constraint R' to the constraint language of a CSP with cardinality constraints does not change the complexity if R' is pp-definable without using the equality relation. Relations expressible in such a weaker way are called *pp-definable without equality*. In fact, relations that are pp-definable in a certain Γ with or without equality can only be different by certain redundant parts that are not so important for constraint problems. Therefore, we can essentially assume that Γ is closed under pp-definitions, and hence we can use the algebraic framework discussed in more detail in the next section.

3.2. Polymorphisms and invariants

Although pp-definitions are helpful in hardness proofs, they do not resolve the main difficulty of studying the complexity of CSPs, as they do not help much in describing constraint languages. However, pp-definitions provide a bridge to a tool that allows to do that. *Polymorphisms* can be viewed as a sort of extended symmetries of relations. Let R be a relation on some set D and f a function on the same set that may depend on more than one variable; let f be n-ary, that is, depends on n variables. The function f is a polymorphism of R if for any choice of tuples $\bar{a}_1, \dots, \bar{a}_n$ from R the tuple $f(\bar{a}_1, \dots, \bar{a}_n)$ obtained by component-wise application of f also belongs to R. Relation R in this case is said to be an *invariant* of f. Polymorphisms and invariants naturally extend to constraint languages and functions: A function is a polymorphism of a constraint language if it is a polymorphism of every relation in it, and a relation is an invariant of a set of functions if it is an invariant of every function in the set. For constraint languages Γ, and set of functions C, by Pol Γ we denote the set of all polymorphisms of Γ, and Inv C the set of all invariants of C, see Figure 7.

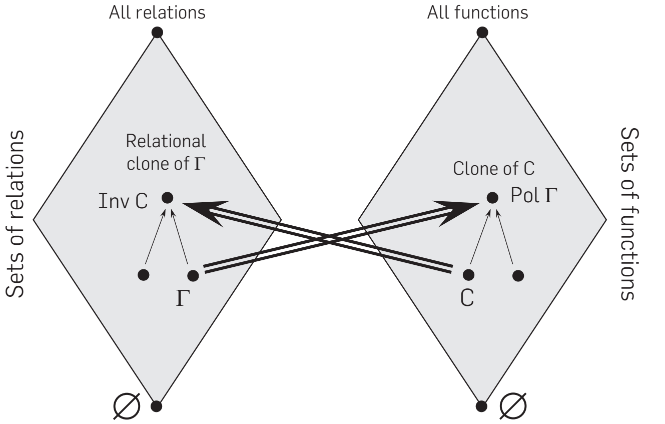
Sets of the form Pol Γ and Inv C have a number of interesting properties, see, e.g., Denecke and Wismath.[16] For any set C of functions Inv C is a *relational clone*, that is, constraint language Δ such that every relation pp-definable in Δ also belongs to Δ. Therefore Jeavons' result (and this paper's analogous result) can be stated in terms of polymorphisms: If Γ and Δ are constraint languages on the same set such that Δ is finite and every polymorphism of Γ is also a polymorphism of Δ, then CSP(Δ) is polynomial-time reducible to CSP(Γ). For CCSP we only have to add the requirement that relations in Δ do not contain redundancies.

For any constraint language Γ the set Pol Γ is a *clone*, that is, a set of functions that contains the identity functions, and closed under compositions. Clones have been a subject of intensive study in algebra for decades; the results of those studies are readily available to be applied to constraint problems.

Clearly, large constraint languages have few polymorphisms. Thus, a number of important properties of relations can be inferred merely from the existence of polymorphisms of certain types. A ternary function h on a set D is said to be *majority function* if $h(x, x, y) = h(x, y, x) = h(y, x, x) = x$ for any $x, y \in D$. If a constraint language has a polymorphism that is a majority function, then the constraint language is 2-decomposable. A ternary operation m is called Maltsev if $m(x, y, y) = m(y, y, x) = x$ for any $x, y \in D$. Any binary relation having a Maltsev polymorphism is a thick mapping, see Figure 8.

For regular CSPs, complexity questions are usually reduced one step further, to universal algebras and their varieties. Most of the strong complexity results about CSPs are obtained this way.[3, 7, 9] Moreover, research on CSP complexity have revolutionized certain fields of algebra, see, e.g., Barto and Kozik.[4] For our result, however, we do not need more algebra than polymorphisms.

Figure 7. Pol and Inv.



3.3. Easy cardinality constraints: The full result

We can finally explain the main result in details. A function f is said to be *conservative* if it always equal to one of its arguments. For instance, a ternary function f is conservative if $f(a, b, c) \in \{a, b, c\}$ for any a, b, c. The main result can be stated compactly the following way:

Main Theorem *Let* Γ *be a finite constraint language. If* Γ *has a majority polymorphism and has a conservative Maltsev polymorphism, then* CCSP(Γ) *is polynomial-time solvable. Otherwise, the problem is NP-complete.*

We can show that if a constraint language Γ satisfies the conditions above, then the problem can be solved in polynomial time by the algorithm presented in Section 2.3. Let Δ be the set of binary relations pp-definable in Γ. Since Γ has a majority polymorphism, it is 2-decomposable; hence, every constraint with a relation $R \in \Gamma$ can be replaced with a collection of binary constraints, the 'projections' of R, which are pp-definable in Γ and thus belong to Δ. Therefore we only need to verify that the Key Property (Section 2.3) always holds. Due to 2-decomposability, Γ can be replaced with Δ. This constraint language has a Maltsev polymorphism, and this makes its relations thick mappings. Suppose now that the graph G of a problem from CCSP(Δ) is connected. For any two variables v, w the set of all allowed combinations of their values is a binary relation, denoted R_{vw} and an implicit constraint. Since Δ contains all binary relations pp-definable in Δ, we have $R_{vw} \in \Delta$. Thus R_{vw} is a thick mapping from D_v to D_w. The connectedness of G and the fact that all relations in Δ are noncrossing can be used to show that R_{vw} is a nontrivial thick mapping. Let α and β be equivalence relations it induces on D_v and D_w, respectively. If we fix a value $a \in D_v$ then the possible values of w are restricted to one equivalence class of β, a proper subset of D_w. As this is true for all variables w, the Key Property follows.

The Main Theorem also leads to a more combinatorial characterization of tractable problems CCSP(Γ): Such a problem is tractable if and only if Γ is 2-decomposable, and the binary relations pp-definable in Γ are noncrossing thick mappings.

Figure 8. Examples of polymorphism.

Majority implies 2-decomposability.
Let R be a ternary relation and h a majority function, which is a polymorphism of R. We show that any triple (a, b, c) such that each of (a, b), (b, c), and (a, c) is extendible to a triple from R, belongs to R. This means the 2-decomposability of R in this case. By the assumption, there are (a, b, z), (a, y, c), $(x, b, c) \in R$ for some x, y, z. Since h is a majority polymorphism of R we have

$$h\begin{pmatrix} a & a & x \\ b & y & b \\ z & c & c \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix},$$

and (a, b, c) belongs to R.

Maltsev implies thick mapping.
Let R be a binary relation and m its Maltsev polymorphism. We have to prove that for any (a, c), (a, d), $(b, c) \in R$ the pair (b, d) also belong to R. It follows from a single application of the Maltsev polymorphism:

$$m\begin{pmatrix} a\,a\,b \\ d\,c\,c \end{pmatrix} = \begin{pmatrix} b \\ d \end{pmatrix}.$$

Linear equations.
As another example of a property of relations expressible by a polymorphism, we consider relations that are solution spaces of systems of linear equations over a finite field F. Then if a relation R has such representation it is an invariant of the *affine* function $f(x, y, z) = x - y + z$, where +, – are operations of the field F. Indeed, let $A \cdot \mathbf{x} = \mathbf{b}$ be the system defining R, and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$.
Then

$$A \cdot f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A \cdot (\mathbf{x} - \mathbf{y} + \mathbf{z}) = A \cdot \mathbf{x} - A \cdot \mathbf{y} + A \cdot \mathbf{z} = \mathbf{b}.$$

In fact, the converse can also be shown: if R is invariant under f then it is the solution space of a certain system of linear equations.

What remains now is to show that otherwise the problem is hard.

4. HARD CSPS WITH CARDINALITY CONSTRAINTS

If one of the three conditions on a constraint language Γ (a) 2-decomposability, (b) all binary pp-definable relations are thick mappings, and (c) all such binary relations are noncrossing does not hold, we show that either Bipartite Independent Set or Linear Equation is reducible to CCSP(Γ), thus showing that CCSP(Γ) is NP-complete. This part is technical, but we outline the intuition behind the technique.

Suppose first that a binary relation R is pp-definable in Γ, but is not a thick mapping. This means that for some a, b, c, d pairs (a, c), (a, d), (b, c) belong to R while (b, d) does not. If a, b, c, d are distinct values, then R contains a fragment that looks like R_{BIS}. We exploit this fact to reduce Bipartite Independent Set to CCSP(Γ) and conclude NP-hardness in this case. In general, it is possible that some of a, b, c, d coincide. However, a case analysis shows that reduction from Bipartite Independent Set is possible in all cases.

If there exist two thick mappings pp-definable in Γ that are not noncrossing, then there are also two equivalence relations with this property; denote them α and β. Since they are not noncrossing, some α-class and some β-class overlap, but are not subsets of one another. Hence for some a, b, c, we have (a, b) is in α but not in β, and (b, c) is in β but not in α. If we can restrict α and β onto $\{a, b, c\}$ somehow, then the product of binary relations $\alpha \circ \beta$ given by a pp-formula $\exists\, z\ \alpha(x, z) \wedge \beta(z, y)$, contains (a, a), (a, c), (c, c), but does not contain (c, a). Again, this fact can be used to reduce Bipartite Independent Set to CCSP(Γ).

Finally, let $R \in \Gamma$ be non-2-decomposable. For simplicity assume R ternary. There is a triple (a, b, c) such that (a, b, z), (a, y, c), (x, b, c) belong to R for some x, y, z, but (a, b, c) does not. We show that either a binary relation which is not a thick mapping can be pp-defined in Γ, or two thick mappings that are not noncrossing, or all the tuples can be chosen such that $a = b = c = 0$, $x = y = z = 1$ (we assume 0 and 1 are elements of the domain we can use here), and R restricted to $\{0, 1\}$ is $R_{\text{ODD-3}}$. Therefore a reduction of Linear Equations to CCSP(Γ) can be found.

5. CONCLUSION

We have completed the study of CSP extended with cardinality constraints, and proved a dichotomy theorem characterizing the complexity of the problem for every constraint language Γ over an arbitrary finite domain D. Dichotomy theorems over non-Boolean domains are notoriously hard to prove, but possibly due to the rather restrictive nature of the CCSP problem, we managed to obtain a complete characterization. One can think of several natural variants with more expressive power, for example, the domain is $\{1, 2, 3, 4\}$, and we have upper bounds on the cardinalities of 1 and 2, while there are lower bounds on the cardinalities of 3 and 4. Therefore, upper and/or lower bounds instead of exact cardinality requirements, bounds only on a subset of values, bounds on the total cardinality of a subset of values, etc. give lots of interesting problems to look at. However, some of these questions seem to be very difficult, as a dichotomy result would immediately imply the Feder–Vardi Dichotomy Conjecture (after all, we do not fully understand CSP even *without* cardinality constraints).

Another natural direction is to consider optimization variants (minimize/maximize the number of times certain values appear) and determine the approximability of the resulting problems. In the Boolean case, the approximability of the MinOnes/MaxOnes problems, where the task is to find a satisfying assignment minimizing/maximizing the number of variables receiving value 1, was classified by Khanna et al.[25] Again, not being able to solve the Feder–Vardi conjecture limits what immediate progress we can expect in the study of non-Boolean domains.

Finally, one can look at CCSP from the viewpoint of parameterized complexity. The basic issues of parameterized complexity is whether an algorithm of running time $f(k) \cdot n^c$ exists, where k is some parameter of the input (for example, the size of the solution we are looking for), $f(k)$ is an arbitrary function depending on k, and c is a universal constant independent of k. For example, in Boolean CCSP, one can answer in time $n^{O(k)}$ whether there is a solution with exactly k variables set to 1, but it would be preferable to find an algorithm with running time of the form $f(k) \cdot n^c$, that is, where the combinatorial explosion is restricted to k and the exponent of n is independent of k. We can ask what those Boolean constraint languages Γ are for which the problem of finding a solution with exactly/at most/at least k variables having 1 can be solved in such running time. These questions have been investigated and completely answered in Kratsch et al. and Marx.[27, 30] Generalization of some of these results to arbitrary non-Boolean domains have been obtained very recently by the authors.[11]

References

1. Allen, J. *Natural Language Understanding*. Benjamin Cummihgs, 1994.
2. Allender, E., Bauland, M., Immerman, N., Schnoor, H., Vollmer, H. The complexity of satisfiability problems: Refining Schaefer's theorem. *J. Comput. Syst. Sci. 75*, 4 (2009), 245–254.
3. Barto, L., Kozik, M. Constraint satisfaction problems of bounded width. In *FOCS* (2009), 595–603.
4. Barto, L., Kozik, M. New conditions for Taylor varieties and CSP. In *LICS*, 2010. to appear.
5. Bazgan, C., Karpinski, M. On the complexity of global constraint satisfaction. In *ISAAC* (2005), 624–633.
6. Bessière. C., Hebrard, E., Hnich, B., Walsh, T. The complexity of global constraints. In *AAAI* (2004), 112–117.
7. Bulatov, A. Tractable conservative constraint satisfaction problems. In *LICS* (2003), 321–330.
8. Bulatov, A., Jeavons, P., Krokhin, A. Functions of multiple-valued logic and the complexity of constraint satisfaction: A short survey. In *ISMVL* (2003), 343–351.
9. Bulatov, A.A. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM 53*, 1 (2006), 66–120.
10. Bulatov, A.A., Krokhin, A.A., Larose, B. Dualities for constraint satisfaction problems. In *Complexity of Constraints* (2008), 93–124.
11. Bulatov, A.A., Marx, D. Constraint satisfaction parameterized by solution size. Manuscript.
12. Bulatov, A.A., Marx, D. The complexity of global cardinality constraints. In *LICS* (2009), 419–428.
13. Bulatov, A.A., Valeriote, M. Recent results on the algebraic approach to the CSP. In *Complexity of Constraints* (2008), 68–92.
14. Creignou, N., Schnoor, H., Schnoor, I. Non-uniform boolean constraint satisfaction problems with cardinality constraint. In *CSL* (2008), 109–123.
15. Dechter, R. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
16. Denecke, K., Wismath, S. *Universal Algebra and Applications in Theoretical Computer Science*. Chapman and Hall, CRC Press, 2002.
17. Dunkin, N., Bater, J., Jeavons, P., Cohen, D. Toward high order constraint represenations for the frequency assignment problem. Technical Report CSD-TR-98–05, Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, UK, 1998.
18. Feder, T., Vardi, M. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput. 28* (1998), 57–104.
19. Freuder, E. Synthesizing constraint expressions. *Commun. ACM 21*, (1978) 958–966.
20. Freuder, E.C. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of AAAI-90* (Boston, MA, 1990), 4–9.
21. Grohe, M. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM 54*, 1 (2007).
22. Hell, P., Nesetril, J. Colouring, constraint satisfaction, and complexity. *Comput. Sci. Rev. 2*, 3 (2008), 143–163.
23. Jeavons, P., Cohen, D., Gyssens, M. Closure properties of constraints. *J.ACM 44* (1997), 527–548.
24. Kautz, H.A., Selman, B. Planning as satisfiability. In *ECAI* (1992), 359–363.
25. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P. The approximability of constraint satisfaction problems. *SIAM J. Comput. 30*, 6 (2001) 1863–1920.
26. Kolaitis, P., Vardi, M. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci. 61* (2000), 302–332.

27 Kratsch, S., Marx, D., Wahlström, M. Parameterized complexity and kernelizability of Max Ones and Exact Ones problems. Submitted, 2010.

28. Ladner, R. On the structure of polynomial time reducibility. *J. ACM 22* (1975), 155–171.
29. Larose, B., Tesson, P. Universal algebra and hardness results for constraint satisfaction problems. In *ICALP* (2007), 267–278.
30. Marx, D. Parameterized complexity of constraint satisfaction problems. *Comput. Complex. 14*, 2 (2005), 153–183. Special issue "Conference on Computational Complexity (CCC) 2004".
31. Marx, D. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC* (2010), to appear.
32. Montanari, U. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci. 7* (1974), 95–132.
33. Nadel, B. Constraint satisfaction in Prolog: Complexity and theory-based heuristics. *Inf. Sci. 83*, 3–4 (1995), 113–131.
34. Schaefer, T. The complexity of satisfiability problems. In *STOC* (1978), 216–226.
35. van Beek, P. Reasoning about qualitative temporal information. *Artif. Intell. 58* (1992), 297–326.

Andrei A. Bulatov (abulatov@cs.sfu.ca), School of Computing Science, Simon Fraser Univerity, 8888 University Drive, Burnaby, BC, Canada.

Dániel Marx (dmarx@cs.bme.hu), School of Computer Science, Tel Aviv University, Tel Aviv, Israel.