On the Minimal Constraint Satisfaction Problem: Complexity and Generation

Guillaume Escamocher(✉) and Barry O'Sullivan

Department of Computer Science, Insight Centre for Data Analytics,
University College Cork, Cork, Ireland
{guillaume.escamocher,barry.osullivan}@insight-centre.org

Abstract. The Minimal Constraint Satisfaction Problem, or Minimal CSP for short, arises in a number of real-world applications, most notably in constraint-based product configuration. Despite its very permissive structure, it is NP-hard, even when bounding the size of the domains by $d \geq 9$. Yet very little is known about the Minimal CSP beyond that. Our contribution through this paper is twofold. Firstly, we generalize the complexity result to any value of d. We prove that the Minimal CSP remains NP-hard for $d \geq 3$, as well as for $d = 2$ if the arity k of the instances is strictly greater than 2. Our complexity result can be seen as providing a *dichotomy theorem* for the Minimal CSP. Secondly, we build a generator that can create Minimal CSP instances of any size, using the constrainedness as a parameter. Our generator can be used to study behaviors that are typical of NP-hard problems, such as the presence of a phase transition, in the case of the Minimal CSP.

1 Introduction

An instance of the Minimal Constraint Satisfaction Problem, or Minimal CSP for short, is a CSP instance where all allowed k-tuples are part of at least one solution, with k being the arity of the instance, that is the size of the scope of the constraints. Since all Minimal CSP instances are satisfiable, solving such an instance does not refer to the decision problem of determining whether it has a solution, but to the exemplification of a solution.

Minimal CSP is often found 'naturally' in configuration problems [8]. A seller might want to offer its customers a large degree of customization. If, for example, the product sold is a car, some possible options might be the color of the vehicle and whether it is automatic or manual. If after choosing "automatic", "red" remains a valid option for the color parameter, then it is preferable that at least one red automatic car can be configured. The Minimal CSP can answer a number of queries relevant to product configuration in polynomial time [6], such as whether a solution exists that satisfies a given unary constraint, or whether an assignment to k variables is consistent in a Minimal CSP where all constraints are defined over k-tuples of the variables. These queries can be answered simply by inspecting the constraints of the problem instance. However, answering queries over arbitrary assignments to the variables remains hard, which has given rise

Z. Lu et al. (Eds.): COCOA 2015, LNCS 9486, pp. 731–745, 2015.
DOI: 10.1007/978-3-319-26626-8_54

to many studies of the use of automata and decision diagrams to reason about the solution sets of complex configuration problems [2].

Gottlob has shown that computing a solution to a binary Minimal CSP is NP-Hard [6]. While considerable work has been done on the study of problem hardness and instance generation for many classes of CSPs [1,10,11], nothing has been done in the case of Minimal CSPs, which are an interesting class of problem in practice. In this paper, we show that when bounding the size of the domains by a constant d and the arity of the constraints by a constant k, Minimal CSP and the general CSP are NP-hard for the exact same values of d and k. We also present an algorithm that can be used to generate Minimal CSP instances of any size, while retaining control over some parameters like the tightness of the constraints. Since tightness often reveals many statistical properties of CSPs, like for example phase transition behaviors, many empirical analysis of such problems use it as a parameter [3,5]. We tested our generator to determine if it was refined enough to detect the diverse properties held by the Minimal CSP, and found that it could indeed expose how the Minimal CSP behaves: like most other NP-hard problems [7], Minimal CSP exhibits an easy-hard-easy pattern as constraint tightness is varied.

The remainder of the paper is organized as follows. In the next Section, we formally define the Minimal CSP and present our complexity results, and in particular a dichotomy theorem, that generalizes Gottlob's complexity result [6] to instances with very small domain sizes. In Sect. 3.1 we provide some preliminary definitions needed for the description of our generator. Then in Sect. 3.2, we present the algorithm itself, which is able to generate in an efficient way Minimal CSP instances of a given size; tightness, that is the average number of incompatibilities in a constraint, is used as a parameter. In Sect. 3.3, we show some empirical results that we obtained when solving Minimal CSP instances generated with this method, revealing the presence of a peak of difficulty. Finally, we conclude by summarizing our contributions and outlining some future work in this area.

2 The Minimal CSP: Definitions and Complexity

2.1 Definitions

We recall the definition of the Constraint Satisfaction Problem, or CSP.

Definition 1 (CSP). *A* CSP instance I *comprises:*

1. *A set* $V = \{v_1, \ldots, v_n\}$ *of* n *variables.*
2. *A set* $A = \{A_{v_1}, \ldots, A_{v_n}\}$ *of* n *domains. For all* $i \in [1, n]$, $A_{v_i} = \{a_1, \ldots, a_{d_i}\}$ *contains the* d_i *possible values for the variable* v_i.
3. *A set* $C = \{C_1, \ldots, C_m\}$ *of* m *constraints. To each constraint* C_i *is associated a scope* $W_i = \{w_1, \ldots, w_{k_i}\} \subseteq V$ *and a set* U_i *of* k_i*-tuples from* $[A_{w_1}] \times [A_{w_2}] \times \cdots \times [A_{w_{k_i}}]$. *We say that these tuples are* allowed, *or* compatible, *for the scope* W_i, *that the tuples from* $[A_{w_1}] \times [A_{w_2}] \times \cdots \times [A_{w_{k_i}}]$ *that are not in* U_i *are*

forbidden, *or* incompatible, *for the scope* W_i *and that* k_i *is the* arity *of the constraint* C_i. *Let* $C_j \in C$ *be the constraint with the highest arity. We say that the arity of* C_j *is the arity of the instance.*

A partial solution *to* I *on* $W = \{w_1, \ldots, w_r\} \subseteq V$ *is a* r*-tuple* $b = \{b_1, \ldots, b_r\}$ *such that* $\forall i \in [1, r]$, $b_i \in A_{w_i}$ *and* $\forall j \in [1, m]$, *such that* k_j *is the arity of* C_j, *there is no* k_j*-tuple* $u \subseteq b$ *that is incompatible on* S_j. *A* solution *to* I *is a partial solution on* V.

We now formally define the Minimal Constraint Satisfaction Problem, or Minimal CSP.

Definition 2 (Minimal CSP). *A CSP instance* $I = \{V, A, C\}$ *is a* Minimal CSP instance *if and only if:* $\forall C_i \in C$, $\forall u$ *such that* u *is a compatible tuple on the scope of* C_i, *there is at least one solution* S *to* I *such that* S *contains* u.

In this paper, we only consider Minimal CSP instances with non-empty domains and such that each value in each domain belongs to at least one compatible tuple.

2.2 Complexity

Not only are Minimal CSP instances always satisfiable, but they typically contain many solutions. A Minimal CSP instance will contain as least as many solutions as there are allowed k-tuples in its least constrained constraint. For this reason, Minimal CSP instances will often be very trivial to solve. Yet, computing a solution to a Minimal CSP instance is NP-hard [6].[1] The proof given in [6] is a reduction from 3-SAT to a set of CSP instances M_9 such that for each instance $I \in M_9$:

- I is either a Minimal CSP instance or unsatisfiable.
- I contains at most 9 values in each domain.

This is stronger than just NP-hardness, the actual result in [6] is that computing a solution to a Minimal CSP instance is NP-hard, even when bounding the size of the domains by a fixed integer $d \geq 9$. We now further generalize this result to any $d \geq 3$:

Proposition 1. *Computing a solution to a Minimal CSP instance is NP-hard, even when bounding the size of the domains by a fixed integer* $d \geq 3$.

Proof. Suppose that we have a k-ary Minimal CSP instance I. Let v be a variable in I such that the domain of v is of size $d_v > 3$. We replace v by two new variables v_1 and v_2 to obtain a new instance I' defined as follow:

[1] We are aware that finding a solution to a Minimal CSP is both a *search* problem (find a solution) and a *promise* problem (the input CSP is satisfiable because it is minimal), rather that a *decision* problem. However, we use this terminology in the same manner as Gottlob [6], where a thorough discussion of the matter can be found.

1. With $\{a_1, a_2, \ldots, a_{d_v}\}$ being the domain of v, we set the domain of v_1 to $\{a_1, a_2, x\}$ and the domain of v_2 to $\{\overline{x}, a_3, \ldots, a_{d_v}\}$.
2. Let $B = \{b_1, b_2, b_3, \ldots, b_k\}$ be a k-tuple such that b_1 is in the domain of v_1 and b_2 is in the domain of v_2. Then B is compatible if and only if ($b_1 = x$, $b_2 \neq \overline{x}$ and there exists some value b' such that the k-tuple $B' = \{b', b_2, b_3, \ldots, b_k\}$ is compatible in I) or ($b_1 \neq x$, $b_2 = \overline{x}$ and there exists some value b' such that the k-tuple $B' = \{b_1, b', b_3, \ldots, b_k\}$ is compatible in I).
3. Let b be a value not in the domain of v_1 or v_2. Let $B = \{b, b', b_3, \ldots, b_k\}$ be a k-tuple such that b' is in the domain of v_1 and B does not contain any value from the domain of v_2. If $b' = a_i$ for $1 \leq i \leq 2$, then B is compatible in I' if and only if B was compatible in I. If $b' = x$, let $B_i = \{b, a_i, b_3, \ldots, b_k\}$ for $3 \leq i \leq d_v$. B is compatible in I' if and only if one of the B_i was compatible in I.
4. Let b be a value not in the domain of v_1 or v_2. Let $B = \{b, b', b_3, \ldots, b_k\}$ be a k-tuple such that b' is in the domain of v_2 and B does not contain any value from the domain of v_1. If $b' = a_i$ for $3 \leq i \leq d_v$, then B is compatible in I' if and only if B was compatible in I. If $b' = x$, let $B_i = \{b, a_i, b_3, \ldots, b_k\}$ for $1 \leq i \leq 2$. B is compatible in I' if and only if one of the B_i was compatible in I.
5. All other k-tuples in I' remain as they were in I.

Figure 1 illustrates an example of the transformation for $k = 2$ and $d_v = 4$. A_v, A_{v_1} and A_{v_2} denote the domains of v, v_1 and v_2 respectively. The continuous lines represent compatibility edges, while the dashed lines represent incompatibility edges.

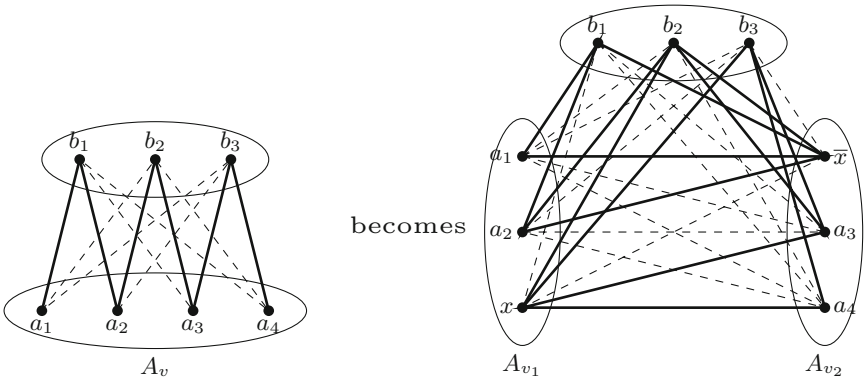


Fig. 1. Transforming a variable v into two variables v_1 and v_2 with smaller domains.

In order to obtain the desired result, we have to prove that I' is a Minimal CSP instance and that finding a solution for I' allows us to find in polynomial time a solution for I.

1. I' is a Minimal CSP instance:
   We have to prove that each compatible k-tuple in I' is in a solution for I'. Let $B = \{b_1, b_2, b_3, \ldots, b_k\}$ be a compatible k-tuple in I' such that:
   - Either b_1 is in the domain A_{v_1} of v_1, or B does not contain any value from A_{v_1}.
   - Either b_2 is in the domain A_{v_2} of v_2, or B does not contain any value from A_{v_2}.

   We necessarily have one of the following seven cases:
   (a) $b_1 = x$ and $b_2 = a_i$ for some $i \in [3, \ldots, d_v]$: from the second point in the definition of I', we know that there is some compatible k-tuple $B' = \{b', a_i, b_3, \ldots, b_k\}$ in I. Since I is a Minimal CSP instance, there is some solution S for I such that B' belongs to S. Let $S' = S \cup \{x\}$. By construction, S' is a solution for I' containing B.
   (b) $b_1 = a_i$ for some $i \in [1, 2]$ and $b_2 = \overline{x}$: same argument as for (a).
   (c) $b_1 = x$ and $b_2 \notin A_{v_2}$: from the third point in the definition of I', we know that there is some compatible k-tuple $B_i = \{a_i, b_2, b_3, \ldots, b_k\}$ in I, with $i \in [3, \ldots, d_v]$. Since I is a Minimal CSP instance, there is some solution S for I such that B_i belongs to S. Let $S' = S \cup \{x\}$. By construction, S' is a solution for I' containing B.
   (d) $b_1 \notin A_{v_1}$ and $b_2 = \overline{x}$: same argument as for (c), using the fourth point in the definition of I' instead of the third.
   (e) $b_1 \in A_{v_1}$, $b_1 \neq x$ and $b_2 \notin A_{v_2}$: from the third point in the definition of I', we know that B is compatible in I too. Since I is a Minimal CSP instance, there is some solution S for I such that B belongs to S. Let $S' = S \cup \{x\}$. By construction, S' is a solution for I' containing B.
   (f) $b_1 \notin A_{v_1}$, $b_2 \in A_{v_2}$ and $b_2 \neq \overline{x}$: same argument as for (e), using the fourth point in the definition of I' instead of the third.
   (g) $b_1 \notin A_{v_1}$ and $b_2 \notin A_{v_2}$: from the fifth point in the definition of I', we know that B is compatible in I too. Since I is a Minimal CSP instance, there is some solution S for I such that B belongs to S. Let a be the point of S in A_v. If $a = a_1$ or $a = a_2$, then let $S' = S \cup \{\overline{x}\}$. Otherwise, let $S' = S \cup \{x\}$. By construction, S' is a solution for I' containing B.

   So if B is a compatible k-tuple of I', then B is in a solution for I'.
2. Finding a solution for I from a solution for I':
   Suppose that we have a solution S' for I'. From the second point in the definition of I', we know that there is no compatible k-tuple containing both x and $\overline{x}$. Therefore, S' must contain one of the a_i. Let $S = S' \backslash \{x\}$ (or $S = S' \backslash \{\overline{x}\}$ if $\overline{x} \in S'$). By construction, S is a solution for I.

The domain size of v_1 is $3 < d_v$, and the domain size of v_2 is $d_v - 1 < d_v$. Therefore, if we have a variable with a domain of size strictly greater than 3, then we can replace it by two variables with strictly smaller domain size. By iteratively applying this operation until all domains have a size of 3 or less, we can reduce I to a Minimal CSP instance where the size of all domains is bounded by 3. Since computing a solution to a Minimal CSP instance is NP-hard [6], we have the result. □

In the case of Boolean domains, we also have NP-hardness if the arity of the instances is $k \geq 3$.

Proposition 2. *For all $k > 2$, k-ary Minimal CSP is NP-hard, even when bounding the size of the domains by 2.*

The proof is based on the transformation from a domain $A = \{a_1, a_2, a_3\}$ of size 3 to three domains $A_1 = \{a_1, \overline{a_1}\}$, $A_2 = \{a_2, \overline{a_2}\}$, $A_3 = \{a_3, \overline{a_3}\}$, each of size 2.

Proof. Let $k > 2$. From Proposition 1, we know that computing a solution to a k-ary Minimal CSP instance is NP-hard, even when bounding the size of the domains by 3. Therefore, we just need to reduce the k-ary Minimal CSP with domain size bounded by 3 to the k-ary Minimal CSP with domain size bounded by 2. Let I be a k-ary Minimal CSP instance such that the size of each domain in I is bounded by 3. Let v be a variable in I such that the domain of v is of size 3. We replace v by three new variables v_1, v_2 and v_3 to obtain a new instance I' defined as follows:

1. With $\{a_1, a_2, a_3\}$ being the domain of v, we set the domain of v_i to be $A_{v_i} = \{a_i, \overline{a_i}\}$ for $1 \leq i \leq 3$.
2. All k-tuples in I' containing both a_i and a_j for some $1 \leq i \neq j \leq 3$ are incompatible.
3. Let $B = \{\overline{a_i}, \overline{a_j}, b_1, b_2, \ldots, b_{k-2}\}$ be a k-tuple for some $1 \leq i \neq j \leq 3$. Let h be the integer between 1 and 3 such that $h \neq i$ and $h \neq j$. B is compatible if and only if there is some compatible k-tuple in I containing a_h and all the b_g for $1 \leq g \leq k-2$. Note that this covers the particular cases when a_h is one of the b_g (in which case B is compatible if and only if there is a compatible k-tuple in I containing all the b_g) and when $\overline{a_h}$ is one of the b_g (in which case B is incompatible because $\overline{a_h}$ does not appear in I).
4. Let $B = \{a_i, b_1, b_2, \ldots, b_{k-1}\}$ be a k-tuple such that $1 \leq i \leq 3$ and no b_j is in the domain of v_1, v_2 or v_3. B is compatible in I' if and only if B is compatible in I.
5. Let $B = \{\overline{a_i}, b_1, b_2, \ldots, b_{k-1}\}$ be a k-tuple such that $1 \leq i \leq 3$ and no b_j is in the domain of v_1, v_2 or v_3. B is compatible if and only if there is some $j \neq i$, with $1 \leq j \leq 3$, such that $\{a_j, b_1, b_2, \ldots, b_{k-1}\}$ is compatible in I.
6. Let $B = \{a_i, \overline{a_j}, b_1, b_2, \ldots, b_{k-2}\}$ be a k-tuple such that $1 \leq i \neq j \leq 3$ and no b_h is in the domain of v_1, v_2, or v_3. B is compatible if and only if there is some b_{k-1} such that $\{a_i, b_{k-1}, b_1, b_2, \ldots, b_{k-2}\}$ is compatible in I.
7. All other k-tuples in I' remain as they were in I.

In order to obtain the desired result, we have to prove that I' is a Minimal CSP instance and that finding a solution for I' allows us to find in polynomial time a solution for I.

1. I' is a Minimal CSP instance:
   We have to prove that each compatible k-tuple in I' is in a solution for I'. Let B be a compatible k-tuple in I'. We necesarily have one of the six following cases:

(a) $B = \{a_i, b_1, b_2, \ldots, b_{k-1}\}$ with $1 \leq i \leq 3$ and neither b_j in the domain of v_1, v_2 or v_3. Without loss of generality, we assume that $i = 1$. From the fourth point in the definition of I', we know that B is also compatible in I. Since I is a Minimal CSP instance, there is some solution S for I such that B belongs to S. Let $S' = S \cup \{\overline{a_2}, \overline{a_3}\}$. By construction, S' is a solution for I' containing B.

(b) $B = \{a_i, \overline{a_j}, b_1, b_2, \ldots, b_{k-2}\}$ with $1 \leq i \neq j \leq 3$ and no b_h in the domain of v_1, v_2 or v_3. Without loss of generality, we assume that $i = 1$. From the sixth point in the definition of I', we know that there is a compatible triple in I containing a_1 and all b_h for $1 \leq h \leq k-1$. Since I is a Minimal CSP instance, there is some solution S for I such that both a_1 and all the b_h belong to S. Let $S' = S \cup \{\overline{a_2}, \overline{a_3}\}$. By construction, S' is a solution for I' containing B.

(c) $B = \{a_i, \overline{a_j}, \overline{a_h}, b_1, b_2, \ldots, b_{k-3}\}$ with $1 \leq i, j, h \leq 3$ and i, j, h all distinct. Without loss of generality, we assume that $i = 1$. From the third point in the definition of I', we know that there is a compatible k-tuple in I containing a_1 and all the b_g for $1 \leq g \leq k-3$. Since I is a Minimal CSP instance, there is a solution S for I such that a_i and all the b_g belong to S. Let $S' = S \cup \{\overline{a_2}, \overline{a_3}\}$. By construction, S' is a solution for I' containing B.

(d) $B = \{\overline{a_i}, b_1, b_2, \ldots, b_{k-1}\}$ with $1 \leq i \leq 3$ and no b_j in the domain of v_1, v_2 or v_3. Without loss of generality, we assume that $i = 1$. From the fifth point in the definition of I', either $\{a_2, b_1, b_2, \ldots, b_{k-1}\}$ or $\{a_3, b_1, b_2, \ldots, b_{k-1}\}$ is compatible in I. Without loss of generality, we assume the former. Since I is a Minimal CSP instance, there is some solution S for I such that $\{a_2, b_1, b_2, dots, b_{k-1}\}$ belongs to S. Let $S' = S \cup \{\overline{a_1}, \overline{a_3}\}$. By construction, S' is a solution for I' containing B.

(e) $B = \{\overline{a_i}, \overline{a_j}, b_1, b_2, \ldots, b_{k-2}\}$ with $1 \leq i \neq j \leq 3$ and no b_h not in the domain of v_1, v_2 or v_3. Without loss of generality, we assume that $i = 1$ and $j = 2$. From the third point in the definition of I', there is a compatible triple in I containing a_3 and all the b_h for $1 \leq h \leq k-2$. Since I is a Minimal CSP instance, there is some solution S for I such that a_3 and all the b_h belong to S. Let $S' = S \cup \{\overline{a_1}, \overline{a_2}\}$. By construction, S' is a solution for I' containing B.

(f) $B = \{b_1, b_2, \ldots, b_k\}$ with no b_i being in the domain of v_j, for any $1 \leq i, j \leq 3$. From the seventh point in the definition of I', we know that B is compatible in I too. Since I is a Minimal CSP instance, there is some solution S for I such that B belongs to S. Let a_i be the point of S in the domain of v. Without loss of generality, we assume that $i = 1$. Let $S' = S \cup \{\overline{a_2}, \overline{a_3}\}$. By construction, S' is a solution for I' containing B.

So if B is a compatible k-tuple of I', then B is in a solution for I'.

2. Finding a solution for I from a solution for I':
Suppose that we have a solution S' for I'. From the second and third points in the definition of I', we know that S' must contain a_i, $\overline{a_j}$ and $\overline{a_h}$, for some distinct i, j and h between 1 and 3. Without loss of generality, we assume that $i = 1$. Let $S = S' \backslash \{\overline{a_2}, \overline{a_3}\}$. By construction, S is a solution for I.

Therefore, if we have a variable with a domain of size 3, then we can replace it by three variables with domains of size 2. By iteratively applying this operation until all domains have a size of 2 or less, we can reduce I to a Minimal CSP instance where the size of all domains is bounded by 2. So we have the result. □

The binary Boolean Minimal CSP is polynomial, since the more general binary Boolean CSP can be trivially reduced to 2-SAT, which is polynomial [9]. Combined with Propositions 1 and 2, and with the triviality of CSP instances consisting entirely of single-valued variables, we have the main Theorem:

Theorem 1 (The Minimal CSP Dichotomy Theorem). *k-ary Minimal CSP when the size of the domains is bounded by d is NP-hard if and only if ($d \geq 3$ or ($d = 2$ and $k \geq 3$)).*

The result is summarized in Table 1.

Table 1. Complexity of the k-ary Minimal CSP with domains of size bounded by d.

d \ k	2	≥ 3
1	tractable	tractable
2	tractable	NP-hard
≥ 3	NP-hard	NP-hard

Corollary 1. *When bounding the size of the domains by a constant d and the arity of the constraints by a constant k, Minimal CSP and the general CSP are NP-hard for the exact same values of d and k.*

3 Generating Minimal CSP Instances

3.1 Preliminary Notions

The unique properties defining Minimal CSP instances are strongly global. Random general CSP instances can be generated constraint after constraint. Even random satisfiable CSP instances, which are also defined by a global property, still have considerable leeway for local modifications [1]. In contrast, a slight change of one given constraint in a Minimal CSP instance can jeopardize its minimality. Therefore, it is not as straightforward to generate Minimal CSP instances, compared to many other kinds of CSP instances.

Since the NP-hardness proofs for the Minimal CSP are all constructive, what would appear to be a method of creating Minimal CSP instances is to reduce general CSP instances to Minimal CSP instances. Unfortunately, this results in instances that are far too large. For example, the reduction used in [6] transforms satisfiable 3-SAT instances with n clauses into Minimal CSP instances with

$1000n$ variables of domain size 9. Therefore, it is not practical to look for hard Minimal CSP instances this way.

Another intuitive idea would be to minimalize random satisfiable CSP instances, that is to only keep the compatibilities of a given satisfiable CSP that are in a solution. Indeed, every satisfiable CSP instance contains a unique minimalized version of itself. However, this approach is neither practical nor efficient. While it avoids too large instances, it does not allow for an effective control of the size of the instances. In particular, if the original satisfiable CSP instance has only very few solutions, then the resulting Minimal CSP instance will be very small, and its resolution trivial. Furthermore, minimalizing an arbitrary CSP instance is NP-hard [6]. Since the output of minimalization is not certain and the effort required to minimalize an instance is too high, another generation method is needed.

Since in a Minimal CSP instance every single compatibility is part of a solution, any Minimal CSP instance I's compatibilities can be defined by a set S of solutions, such that each solution from S is a solution to I, and any compatibility of I belongs to one of the solutions in S. The solutions from S can be intersecting, and for a given Minimal CSP instance I, the set S is in general not unique. The general intuition behind our algorithm is to start with a Minimal CSP instance with the fewest possible number of solutions in the set S, then add solutions to the set until we reach the desired constrainedness. We now define some notions that we will be using when describing our generator.

Definition 3 (CSP size when domains are equal). *Let I be a k-ary CSP instance with n variables, such that the domain of each variable contains exactly d values. Then we say that I is* of size (d, n, k).

When the domains of the variables do not all have the same size, we have the following definition:

Definition 4 (CSP size when domains are different). *Let I be a k-ary CSP instance with n variables, such that the domain of each variable contains at most d values. Then we say that I is* of size (d^-, n, k).

Consider Minimal CSP instances of size (d, n, k) for some given d, n and k. Without loss of generality, we can assume that the domain of each variable contains the integers 1 through d. The Minimal CSP instance I of this size with the fewest compatibilities will be the one defined by a set $S = \{s_1, \ldots, s_d\}$ of d solutions, such that for each i, s_i is the solution where all variables take the value i. In this particular case, the set S defining I is unique, and the total number of solutions to I is the same as the number of solutions in S, d. The tightness of I is $\frac{d^k - d}{d^k} = 1 - \frac{1}{d^{k-1}}$, because there are $d^k - d$ incompatibilities in each constraint.

Definition 5 (Bare Minimal CSP when domains are equal). *Let I be a CSP instance of size (d, n, k). We say that I is a* bare *Minimal CSP instance if for all k-tuple of variables $C = \{v_{i_1}, \ldots, v_{i_k}\}$, for all k-tuple of assignments $A = \{(v_{i_1}, a_{i_1}), \ldots, (v_{i_k}, a_{i_k})\}$, A is compatible if and only if $a_{i_j} = a_{i_{j'}}$ for all $1 \leq j < j' \leq n$.*

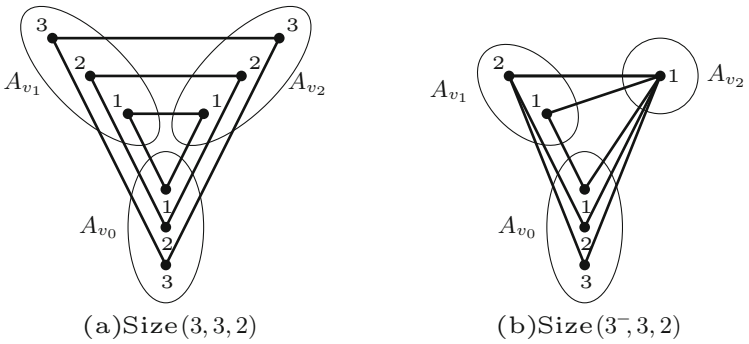


Fig. 2. Two examples of bare Minimal CSP instances.

An example of a bare binary Minimal CSP instance with three variables v_0, v_1 and v_2 and with all domains A_{v_0}, A_{v_1} and A_{v_2} of same size $d = 3$ is given in Fig. 2(a). In this figure, continuous lines represent compatibilities, and if there is no straight line between two values then the values are incompatible.

The notion of bare Minimal CSP instance can be generalized to CSP instances where the domains of the variables have different sizes.

Definition 6 (Bare Minimal CSP when domains are different). *Let I be a CSP instance of size (d^-, n, k). We say that I is a* bare *Minimal CSP instance if for all k-tuple of variables $C = \{v_{i_1}, \ldots, v_{i_k}\}$, for all k-tuple of assignments $A = \{(v_{i_1}, a_{i_1}), \ldots, (v_{i_k}, a_{i_k})\}$, A is compatible if and only if for all $1 \leq j < j' \leq n$ at least one of the following is true:*

- $a_{i_j} = a_{i_{j'}}$
- $a_{i_j} \leq a_{i_{j'}}$ *and the domain of* v_{i_j} *is of size* a_{i_j}
- $a_{i_j} \geq a_{i_{j'}}$ *and the domain of* $v_{i_{j'}}$ *is of size* $a_{i_{j'}}$

By noticing that in such a Minimal CSP instance the number of compatibilities in each constraint is equal to the size of the largest domain, it is easy to verify that a bare Minimal CSP instance of size (d^-, n, k) has the fewest possible compatibilities for a Minimal CSP instance of this size.

An example of bare binary Minimal CSP instance with three variables v_0, v_1 and v_2 and with the associated domains A_{v_0}, A_{v_1} and A_{v_2} of different sizes is given in Fig. 2(b). Like in Fig. 2(a), continuous lines represent compatibilities, and if there is no straight line between two values then the values are incompatible.

3.2 The Generator

Suppose that we want a Minimal CSP instance of size (d, n, k), such that the average tightness of each constraint is t, for some given t. What we do is generate a bare Minimal CSP instance I of this size, then we add solutions to I until we have the desired tightness. This is illustrated by Algorithm 1.

Data: d, n, k integers, $t \in [0, 1]$
Result: A Minimal CSP instance of size (d, n, k), and with a tightness equal to $t' \in [t - \frac{1}{d^k}, t]$.
Generate a bare Minimal CSP instance I;
$t_i \leftarrow$ tightness of I;
while $t_i > t$ **do**
  Add solution to I;
  $t_i \leftarrow$ tightness of I;
end
return I

Algorithm 1. Minimal CSP Instance Generator

Data: d, n, k integers, $t \in [0, 1]$
Result: A Minimal CSP instance of size (d, n, k), and with a tightness equal to $t' \in [t - \frac{1}{d^k}, t]$.
Generate a bare Minimal CSP instance I;
for $i \leftarrow 1$ **to** n **do**
  randomize the ordering of the values in the domain of v_i;
end
$t_I \leftarrow$ tightness of I;
while $t_I > t$ **do**
  $C = \{v_{i_1}, \ldots, v_{i_k}\} \leftarrow$ most constrained constraint in I;
  $A = \{v_{i_1} : a_{i_1}, \ldots, v_{i_k} : a_{i_k}\} \leftarrow$ random incompatibility k-tuple in C;
  Add solution containing A to I;
  $t_I \leftarrow$ tightness of I;
end
return I

Algorithm 2. Improved Minimal CSP Instance Generator

A few adjustments must be made to the algorithm in order to obtain the optimal result. Specificically, we must randomize the order of the values in each domain after generating the bare Minimal CSP instance, so that looking at the first instantiation in the lexicographical order does not yield a trivial solution. Also, the solution that is added at each iteration of the algorithm must be built around one of the incompatibilities in the most constrained constraint. This serves two purposes. Firstly, building a solution around an incompatibility ensures that the number of compatibilities strictly increases at each iteration, so that no iteration is wasted and the algorithm always terminates. Secondly, choosing the most constrained constraint ensures that the tightness of each constraint is as balanced as possible. Algorithm 2 illustrates the improvements made to the original algorithm.

Adding one solution to an instance can, and will in the first iterations, add several compatibilities at once. Therefore, the tightness of the resulting instance will not always be exactly equal to the desired value t. However, adding one solution only adds at most one compatibility per constraint. Therefore, in the worst case, the tightness of the resulting instance will deviate from t by $\frac{1}{d^k}$. The

number of compatibilities added at each iteration will decrease along with the tightness. During the first iterations, adding one solution will add one compatibility in most constraints. If the desired tightness t is low, then during the last iterations adding one solution will only add the one compatibility obtained from picking the most constrained constraint. In the general case, adding one solution to a CSP instance I will add in average t_I compatibilities in each constraint, with t_I being the tightness of I. Therefore, the tightness of the resulting instance will, on average, deviate from the desired tightness t by $\frac{t}{d^k}$.

3.3 Behavior of the Minimal CSP

An important question is whether our generator is refined enough to observe any interesting property held by the Minimal CSP. To this end, we used our algorithm to generate Minimal CSP instances with varying numbers of compatibilities, then solved them with two state of the art solvers. We found that we could observe the same behavior found in most other NP-hard problems, that is an easy-hard-easy pattern in the difficulty of the instances when varying the constrainedness.

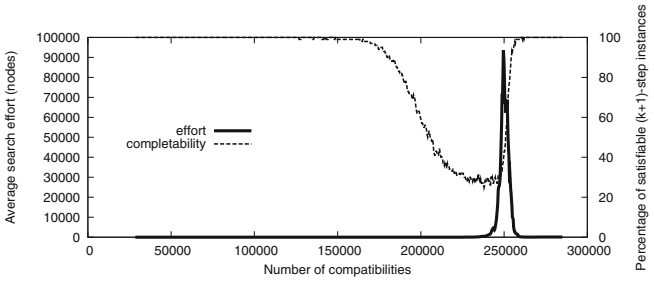
We, therefore, looked for the phase transition behind this phenomenon. The classical notion of phase transition relies on satisfiability: loosely constrained CSP instances are usually satisfiable and easy to solve, while highly constrained CSP instances are rarely satisfiable, albeit also easy to solve. The transition from the former set of instances to the latter is very sharp, and includes what appears to be the hardest to solve CSP instances.

However, because of the nature of the Minimal CSP, we cannot rely directly on satisfiability to find an adequate phase transition. Other approaches, like for example counting the number of backbone variables [1], also fail to adapt to the Minimal CSP. In order to avoid this problem, we introduced the notion of p-step instance, specifically tailored to be able to be used in the case of the Minimal CSP.

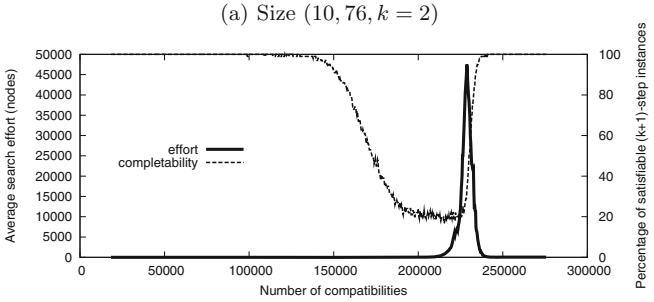
What we call a p-step instance of a minimal instance I of size (d, n, k), formally defined in Definition 7, is the sub-instance of size $(d^-, n - p, k)$ obtained from I after making p random assignments while propagating $(1, k-1)$-consistency [4] after each assignment.

Definition 7 (p-Step Instance). *Let I and I' be two CSP instances. Let I_{cons} be the instance obtained from I by propagating $(1, k-1)$-consistency [4]. For all p, we say that I' is a* p-step *instance of I if any of the following is true:*

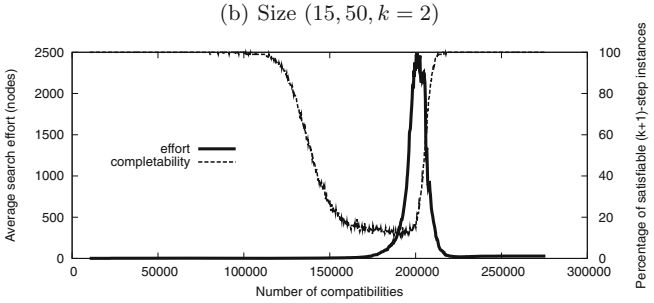
- *$p = 0$ and $I' = I_{cons}$.*
- *$p = 1$, all the domains of I_{cons} contain at most one value, and $I' = I_{cons}$.*
- *$p = 1$ and I' is the instance obtained from I_{cons} by assigning some value to some variable v in I_{cons} such that the domain of v contains at least two values, then propagating $(1, k-1)$-consistency.*
- *$p > 1$ and there is a CSP instance I'' such that I'' is a $(p-1)$-step instance of I and I' is a 1-step instance of I''.*



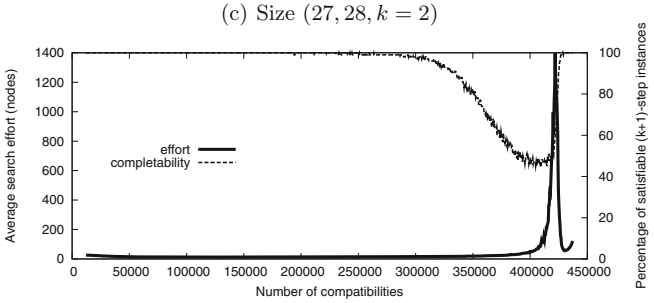
(a) Size $(10, 76, k = 2)$



(b) Size $(15, 50, k = 2)$



(c) Size $(27, 28, k = 2)$



(d) Size $(6, 24, k = 3)$

Fig. 3. Comparison between the difficulty of a given Minimal CSP instance and the percentage of its $(k + 1)$-step instances that are satisfiable, for various problem sizes.

We looked at the satisfiablity of p-step instances. From the definition of minimality, we know that if I is a Minimal CSP instance of arity k, and if I' is a k-step instance of I, then I' is satisfiable. Therefore, in order to not get trivial results, we chose $p = k + 1$ in all our experiments. We noticed what appeared to be a stong correlation between the percentage of satisfiable $(k+1)$-step instances of a given Minimal CSP instance I and the effort required to solve I.

We present the results of our experiments in Fig. 3.[2] Because of lack of space, we only show four instance sizes: (10,76,2), (15,50,2), (27,28,2) and (6,24,3). We used two state of the art solvers: Mistral and MiniSat. We found that when solving binary instances Mistral was faster, while on the other hand MiniSat was more efficient for instances of arity greater or equal than 3. Also because of lack of space, we only show for each size the results from the faster solver, that is Mistral for the binary cases and MiniSat for the last, ternary, case.

In all four figures, for both continuous and dashed subplots, the horizontal axis represents the number of compatibilities of the instance. In the case of the continuous subplot (effort), the vertical axis represents in all four figures the number of nodes the solver requires to find a solution, taken across 100 instances by point. In the case of the dashed subplot (completability), the vertical axis represents in all four figures the percentage of p-step instances that have a solution, with $p = 3$ for all binary instances and $p = 4$ for ternary instances. Therefore, if there is a point at the position (x, y) of the dashed subplot in the figure representing our experiments for size (d, n, k), it means that $y\%$ of the $k+1$-step instances of Minimal CSP instances of size (d, n, k) with x compatibilities are satisfiable.

All our experiments run over the full range of possible values of compatibilities for Minimal CSP instances, i.e. for a given size (d, n, k) from bare instances with $d \times \frac{n!}{(n-k)!k!}$ compatibilities to complete instances with $d^k \times \frac{n!}{(n-k)!k!}$ compatibilities. Since the number of compatibilities in a given CSP instance is directly related to the average tightness of its constraints, plotting against the number of compatibilities, from low to high, is equivalent to plotting against the constrainedness, from high to low.

4 Conclusion

While the Minimal Constraint Satisfaction Problem has been tackled before, we presented in this paper new results and tools which will be extremely useful in any future work on the Minimal CSP. We have shown that even when bounding the size d of the domains and the arity k of the constraints, the Minimal CSP is NP-hard exactly when the general CSP is, that is for $d \geq 3$ or ($d = 2$ and $k \geq 3$). We also have provided a generation algorithm which can be used to create and parameterize Minimal CSP instances of any size. We ran our generator in order to verify that it is refined enough to be able to expose any interesting property

[2] Our experiments are run under CentOS 6.6, on two Intel processors (1.33 Ghz each), and with 12 GB of DDR2 FB-DIMM RAM.

held by the Minimal CSP. This led us to the empirical discovery of an apparent correlation between the effort required to solve a given Minimal CSP instance and the percentage of its $(k+1)$-step instances that are satisfiable, with p-step instances being a new notion that we introduced.

The work we have done in this paper opens up a number of future avenues of research. In particular, one could look at the relation between the completability of $(k+1)$-step instances and the effort required to find a solution to Minimal CSP instances, and find out how the former influences the latter. Alternatively, one could further study this new notion of p-step instance, by using for example higher values for p or by applying the concept to other kinds of CSP instances. Indeed, one major asset of p-step instances is that their definition is not restricted to the Minimal CSP, nor any specific subclass of the CSP.

Acknowledgments. This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

References

1. Achlioptas, D., Gomes, C., Kautz, H., Selman, B.: Generating satisfiable problem instances. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of AAAI, pp. 256–261. AAAI Press/The MIT Press (2000)
2. Amilhastre, J., Fargier, H., Marquis, P.: Consistency restoration and explanations in dynamic csps application to configuration. Artif. Intell. **135**(1–2), 199–234 (2002)
3. Clark, D.A., Frank, J., Gent, I.P., MacIntyre, E., Tomov, N., Walsh, T.: Local search and the number of solutions. In: Freuder, Eugene C. (ed.) CP 1996. LNCS, vol. 1118. Springer, Heidelberg (1996)
4. Freuder, E.C.: A sufficient condition for backtrack-bounded search. J. ACM **32**(4), 755–761 (1985)
5. Gent, I.P., MacIntyre, E., Prosser, P., Walsh, T.: The constrainedness of search. In: Clancey, W.J., Weld, D.S. (eds.) Proceedings of AAAI, pp. 246–252. AAAI Press/The MIT Press (1996)
6. Gottlob, G.: On minimal constraint networks. Artif. Intell. **191–192**, 42–60 (2012)
7. Hogg, T., Huberman, B.A., Williams, C.P.: Phase transitions and the search problem. Artif. Intell. **81**(1–2), 1–15 (1996)
8. Junker, U.: Configuration. In: Handbook of Constraint Programming. Foundations of Artificial Intelligence, pp. 837–873. Elsevier (2006)
9. Schaefer, T.J.: The complexity of satisfiability problems. In: Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V. (eds.) Proceedings of the 10th Annual ACM Symposium on Theory of Computing, 1–3 May 1978, San Diego, pp. 216–226. ACM (1978)
10. Xu, K., Li, W.: Exact phase transitions in random constraint satisfaction problems. J. Artif. Intell. Res. (JAIR) **12**, 93–103 (2000)
11. Xu, K., Li, W.: Many hard examples in exact phase transitions. Theor. Comput. Sci. **355**(3), 291–302 (2006)