Random Constraint Satisfaction: Easy Generation of Hard (Satisfiable) Instances

Ke Xu [1,*]

National Lab of Software Development Environment,
School of Computers,
Beihang University,
Beijing 100083, China

Frédéric Boussemart [2] Fred Hemery [2] Christophe Lecoutre [2,*]

CRIL (Centre de Recherche en Informatique de Lens)
CNRS FRE 2499
rue de l'université, SP 16
62307 Lens cedex, France

Abstract

In this paper, we try to further demonstrate that the models of random CSP instances proposed by Xu and Li are of theoretical and practical interest. Indeed, these models, called RB and RD, present several nice features. First, it is quite easy to generate random instances of any arity since no particular structure has to be integrated, or property enforced, in such instances. Then, the existence of an asymptotic phase transition can be guaranteed while applying a limited restriction on domain size and on constraint tightness. In that case, a threshold point can be precisely located and all instances have the guarantee to be hard at the threshold, i.e., to have an exponential tree-resolution complexity. Next, a formal analysis shows that it is possible to generate forced satisfiable instances whose hardness is similar to unforced satisfiable ones. This analysis is supported by some representative results taken from an intensive experimentation that we have carried out, using complete and incomplete search methods.

Key words: Phase Transition, Constraint Network, Hard Random Instances

Preprint submitted to Elsevier *31 January 2007*

1 Introduction

Over the past ten years, the study of phase transition phenomena has been one of the most exciting areas in Computer Science and Artificial Intelligence. Numerous studies have established that for many NP-complete problems (e.g., SAT and CSP), the hardest random instances occur, while a control parameter is varied accordingly, between an under-constrained region where all instances are almost surely satisfiable and an over-constrained region where all instances are almost surely unsatisfiable. In the transition region, there is a threshold where half the instances are satisfiable (and half the instances are unsatisfiable). Generating hard instances is important both for understanding the complexity of the problems and for providing challenging benchmarks [11].

Another remarkable progress in Artificial Intelligence has been the development of incomplete algorithms for various kinds of problems. And, since this progress, one important issue has been to produce hard satisfiable instances in order to evaluate the efficiency of such algorithms, as the approach that involves exploiting a complete algorithm in order to keep random satisfiable instances generated at the threshold can only be used for instances of limited size. Also, it has been shown that generating hard (forced) satisfiable instances is related to some open problems in cryptography such as computing a one-way function [20,11].

In this paper, we mainly focus on random CSP (Constraint Satisfaction Problem) instances. Initially, four "standard" models, denoted A, B, C and D [39,16], have been introduced to generate random binary CSP instances. However, Achlioptas et al. [3] have identified a shortcoming of all these models. Indeed, they prove that random instances generated using these models suffer from (trivial) unsatisfiability as the number of variables increases. To overcome the deficiency of these standard models, several alternatives have been proposed.

On the one hand, a model E has been proposed in [3] and a generalized model in [31]. However, the model E does not permit to tune the density of the instances and the generalized model requires an awkward exploitation of probability distributions. Also, other alternatives correspond to incorporating some "structure" in the

[*] Please correspond with the first author for the theory of this paper and the last author for the experiment.

Email addresses: kexu@nldse.buaa.edu.cn (Ke Xu), boussemart@cril.univ-artois.fr (Frédéric Boussemart), hemery@cril.univ-artois.fr (Fred Hemery), lecoutre@cril.univ-artois.fr (Christophe Lecoutre).

[1] Author partially supported by the National 973 Program of China (Grant No. 2005CB321901), NSFC Grant 60403003 and FANEDD Grant 200241.

[2] Authors supported by the CNRS, the "programme COCOA de la Région Nord/Pas-de-Calais" and by the "IUT de Lens'.

generated random instances. Roughly speaking, it involves ensuring that the generated instances be arc consistent [16] or path consistent [15]. The main drawback of all these approaches is that generating random instances is no more quite a natural and easy task.

On the other hand, standard models have been revised [44,45,14,38] by controlling the way parameters change as the problem size increases. The alternative model D scheme [38], where both the domain size and the average degree of the constraint graph increase with the number of variables, guarantees the occurrence of an asymptotic phase transition as the constraint tightness is varied. The two revised models, called RB and RD [44,45] provide the same guarantee by varying one of two control parameters around a critical value that, in addition, can be computed. Also, in [14], a range of suitable parameter settings is identified which allows to exhibit a non-trivial threshold of satisfiability. Their theoretical results apply to binary instances taken from model A and to "symmetric" binary instances from a so-called model B which, not corresponding to the standard one, associates the same relation with every constraint.

The models RB and RD present several nice features:

- it is quite easy to generate random instances of any arity as no particular structure has to be integrated, or property enforced, in such instances.
- the existence of an asymptotic phase transition can be guaranteed while applying a limited restriction on domain size and on constraint tightness. For instances involving constraints of arity k, the domain size is required to be greater than the kth root of the number of variables and the (threshold value of the) constraint tightness is required to be at most $\frac{k-1}{k}$.
- when the asymptotic phase transition exists, a threshold point can be precisely located, and all instances generated following models RB and RD have the guarantee to be hard at the threshold, i.e., to have an exponential tree-resolution complexity.
- it is possible to generate forced satisfiable instances whose hardness is similar to unforced satisfiable ones.

Concerning the last item, note that instances forced to be satisfiable are simply built by randomly generating first a solution, and then a set of constraints guaranteed to support this solution. We show that under the same conditions, those used to establish the existence of an asymptotic phase transition in model RB, there is an asymptotic similarity between forced and unforced satisfiable instances of model RB with respect to the number and the distribution of solutions. We believe that this is one important aspect of our contribution.

Also, remember that CSP is a generalization of the SAT problem which is the first proven NP-complete problem and plays a central role in computational complexity. In the study of phase transitions in NP-complete problems, random 3-SAT has received most attention, both theoretically and experimentally, in the past decade or so. However, until now, the existence of the threshold phenomenon in random 3-SAT has not been established, not even the exact value of the threshold point. As such, much of the work on random 3-SAT has been focused on proving lower bounds and upper bounds for the threshold point. Through a series of hard work, the current best lower bound and upper bound for random 3-SAT are 3.42 [22] and 4.506 [13], respectively. In contrast, the existence of phase transitions in model RB has been established and the threshold points are also known exactly. In fact, these results about model RB are mainly obtained by direct application of the second moment method which, unfortunately, fails on random 3-SAT. The reason for this is that the distribution of the number of solutions for model RB is very uniform (i.e. almost all instances have the same number of solutions) while for random 3-SAT, this distribution is highly skewed (i.e. a few instances have far more solutions than most instances). From the viewpoint of phase transitions in NP-complete problems, it is therefore interesting to further investigate in which aspects random 3-SAT and model RB behave differently, and if such comparisons can shed any light on the fundamental properties of NP-complete problems. In this paper, we will see an example of such a comparison: when random 3-SAT and model RB are used to generate satisfiable instances, the same strategy can produce very different results.

Finally, note that in experimental CSP studies, random instances of model B with varying tightness and density are often used as benchmarks for algorithms. However, such instances can not be used to evaluate the asymptotic performance of algorithms. To fill this need, a good way is to generate random instances of increasing size (i.e. number of variables) which are guaranteed to be hard. An advantage of model RB over model B is that it provides a simple method to generate such asymptotically hard instances (you can download such a generator at `http://www.cril.univ-artois.fr/~lecoutre`) which are also useful for understanding what makes a problem really hard to solve.

This paper (an extended version of [43]) is organized as follows. We first introduce constraint networks and give an overview of the different models that have been proposed to generate random networks (Section 2). Then, we introduce Models RB and RD (Section 3) and present a variant of model RD which can be used to generate random networks involving constraints of any arity (Section 4). Next, we provide a formal analysis about generating both forced and unforced hard satisfiable instances (Section 5). Finally, we present the algorithms (Section 6) that we have used for our experimentation (Section 7), and, before concluding, we discuss some related work.

2 Models of Random Constraint Networks

We first introduce constraint networks and the Constraint Satisfaction Problem.

Definition 1 *A Constraint Network is a pair* $(\mathscr{X}, \mathscr{C})$ *where:*

- $\mathscr{X} = \{X_1, \ldots, X_n\}$ *is a finite set of* n *variables such that each variable* X_i *has an associated domain, denoted dom*(X_i)*, which contains the set of values allowed for* X_i*,*
- $\mathscr{C} = \{C_1, \ldots, C_m\}$ *is a finite set of* m *constraints such that each constraint* C_j *involves a subset of variables of* $\mathscr{X}$*, called scope and denoted vars*(C_j)*, and has an associated relation, denoted rel*(C_j)*, which contains the set of tuples allowed for the variables in vars*(C_j)*.*

The arity of a constraint C is the number of variables involved in C, i.e., the number of variables of its scope. The tightness of a constraint corresponds to the proportion of disallowed tuples in the associated relation. However, when one generates random constraints, it is possible to define the tightness as the probability that a tuple be disallowed.

A solution to a constraint network is an assignment of values to all the variables such that all the constraints are satisfied. The set of solutions of P is denoted sol(P). A constraint network is said to be satisfiable iff it admits at least one solution. The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether a given constraint network is satisfiable. A CSP instance is then defined by a constraint network, and solving it involves either finding one (or more) solution or determining its unsatisfiability.

Now, we can give a quick overview of the different models that have been proposed to generate random constraint networks. Four conventional models denoted A, B, C and D [39,16] to generate random binary constraint networks have been introduced. Whatever model is chosen, two steps are necessary to generate such a network. In the first step, we have to build its macro-structure called constraint hyper-graph, whereas in the second step, we have to build its micro-structure called consistency hyper-graph. Actually, the two steps can be interleaved. The constraint hyper-graph is composed of n vertices corresponding to variables and m hyper-edges corresponding to constraints ; each hyper-edge links the variables occurring in the scope of the constraint it represents. The consistency hyper-graph is composed of $\sum_{i=1}^{n} \mid dom(X_i) \mid$ vertices corresponding to all values of all domains and of $\sum_{j=1}^{m} \mid rel(C_j) \mid$ hyper-edges corresponding to all supports (allowed tuples) of all relations. Note that the density of a constraint network (or hyper-graph), assuming that it only involves constraints of arity k (and distinct scopes), is equal to $m/\binom{n}{k}$.

Each random CSP instance is characterized by a tuple (k, n, d, p_1, p_2) where k denotes the arity of the constraints, n the number of variables, d the uniform domain size, p_1 a measure of the density of the constraint graph and p_2 a measure of the tightness of the constraints. There are four models since p_1 and p_2 can represent either a probability or a proportion. p_1 is considered as a probability in models A,C and as a proportion in models B,D while p_2 is considered as a probability in models A,D and as a proportion in models B,C. Note that, for binary instances, k is usually omitted.

Model B is defined as follows:

Definition 2 (Model B) *A class of random CSP instances of model B will be denoted B(k, n, d, p_1, p_2) where, for each instance:*

- $k \geq 2$ *denotes the arity of each constraint,*
- $n \geq 2$ *denotes the number of variables,*
- $d \geq 2$ *denotes the size of each domain,*
- $1 \geq p_1 > 0$ *determines the number* $m = p_1\binom{n}{k}$ *of constraints,*
- $1 > p_2 > 0$ *determines the number* $t = p_2 d^k$ *of disallowed tuples of each relation.*

To generate one instance $P \in$ *B*(k, n, d, p_1, p_2)*, we have to build* m *constraints, each one formed by randomly selecting (without repetition) a scope of* k *(distinct) variables and randomly selecting (with repetition) a relation of* t *distinct disallowed tuples.*

Note that $p_1\binom{n}{2}$ and $p_2 d^k$ may have to be rounded to the nearest integer. Then, one can prefer, instead of proportions, directly using the number m of constraints and the number t of disallowed tuples.

Model D is defined as follows:

Definition 3 (Model D) *A class of random CSP instances of model D will be denoted D(k, n, d, p_1, p_2) where, for each instance:*

- $k \geq 2$ *denotes the arity of each constraint,*
- $n \geq 2$ *denotes the number of variables,*
- $d \geq 2$ *denotes the size of each domain,*
- $1 \geq p_1 > 0$ *determines the number* $m = p_1\binom{n}{k}$ *of constraints,*
- $1 > p_2 > 0$ *denotes the constraint tightness in terms of probability.*

To generate one instance $P \in$ *D*(k, n, d, p_1, p_2)*, we have to build* m *constraints, each one formed by randomly selecting (without repetition) a scope of* k *(distinct variables) and randomly selecting (with repetition) a relation such that each one of the* d^k *tuples is not allowed with probability* p_2*.*

It has been shown [10] that the hardest random instances occur at a so-called phase transition between an under-constrained region where all instances are almost surely satisfiable and an over-constrained region where all problems are almost surely unsatisfiable. This phase transition occurs when a control parameter is varied accordingly, usually p_2 or κ which measures the constrainedness of an instance [17]. Furthermore, the peak of difficulty corresponds approximately to the value of the control parameter where 50% of instances are satisfiable and is referred to as the threshold or crossover point. The mushy region [39,34] denotes the range of values of the control parameter over which the phase transition takes place. Locating the phase transition has been addressed in [41,39].

However, Achlioptas et al. [3] have identified a shortcoming of all four standard models. Indeed, they prove that random problem instances generated using these models suffer from (trivial) insolubility as problem size increases. More precisely, they show that, asymptotically, if $p_2 \geq 1/d$, such instances almost surely contain a flawed variable when the number of variables increases while other parameters are kept constant. A flawed variable is a variable such that each value from its associated domain is flawed, i.e., is not consistent with respect to a constraint of the instance. Therefore, an instance involving a flawed variable is clearly unsatisfiable and this can be shown in polynomial time.

To overcome the deficiency of standard models, several alternatives have been proposed. In [3], a model E is introduced by selecting, with probability p, each one of the $d^2\binom{n}{2}$ tuples (X, Y, a, b) where X and Y are two distinct variables, $a \in dom(X)$ and $b \in dom(Y)$. This new model is proved to be asymptotically interesting. However, it provides less flexibility in the construction of the network and involves quickly generating a complete constraint graph even if p is small. A generalized model has been proposed by Molloy [31] with the introduction of a probability distribution in order to directly select constraints (instead of selecting allowed tuples, one by one). The author proves that *very well behaved* sets of constraints obtained with a probability distribution allow exhibiting a phase transition. As using such distributions can be awkward in practice, the issue of generating difficult instances for some distributions is also addressed in [31].

On the other hand, some works have focused on incorporating some “structure” in the generated random instances. Roughly speaking, the principle is to ensure that the generated instances be arc consistent [16] or (strongly) path consistent [15]. More precisely, Gent et al. [16] propose variants, called flawless models, of standard models which prevent the presence of flawed values. They consider that “each value must be supported by at least one unique value, i.e., one value which is not also required to support another value” and achieve it by fixing in each binary constraint, a set S of d allowed pairs such that any pair $t1, t2$ of elements of S is such that $t1[1] \neq t2[1] \wedge t1[2] \neq t2[2]$. Then, the instances generated according to flawless models are guaranteed to be arc-consistent and at any value of $p_2 < 1/2$ do not suffer asymptotically from trivial insolubility”. Unluckily, they can be solved in polynomial time as they embed easy sub-problems [15]. A generalization of this approach has then been proposed by Gao and Culberson [15]. The authors show that if each relation is chosen in such a way that the generated instances are strongly path consistent, then such instances admit an exponential resolution complexity no matter how large the constraint tightness is. By ensuring the presence of a l-regular bipartite graph in each generated relation (with a sufficiently large l), the instances are guaranteed to be strongly path consistent. The main drawback of these approaches is that generating random instances is no more quite a natural and easy task.

Finally, in [44,38,45,14] standard models have been revised by controlling the way parameters change as the problem size increases. The alternative model D scheme proposed by Smith in [38] guarantees the occurrence of a phase transition when some parameters are controlled and when the constraint tightness is within a certain range. The two revised models RB and RD introduced by Xu and Li in [44,45] provide the same guarantee by varying one of two control parameters around a critical value that can be computed. In the next section, we will see that almost all instances of models RB and RD are hard, i.e., do not present resolution proofs of size less than exponential size. Also, Frieze and Molloy [14] identify a range of suitable parameter settings in order to exhibit a non-trivial threshold of satisfiability. Their theoretical results apply to binary instances taken from model A and to "symmetric" binary instances from a so-called model B which, not corresponding to the standard one, associates the same relation with every constraint.

3 Models RB and RD

In this section, we introduce some theoretical results about two models defined in [44,45]. First, we present the model RB that represents an alternative to model B. Note that in order to simplify the theoretical analysis, unlike model B, model RB allows selecting constraints with identical scopes. However, it may be that the results introduced below also hold for Model B (i.e. when selection of scopes is performed without repetition). The reason for this is that the number of repeated constraints is asymptotically much smaller than the total number of constraints and thus can be neglected in the analysis. But the main difference of model RB with respect to model B is that the domain size of each variable grows polynomially with the number of variables.

Definition 4 (Model RB) *A class of random CSP instances of model RB is denoted RB(k,n,α,r,p) where, for each instance:*

- $k \geq 2$ *denotes the arity of each constraint,*
- $n \geq 2$ *denotes the number of variables,*
- $\alpha > 0$ *determines the domain size* $d = n^\alpha$ *of each variable,*
- $r > 0$ *determines the number* $m = r.n. \ln n$ *of constraints,*
- $1 > p > 0$ *determines the number* t = pd^k *of disallowed tuples of each relation.*

To generate one instance $P \in$ *RB(k,n,α,r,p), we have to build* m *constraints, each one formed by randomly selecting (with repetition) a scope of* k *(distinct) variables and randomly selecting (with repetition) a relation of* t *distinct disallowed tuples.*

When fixed, α and r give an indication about the growth of the domain sizes and of the number of constraints as n increases since $d = n^\alpha$ and $m = rn \ln n$, respectively. It is then possible, for example, to determine the critical value p_{cr} of p where the hardest instances must occur. Indeed, we have $p_{cr} = 1 - e^{-\alpha/r}$ which is equivalent to the expression of p_{cr} given in [39]. Note that n^α, $r.n. \ln n$ and pd^k may have to be rounded to the nearest integer.

Another model, denoted model RD, is similar to model RB except that p denotes a probability instead of a proportion.

Definition 5 (Model RD) *A class of random CSP instances of model RD is denoted RD(k,n,α,r,p) where, for each instance:*

- $k \geq 2$ *denotes the arity of each constraint,*
- $n \geq 2$ *denotes the number of variables,*
- $\alpha > 0$ *determines the domain size* $d = n^\alpha$ *of each variable,*
- $r > 0$ *determines the number* $m = r.n. \ln n$ *of constraints,*
- $1 > p > 0$ *denotes the constraint tightness in terms of probability.*

To generate one instance $P \in$ RD(k,n,α,r,p), we have to build m constraints, each one formed by randomly selecting (with repetition) a scope of k (distinct) variables and randomly selecting (with repetition) a relation such that each one of the d^k tuples is not allowed with probability p.

For convenience, in this paper, although all given results hold for models RB and RD, we will exclusively refer to model RB. In [44], it is proved that model RB, under certain conditions, not only avoids trivial asymptotic behaviors but also guarantees exact phase transitions. More precisely, with Pr denoting a probability distribution, the following theorems hold.

Theorem 1 *If k, $\alpha > \frac{1}{k}$ and $p \leq \frac{k-1}{k}$ are constants then*

$$\lim_{n \to \infty} Pr[P \in RB(k, n, \alpha, r, p) \ is \ sat\,] = \begin{cases} 1 \ if \ r < r_{cr} \\ 0 \ if \ r > r_{cr} \end{cases}$$

where $r_{cr} = -\frac{\alpha}{\ln(1-p)}$.

Theorem 2 *If k, $\alpha > \frac{1}{k}$ and $p_{cr} \leq \frac{k-1}{k}$ are constants then*

$$\lim_{n \to \infty} Pr[P \in RB(k, n, \alpha, r, p) \ is \ sat\,] = \begin{cases} 1 \ if \ p < p_{cr} \\ 0 \ if \ p > p_{cr} \end{cases}$$

where $p_{cr} = 1 - e^{-\frac{\alpha}{r}}$.

Remark that the condition $p_{cr} \leq \frac{k-1}{k}$ is equivalent to $ke^{-\frac{\alpha}{r}} \geq 1$ given in [44]. Theorems 1 and 2 indicate that a phase transition is guaranteed provided that the domain size is not too small and the constraint tightness or the threshold value of the constraint tightness not too large. As an illustration, for instances involving binary constraints, the domain size is required to be greater than the square root of the number of variables (as $\alpha > 1/2$ and $d = n^\alpha$) and the constraint tightness or threshold value of the tightness is required to be at most 50%. The following table gives the limits of Theorems 1 and 2 for different arities.

k	α	p or p_{cr}
2	$> 1/2$	$\leq 1/2$
3	$> 1/3$	$\leq 2/3$
4	$> 1/4$	$\leq 3/4$
5	$> 1/5$	$\leq 4/5$

The next theorem establishes that unsatisfiable instances of model RB almost surely have the guarantee to be hard. A similar result for model A has been obtained by [14] with respect to binary instances (i.e. $k = 2$) and Mitchell [29] has described such kind of behaviour for instances where constraints have fewer than $d/2$ disallowed tuples.

Theorem 3 *If $P \in$ RB(k,n,α,r,p) and k, α, r and p are constants, then, almost surely*[3]*, P has no tree-like resolution of length less than $2^{\Omega(n)}$.*

The proof, which is based on a strategy following some results of [6,30], is omitted but can be found in [45].

To summarize, model RB guarantees exact phase transitions and hard instances at the threshold. It then contradicts the statement of [15] about the requirement of an extremely low tightness for all existing random models in order to have non-trivial threshold behaviors and guaranteed hard instances at the threshold.

4 Random Constraint Networks in Intention

In Constraint Programming, most of the experiments performed so far about random constraint networks involve binary instances. The main reason of this limitation is due to space complexity considerations. Indeed, the space required to store the tuples allowed (or disallowed) by (the relation associated with) a constraint exponentially grows with its arity. For instance, at least 10^9 memory units are needed

[3] We say that a property holds almost surely when this property holds with probability tending to 1 as the number of variables tends to infinity.

to store 1% of the tuples of a 10-ary constraint with 10 values per domain since 10^8 tuples, each one composed of 10 values, must be stored. Hence, in practice, it is not possible to represent large non binary random networks when constraints are given in extension.

In this section, we propose a variant (initially introduced in [24]) of model RD such that any network involves constraints defined in intention, i.e. by a predicate. As an immediate consequence, there is no more obstacle to generate problem instances with constraints of any arity, and then to perform experiments on such instances.

First, we show how it is possible to define a set $\mathscr{C}$ of random constraints defined in intention. Note that we consider, without loss of generality, that each constraint C in $\mathscr{C}$ has a unique identification number $id(C)$.

The function $isConsistent$ (see Algorithm 1) can be viewed as the predicate defining all random constraints defined in intention of a problem instance. Indeed, for any k-ary constraint C and any k-tuple t, $isConsistent(C, t, p)$ determines if the tuple t is allowed by C while considering tightness p. To achieve this, a unique real random value between 0 and 1 (exclusive) is computed by a function f whose parameters are the identification number $id(C)$ of C and the given tuple t. Then, we have simply to compare the computed random value with the constraint tightness p seen as an acceptance boundary.

Algorithm 1 Function $isConsistent$(C : Constraint, t : tuple, p : 0..1) : Boolean

```
variable realRandomValue : real

realRandomValue $\leftarrow$ f(id(C),t)
return realRandomValue $\geq$ p
```

The function f can be implemented in different ways. For instance, we can compute a seed from the parameters of f, and then use this seed to get a real value using a pseudo-random number generator. Also, we can compute a hash value using a message digest algorithm like MD5 [35] or SHA [33]. These algorithms correspond to secure one-way hash functions that take arbitrary-sized data, called documents, and output fixed-length hash values called fingerprints or message digests. For any bit changed in a document, the best hash functions involves modifying at least 50% of the bits occurring in the computed fingerprint. In our context, the document corresponds to the parameters of f, namely, the identification number of the constraint and the given tuple.

When using hash functions, one can obtain a real value by applying some simple operations as shown by Algorithm 2. After creating a document by simply concatenating the binary description of all parameters of f, a hash function (MD5, for instance) is called in order to generate a fingerprint. Then, the exclusive or operation ($\oplus$) is iteratively applied on each byte of the fingerprint. Finally, a real value between 0 and 1 is calculated by dividing the value of the resulting byte by 256.

Algorithm 2 Function f(id : integer, t : tuple) : real

```
variable b : byte
variable document : array of bytes
variable fingerprint : array of bytes

document ← createDocumentWith(id,t)
fingerprint ← MD5(document)
b ← fingerprint[1]
for i ranging from 2 to length(fingerprint) do
   b ← b ⊕ fingerprint[$i$]
end for
return $b/256$
```

It is important to note the correctness of this approach. First, $isConsistent$ always produces the same result for a given set of parameters since the same real value is always computed by f. Second, each tuple is considered, independently with probability p, as being not allowed by C. Remark that we need to introduce the constraint identification number as a parameter of f in order to get distinct constraints of same arity.

We are now in a position to define a variant, denoted RD^{int}, of model RD that can be exploited in practice for any arity.

Definition 6 (Model RD^{int}) *A class of random CSP instances of model RD^{int} is denoted RD^{int}(k,n,α,r,p) where, for each instance:*

- $k \geq 2$ *denotes the arity of each constraint,*
- $n \geq 2$ *denotes the number of variables,*
- $\alpha > 0$ *determines the domain size* $d = n^{\alpha}$ *of each variable,*
- $r > 0$ *determines the number* $m = r.n. \ln n$ *of constraints,*
- $1 > p > 0$ *denotes the constraint tightness in terms of probability.*

To generate one instance $P \in$ *RD^{int}(k,n,α,r,p), we have to build* m *constraints in intention, each one formed by randomly selecting (with repetition) a scope of* k *(distinct) variables and (implicitly) defined by the function* $isConsistent$ *described above.*

This new model involves an easy implementation, no space requirement and the possibility to perform experiments with large arity constraints.

5 Generating Hard Satisfiable Instances

For CSP and SAT (whose task is to determine if a Boolean formula, also called SAT instance, is satisfiable), there is a natural strategy to generate *forced* satisfiable instances, i.e., instances on which a solution is imposed. It suffices to generate first a random (total) assignment t and then a random instance with n variables and m constraints (clauses for SAT) such that any constraint violating t is rejected. t is then a *forced* solution.

More precisely, the method that we have considered to generate forced satisfiable instances of Model RB is as follows:

- generate a random solution t
- generate m scopes (of constraints) of size k (with repetition)
- for each constraint (of arity k), generate nb allowed tuples with nb being the nearest integer to $(1-p)d^k$
  - · select as first allowed tuple the one that satisfies the solution t
  - · select (without repetition) $nb-1$ other tuples [4] .

This strategy, quite simple and easy to implement, allows generating hard forced satisfiable instances of model RB provided that Theorem 1 or 2 holds. Nevertheless, this statement deserves a theoretical analysis.

Assuming that d denotes the domain size (d = 2 for SAT), we have exactly d^n possible (total) assignments, denoted by $t_1, t_2, \cdots, t_{d^n}$, and d^{2n} possible assignment pairs where an *assignment pair* $< t_i, t_j >$ is an ordered pair of two assignments t_i and t_j. We say that $< t_i, t_j >$ satisfies an instance if and only if both t_i and t_j satisfy the instance. Then, the expected (mean) number of solutions $E_f[N]$ for instances that are forced to satisfy an assignment t_i (which is then a forced solution) is:

$$E_f[N] = \sum_{j=1}^{d^n} \frac{\Pr[< t_i, t_j >]}{\Pr[< t_i, t_i >]}$$

where $\Pr[< t_i, t_j >]$ denotes the probability that $< t_i, t_j >$ satisfies a random instance. Note that $E_f[N]$ should be independent of the choice of the forced solution t_i. So we have:

$$E_f[N] = \frac{\sum\limits_{1 \le i,j \le d^n} \Pr[< t_i, t_j >]}{d^n \Pr[< t_i, t_i >]} = \frac{E[N^2]}{E[N]}.$$

where $E[N^2]$ and $E[N]$ are, respectively, the second moment and the first moment of the number of solutions for random unforced instances.

For random 3-SAT, it is known that the strategy mentioned above is unsuitable as it produces a biased sampling of instances with many solutions clustered around t [1]. Experiments show that forced satisfiable instances are much easier to solve

[4] For example, we can associate an integer with each tuple, put all such integers (except the one that satisfies the solution) into a set and randomly picking (and removing) $nb-1$ integers from this set.

than unforced satisfiable instances. In fact, it is not hard to show (from the result on satisfying assignment pairs in [42] that, asymptotically, $E[N^2]$ is exponentially greater than $E^2[N]$. The same conclusion can also be found in [4]. Thus, the expected number of solutions for forced satisfiable instances is exponentially larger than the one for unforced satisfiable instances. It then gives a good theoretical explanation of why, for random 3-SAT, the strategy is highly biased toward generating instances with many solutions.

For model RB, recall that when the exact phase transitions were established [44], it was proved that $E[N^2]/E^2[N]$ is asymptotically equal to 1 below the threshold, where almost all instances are satisfiable, i.e. $E[N^2]/E^2[N] \approx 1$ for $r < r_{cr}$ or $p < p_{cr}$. Hence, the expected number of solutions for forced satisfiable instances below the threshold is asymptotically equal to the one for unforced satisfiable instances, i.e. $E_f[N] = E[N^2]/E[N] \approx E[N]$. In other words, when using model RB, the strategy has almost no effect on the number of solutions and does not lead to a biased sampling of instances with many solutions.

In addition to the analysis above, we can also study the influence of the strategy on the distribution of solutions with respect to the forced solution. Based on Hamming distance, we first define the distance $d^f(t_i, t_j)$ between two assignments t_i and t_j as the proportion of variables that have been assigned a different value in t_i and t_j. We have $0 \le d^f(t_i, t_j) \le 1$.

For forced satisfiable instances of model RB, with $E_f^\delta[N]$ denoting the expected number of solutions whose distance from the forced solution (identified as t_i, here) is equal to δ, we obtain by an analysis similar to that in [44]:

$$\begin{aligned}
E_f^\delta[N] &= \sum_{j=1}^{d^n} \frac{\Pr[< t_i, t_j >]}{\Pr[< t_i, t_i >]} \quad \text{with } d^f(t_i, t_j) = \delta \\
&= \binom{n}{n\delta} (n^\alpha\text{-}1)^{n\delta} \left[\frac{\binom{n-n\delta}{k}}{\binom{n}{k}} + (1-p)\left(1 - \frac{\binom{n-n\delta}{k}}{\binom{n}{k}}\right)\right]^{rn\ln n} \\
&= \exp\left[n \ln n \left(r \ln\left(1 - p + p(1-\delta)^k\right) + \alpha\delta\right) + O(n)\right]
\end{aligned}$$

It can be shown, from the results in [44] that $E_f^\delta[N]$, for $r < r_{cr}$ or $p < p_{cr}$, is asymptotically maximized when δ takes the largest possible value, i.e. $\delta = 1$. The intuition behind this is that the value of $E_f^\delta[N]$ is determined by two factors: the first one is the number of assignments whose distance from the forced solution is δ, and the other one is the probability of such an assignment being a solution. It is easy to see that the larger the distance, the smaller the probability, which means that an assignment more similar to the forced solution is more likely to be a solution. On the other hand, the number of assignments with a distance δ from the forced solution grows with δ. Then, when $r < r_{cr}$ or $p < p_{cr}$, this factor plays a more dominant role than the probability factor as the number n of variables approaches infinity, which leads to the result as stated. It is worth mentioning that when $r > r_{cr}$ or $p > p_{cr}$, the total number of solutions for a forced satisfiable instance might be greater than 1 and the dominant role will alternate between the above two factors, yielding the result that $E_f^\delta[N]$ is asymptotically maximized at $\delta = 0$. This implies that for a forced satisfiable instance to the right of the threshold, there may exist some solutions other than the forced one and these solutions are distributed very closely around the forced one.

For unforced satisfiable instances of model RB, with $E^\delta[N]$ denoting the expected number of solutions whose distance from t_i (not necessarily a solution) is equal to δ, we have:

$$\begin{aligned} E^\delta[N] &= \binom{n}{n\delta} (n^\alpha - 1)^{n\delta} (1-p)^{rn\ln n} \\ &= \exp\left[n \ln n \left(r \ln(1-p) + \alpha\delta\right) + O(n)\right] \end{aligned}$$

It is straightforward to see that the same pattern holds for this case, i.e. $E^\delta[N]$ is asymptotically maximized when $\delta = 1$.

Intuitively, with model RB, both unforced satisfiable instances and instances forced to satisfy an assignment t are such that most of their solutions distribute far from t. This indicates that, for model RB, the strategy has little effect on the distribution of solutions, and is not biased towards generating instances with many solutions around the forced one.

For random 3-SAT, we have:

$$\begin{aligned} E_f^\delta[N] &= \binom{n}{n\delta} \left[\frac{\binom{n-n\delta}{3}}{\binom{n}{3}} + \frac{6}{7}\left(1 - \frac{\binom{n-n\delta}{3}}{\binom{n}{3}}\right)\right]^{rn} \\ &= \text{poly}(n) \exp\left[n\left(-\delta\ln\delta - (1-\delta)\ln(1-\delta) + r\ln\frac{6+(1-\delta)^3}{7}\right)\right] \\ &= \text{poly}(n)\exp[nf_1(r,\delta)] \end{aligned}$$

and

$$\begin{aligned} E^\delta[N] &= \binom{n}{n\delta}\left(\frac{7}{8}\right)^{rn} \\ &= \text{poly}(n)\exp\left[n\left(-\delta\ln\delta - (1-\delta)\ln(1-\delta) + r\ln\frac{7}{8}\right)\right] \\ &= \text{poly}(n)\exp[nf_2(r,\delta)] \end{aligned}$$

From the above two equations, we can see that $E_f^\delta[N]$ and $E^\delta[N]$ are asymptotically determined by $f_1(r,\delta)$ and $f_2(r,\delta)$ respectively. Below are two figures (See Figures 1 and 2) for $f_1(r,\delta)$ and $f_2(r,\delta)$ as a function of δ with $r = 4.25$ (the ratio of clauses to variables). It follows from these two figures that as r approaches 4.25, $f_1(r,\delta)$

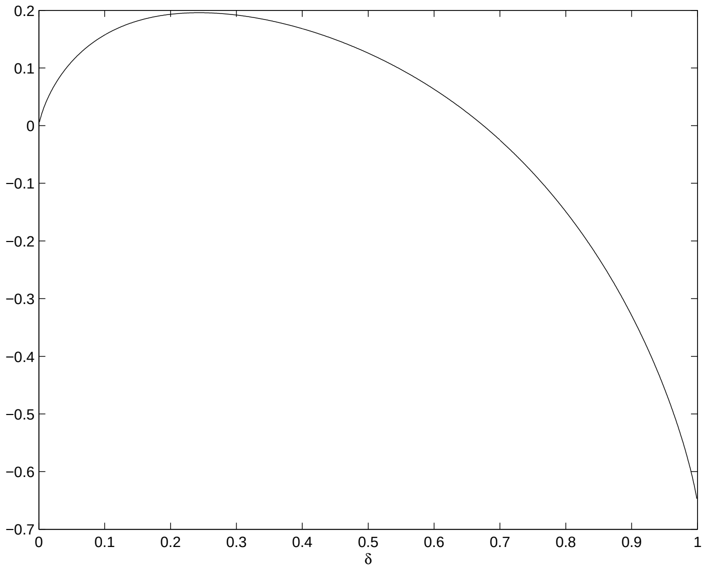

Fig. 1. $f_1(r, \delta)$ as a function of δ with $r = 4.25$

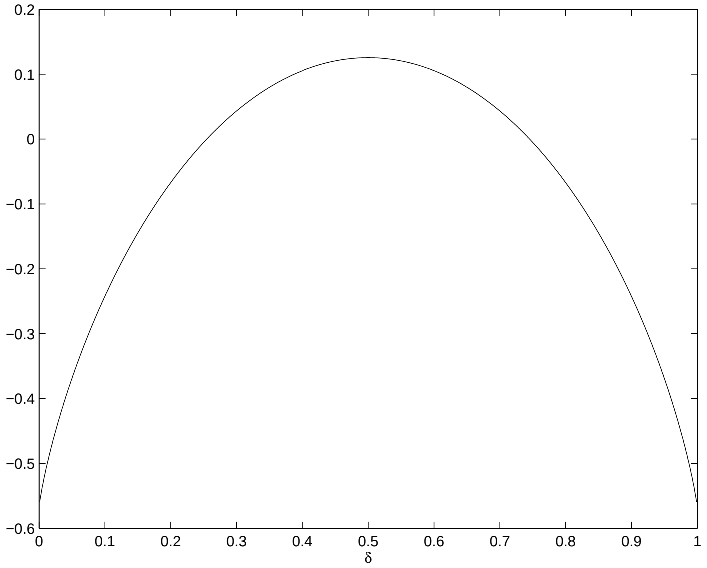

Fig. 2. $f_2(r, \delta)$ as a function of δ with $r = 4.25$

and $f_2(r, \delta)$ are maximized when $\delta \approx 0.24$ and $\delta = 0.5$, respectively. This means, in contrast to model RB, that when r is near the threshold, most solutions of forced instances distribute in a place much closer to the forced solution than solutions of unforced satisfiable instances.

6 Solving Constraint Networks

Before presenting the results of our experimentation, we need to introduce the algorithms that we have used. To solve a CSP instance, a depth-first search algorithm with backtracking can then be applied, where at each step of the search, a variable assignment is performed followed by a filtering process called constraint propagation. Usually, domains are filtered (reduced) by considering some properties of constraint networks. Such properties are called domain filtering consistencies [12] and Generalized Arc Consistency (GAC) remains the central one. For binary constraints, this property is simply called Arc Consistency (AC).

Definition 7 *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN. A pair (X, a), with $X \in \mathscr{X}$ and $a \in dom(X)$, is generalized arc consistent (GAC) iff $\forall C \in \mathscr{C} \mid X \in vars(C)$, there exists a support of (X, a) in C, i.e., a tuple[5] $t \in rel(C)$ such that $t[X] = a$ and $t[Y] \in dom(Y)$ $\forall Y \in vars(C)$. P is GAC iff $\forall X \in \mathscr{X}$, dom(X) $\neq \emptyset$ and $\forall a \in dom(X)$, (X, a) is GAC.*

Establishing Generalized Arc Consistency on a given network P involves removing all values that are not GAC. Many generic algorithms have been proposed to establish AC and GAC (e.g. see [27,8,23]). When constraints are given in extension (usually under the form of tables), it is possible to adopt different propagation schemes [7,26]. However, for instances of model RD^{int}, one necessary needs to use the so-called GAC-valid scheme [7,26].

GAC(P) will denote the constraint network obtained after enforcing GAC on a given constraint network P. If there is a variable with an empty domain in GAC(P), denoted GAC(P) = $\perp$, then P is clearly unsatisfiable.

The MGAC algorithm, i.e., the algorithm which maintains GAC during the search of a solution, is nowadays considered as the best generic algorithm to solve CSP instances (provided that the arity of the constraints be not too high). For binary instances (i.e. instances only involving binary constraints), it is called MAC [36]. Algorithms 3 and 4 correspond to a recursive version of MGAC (using 2-way branching). One can solve a CSP instance by calling the $MGAC$ function: one solution is displayed iff the instance is satisfiable. Before giving more details about the algorithm, we need to introduce the following notations. $P|_{X=a}$ denotes the constraint network obtained from P by restricting the domain of X to the singleton $\{a\}$ whereas $P|_{X \neq a}$ denotes the constraint network obtained from P by removing the value a from the domain of X. $P \backslash X$ denotes the constraint network obtained from P by removing the variable X (it means that, not only, X is removed from $\mathscr{X}$ but also that X is eliminated from any constraint involving it by projecting the associated relation over all other involved variables). The recursive algorithm is started by calling the $MGAC$ function (Algorithm 3) with a constraint network to

[5] $t[X]$ denotes the value assigned to X in t

Algorithm 3 MGAC(P = $(\mathscr{X},\mathscr{C})$: Constraint Network)

```
$P' \leftarrow GAC(P)$
if $P' \neq \bot$ then
    searchMGAC($P'$)
end if
```

Algorithm 4 searchMGAC(P = $(\mathscr{X},\mathscr{C})$: Constraint Network) : Boolean

```
 1: if $\mathscr{X} = \emptyset$ then
 2:     return true
 3: else
 4:     select a pair ($X$,$a$) such that $X \in \mathscr{X}$ and $a \in dom(X)$
 5:     $P' \leftarrow GAC(P|_{X=a})$
 6:     if $P' \neq \bot$ then
 7:         if searchMGAC($P' \backslash X$) then
 8:             write($X$,$a$) ; return true
 9:     $P' \leftarrow GAC(P|_{X \neq a})$
10:     if $P' \neq \bot$ then
11:         if searchMGAC($P'$) then
12:             return true
13:     return false
14: end if
```

be solved. If the given constraint network P can be made generalized arc consistent then the search for a solution begins (Algorithm 4). It involves selecting a pair (X,a) and trying first $X = a$ and then $X \neq a$ (if no solution has been found with $X = a$). After any consistent assignment, the assigned variable is eliminated from the network and search is continued (line 7). When the constraint network becomes empty (line 1), it means that a solution has been found and that the search can be stopped by returning true (lines 2, 8 and 12).

MGAC is a complete search algorithm. It means that one can use this algorithm to find a solution or prove that no solution exists. However, there exists another category of algorithms which are said to be incomplete. Such algorithms may find solutions but cannot prove unsatisfiability. They have been shown to be very efficient on some domains of applications. Local search algorithms belong to this category. The principle of local search is to move in the space of all possible interpretations (here, an interpretation is the assignment of a value to all variables) until a solution is found. Usually, a move is selected in the neighborhood of the current interpretation and corresponds to a maximum improvement of a cost function.

Algorithms 5 and 6 correspond to a general description of a local search algorithm called TABU. The algorithm is started by calling the $TABU$ function (Algorithm 5) with a constraint network to be solved. If the given constraint network P can be made arc consistent then the search for a solution begins (Algorithm 6). It involves performing several runs such that, initially, for each run, a tabu list is initialized and an initial random interpretation is generated (the function rand randomly selects a

Algorithm 5 TABU(P = $(\mathscr{X},\mathscr{C})$: Constraint Network)

```
$P' \leftarrow GAC(P)$
**if** $P' \neq \bot$ **then**
    searchTABU($P'$)
**end if**
```

Algorithm 6 searchTABU(P = $(\mathscr{X},\mathscr{C})$: Constraint Network)

```
**for** $i$ ranging from 1 to $maxRuns$ **do**
    tabu $\leftarrow \emptyset$
    sol $\leftarrow \{(X, rand(X)) \mid X \in \mathscr{X}\}$
    **for** $j$ ranging from 1 to $maxMoves$ **do**
        select a pair ($X$,$a$) s.t. $X \in \mathscr{X}$, $a \in dom(X)$, ($X$,$a$) $\notin sol$ and ($X$,$a$) $\notin tabu$
        replace ($X$,$b$) $\in sol$ with ($X$,$a$)
        **if** sol $\in$ sol($P$) **then**
            write(sol) ; return
        **end if**
        **if** isFull(tabu) **then**
            tabu $\leftarrow$ tabu $\setminus$ firstIn(tabu)
        **end if**
        tabu $\leftarrow$ tabu $\cup$ $\{(X, b)\}$
    **end for**
**end for**
```

value in the domain of the given variable). Then, until the current interpretation corresponds to a solution or the number of performed moves reaches the number of allowed moves for the current run, a neighbor is selected. Here, the neighborhood is composed of all interpretations which differ from the current interpretation by exactly one value. The role of the tabu list is to avoid becoming stuck in local minima by repeatedly performing the same cycle of moves. Note that the selected move (represented by a pair (X,b)) cannot correspond to a move recently performed and recorded in the tabu list. After each move, we remove the oldest element of the tabu list (the function firstIn returns this element) if the maximum capacity of the list is reached (the function isFull determines if this is the case), and we add this new move to the list.

The performance of the MGAC and TABU algorithms highly depend on the heuristics used to select the next pair (X,a) in the main loop of both algorithms. It has been shown that constraint weighting can be exploited in order to concentrate search on the hard parts of the constraint network. It does respect the fail-first principle : “To succeed, try first where you are most likely to fail” [19]. The idea is to associate a weight with any constraint C and to increment the weight of C whenever C is violated during search. As search progresses, the weight of hard constraints become more and more important and this particularly helps the heuristic to select variables appearing in the hard part of the network. More information can be found in [32,37,28,40,9,25].

7 Experimental Results

As all introduced theoretical results hold when $n \to \infty$, the practical exploitation of these results is an issue that must be addressed. In this section, we give some representative experimental results which indicate that practice meets theory even if the number n of variables is small. Note that different values of parameters α and r have been selected in order to illustrate the broad spectrum of applicability of model RB (and model RD). The platform that we have used for our experimentation (conducted on a PC Pentium IV 2.4GHz 512Mo under Linux) is called *Abscon* (see `http://www.cril.univ-artois.fr/~lecoutre`).

Theorems 1 and 2 guarantee asymptotically the presence of a phase transition and the exact localization of the threshold. First, it is valuable to know in practice, to what extent, Theorems 1 and 2 give precise thresholds according to different values of α, r and n. The experiments that we have run wrt Theorem 2, as depicted in Figure 3, suggest that all other parameters being fixed, the greater the value of α, r or n is, the more precise Theorem 2 is. More precisely, in Figure 3, the difference between the threshold theoretically located and the threshold experimentally determined is plotted against $\alpha \in [0.2, 1]$ ($d \in [2..20]$), against $r \in [0.8, 2.5]$ ($m \in [50..150]$) and against $n \in [8..100]$. Note that the vertical scale refers to the difference in constraint tightness and that the horizontal scale is normalized (value 0 respectively corresponds to $n = 8$, $\alpha = 0.2$ and $r = 0.8$, etc.).

To solve the random instances generated by model RB, we have used the MGAC

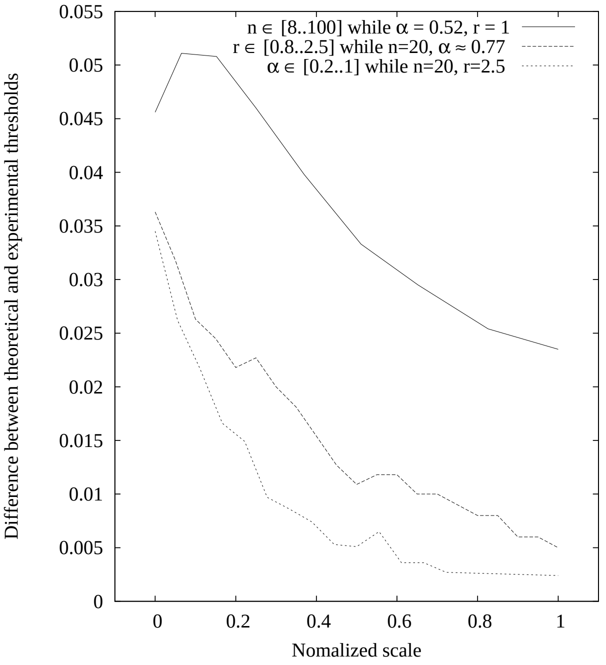

Fig. 3. Difference between theoretical and experimental thresholds against α, r and n

and TABU algorithms described in previous section. For MGAC, we have used $dom/wdeg$ [9,25] as variable ordering heuristic, $lexico$ as value ordering heuristic and GAC3.2 [23] as embedded arc consistency algorithm. For TABU, we have used constraint weighting [32,37,40] to perform local moves, set the size of the tabu list to 25 and the maximum number of moves of any run to $150,000$. Also, every three runs, constraint weights are reset to 0.

We have studied the difficulty of solving with MAC the binary instances of model RB generated around the theoretical threshold $p_{cr} \approx 0.23$ given by Theorem 2 for $k = 2$, $\alpha = 0.8$, $r = 3$ and $n \in \{20, 30, 40\}$. It means that for $n = 20$, we have $d = 11$ and $m = 180$, for $n = 30$, we have $d = 15$ and $m = 306$, and for $n = 40$, we have $d = 19$ and $m = 443$. In Figure 4, it clearly appears that the hardest instances are located quite close to the theoretical threshold and that the difficulty grows exponentially with n (note the use of a log scale). A similar behavior is observed in Figure 5 with respect to ternary instances generated around the theoretical threshold $p_{cr} \approx 0.63$ for $k = 3$, $\alpha = 1$, $r = 1$ and $n \in \{16, 20, 24\}$. It means that for $n = 16$, we have $d = 16$ and $m = 44$, for $n = 20$, we have $d = 20$ and $m = 60$, and for $n = 24$, we have $d = 24$ and $m = 76$.

Figures 6 and 7 show the results obtained with a tabu search with respect to similar instances (compare with Figures 4 and 5). In fact, for ternary instances, we have limited the number of variables to $n \in \{15, 18, 21\}$. The search effort is given by a median cost since when using an incomplete method, there is absolutely no guarantee of finding a solution in a given limit of time. Remark that all unsatisfiable (unforced) instances below the threshold have been filtered out in order to make a fair comparison. It appears that both complete and incomplete methods behave similarly: the search effort grows exponentially with n and forced instances are (almost) as hard as unforced satisfiable ones.

To deal with classes of instances of larger arities, we have used the model presented at Section 4. We have studied the difficulty of solving with MGAC the instances of model RD^{int} generated around the theoretical threshold $p_{cr} \approx 0.45$ given by Theorem 2 for $k \in \{6, 8\}$, $\alpha = 0.6$, $r = 1$ and $n \in \{10, 15\}$. It means that for $n = 10$, we have $d = 4$ and $m = 23$, and for $n = 15$, we have $d = 5$ and $m = 40$. In Figure 8, one can observe that forced instances are as hard as unforced satisfiable ones. Unsurprisingly, unforced unsatisfiable instances are more difficult.

It is also interesting to observe the behaviour of MGAC when small values of α and r are considered. This is the reason why we have studied the difficulty of solving with MAC the instances of model RD generated around the theoretical threshold $p_{cr} \approx 0.39$ given by Theorem 2 for $k = 2$, $\alpha = 0.51$, $r = 1$ and $n \in \{34, 46, 59, 74, 91, 110\}$. It means that for $n = 34$, we have $d = 6$ and $m = 120$, for $n = 46$, we have $d = 7$ and $m = 176$, for $n = 59$, we have $d = 8$ and $m = 241$, for $n = 74$, we have $d = 9$ and $m = 319$, for $n = 91$, we have $d = 10$ and $m = 411$, for $n = 100$, we have $d = 11$ and $m = 517$. Figure 9

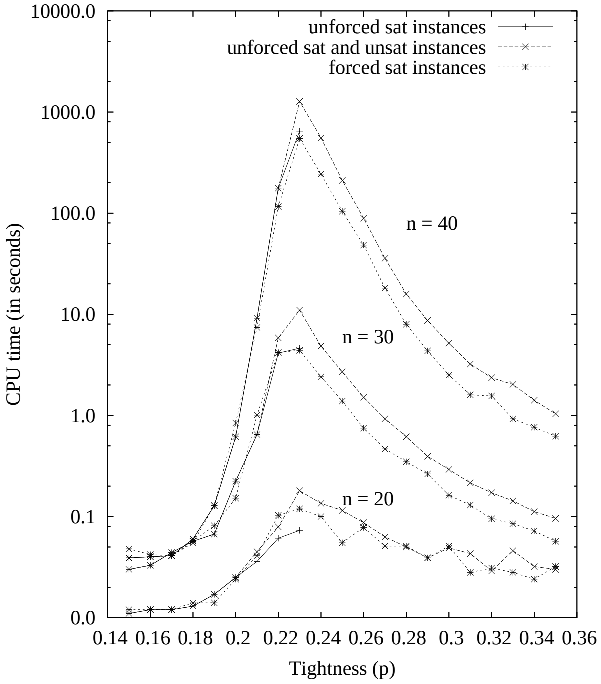

Fig. 4. Mean search cost (50 instances) of solving instances in RB(2,$\{20, 30, 40\}$,0.8,3,p) with MAC

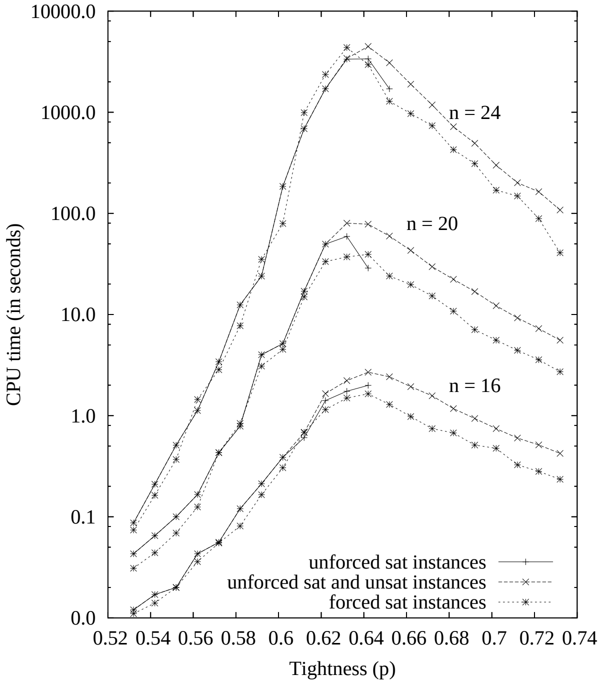

Fig. 5. Mean search cost (50 instances) of solving instances in RB(3,$\{16, 20, 24\}$,1,1,p) with MGAC

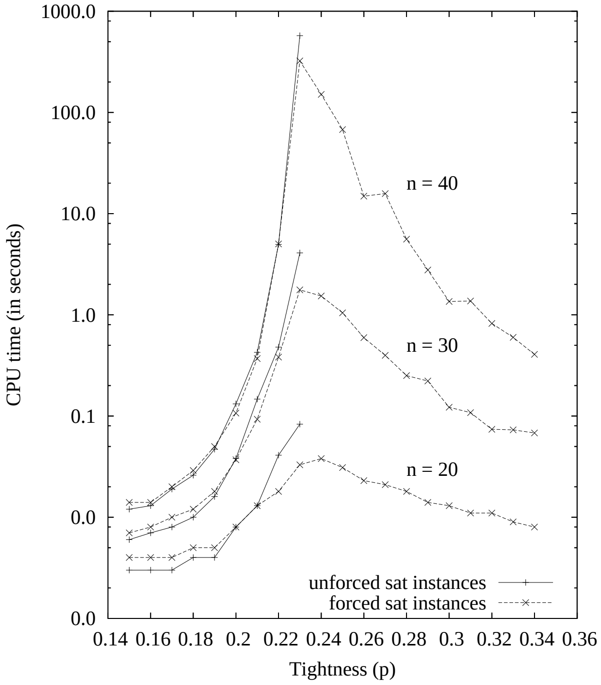

Fig. 6. Median search cost (50 instances) of solving instances in RB(2,$\{20, 30, 40\}$,0.8,3,p) using a tabu search

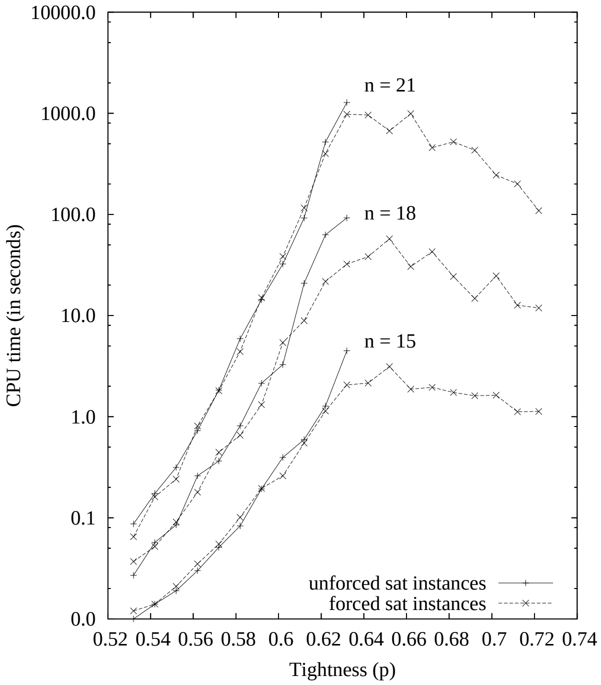

Fig. 7. Median search cost (50 instances) of solving instances in RB(3,$\{15, 18, 21\}$,0.8,3,p) using a tabu search

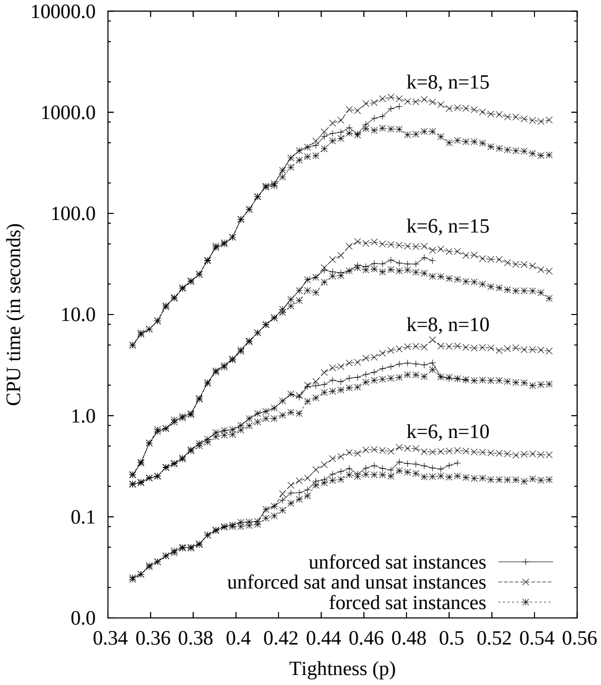

Fig. 8. Mean search cost (50 instances) of solving instances in RD^{int}($\{6, 8\}$,$\{10, 15\}$,0.6,1,p) using MGAC

shows the obtained results (note that n is in $\{34, 46, 59, 74, 91, 110\}$ from bottom to top). It shows the wide applicability of the main result of this paper (i.e. forced instances are almost as hard as unforced instances as long as values of parameters are within the range allowed by the theorems) and that smaller values of α and r make the instances at the threshold much easier to solve, which implies that in such a case, we have to use a large number of variables to get hard instances. Figure 10 depicts the phase transitions (note that n is in $\{34, 46, 59, 74, 91, 110\}$ from left to right). As expected, when n is increased, the size of the phase transition decreases.

As the number and the distribution of solutions are the two most important factors determining the cost of solving satisfiable instances, we can expect, from the analysis given in Section 5, that for model RB, the hardness of solving forced satisfiable instances should be similar to that of solving unforced satisfiable ones. This is what is observed in Figure 4 for example. To confirm this, we have focused our attention to a point just below the threshold as we have then some (asymptotic) guarantee about the difficulty of both unforced and forced instances (see Theorems 5 and 6 in [45]) and the possibility of generating easily unforced satisfiable instances. Figure 11 shows the difficulty of solving with MAC both forced and unforced instances of model RB at $p_{cr} - 0.01 \approx 0.40$ for $k = 2$, $\alpha = 0.8$, $r = 1.5$ and $n \in [20..50]$.

To confirm the inherent difficulty of the (forced and unforced) instances generated at the threshold, we have also studied the runtime distribution produced by a randomized search algorithm on distinct instances [18]. For each instance, we have performed 5000 independent runs. Figure 12 displays the survival function, which

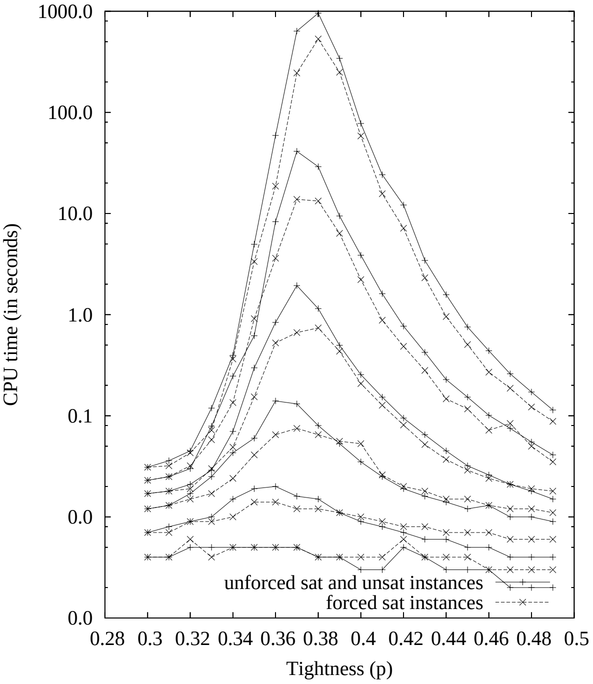

Fig. 9. Mean search cost (50 instances) of solving instances in RD(2,$\{34, 46, 59, 74, 91, 110\}$,$0.51$,1,$p$) using MAC

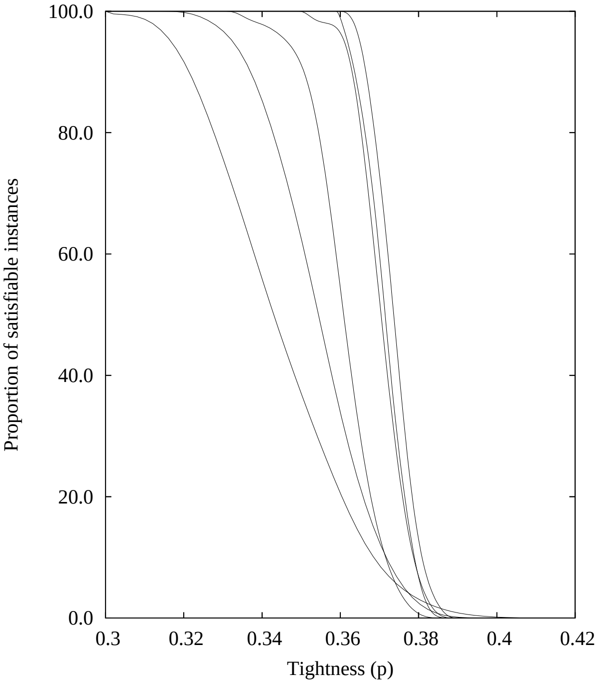

Fig. 10. Phase transitions for RD(2,$\{34, 46, 59, 74, 91, 110\}$,$0.51$,1,$p$)

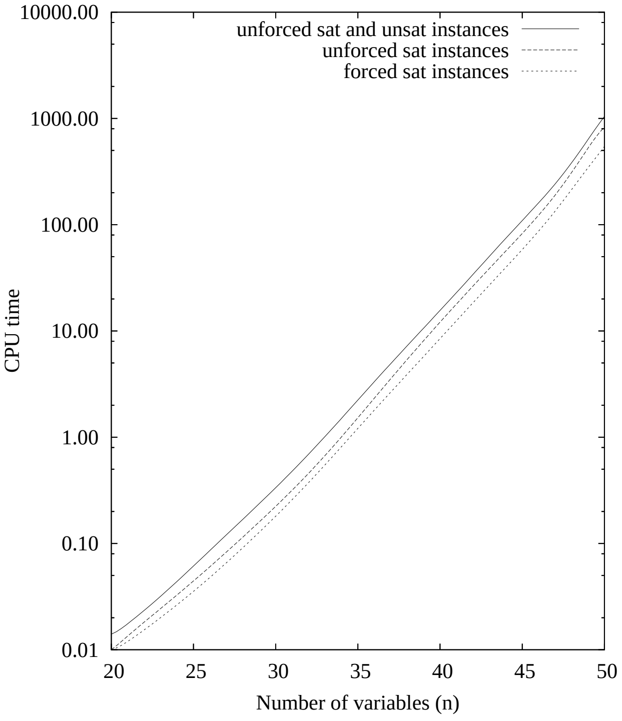

Fig. 11. Mean search cost (50 instances) of solving instances in RB(2,[20..50],0.8,1.5,$p_{cr} - 0.01$) using MAC

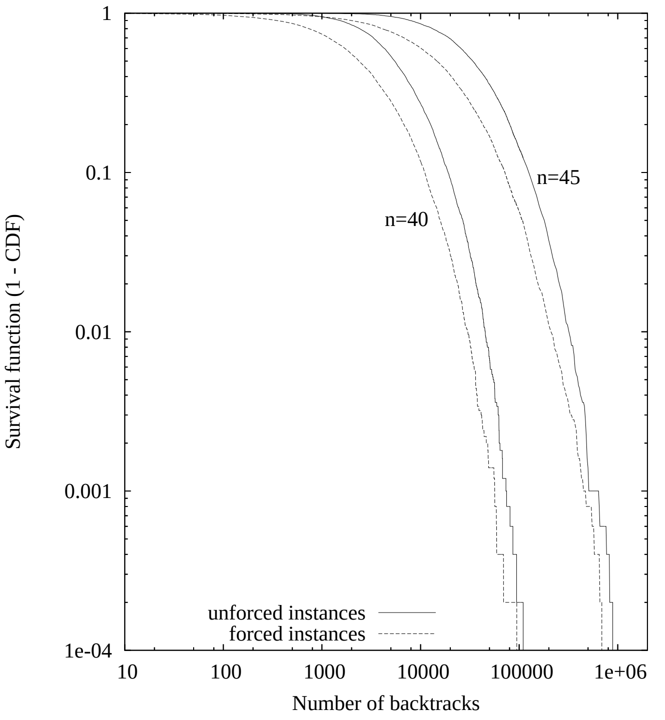

Fig. 12. Non heavy-tailed regime for instances in RB(2,$\{40, 45\}$,0.8,1.5,$p_{cr} \approx 0.41$)

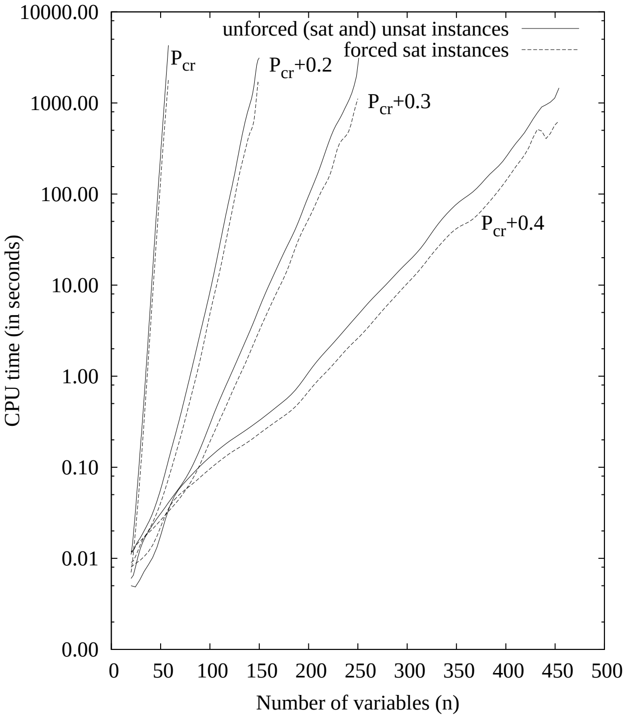

Fig. 13. Mean search cost (50 instances) of solving instances in RB(2,[20..450],0.8,1.5,p) using MAC

corresponds to the probability of a run taking more than x backtracks, of a randomized MAC algorithm for two representative instances generated at $p_{cr} \approx 0.41$ for $k = 2$, $\alpha = 0.8$, $r = 1.5$ and $n \in \{40, 45\}$. One can observe that the runtime distribution (a log-log scale is used) do not correspond to an heavy-tailed one, i.e., a distribution characterized by an extremely long tail with some infinite moment. It means that all runs behave homogeneously and, therefore, it suggests that the instances are inherently hard [18].

Then, we have focused on unforced unsatisfiable instances of model RB as Theorem 3 indicates that such instances have an exponential resolution complexity. We have generated unforced and forced instances with different constraint tightness p above the threshold $p_{cr} \approx 0.41$ for $k = 2$, $\alpha = 0.8$, $r = 1.5$ and $n \in [20..450]$. Figure 13 displays the search effort of a MAC algorithm to solve such instances against the number of variables n. It is interesting to note that the search effort grows exponentially with n, even if the exponent decreases as the tightness increases. Also, although not currently supported by any theoretical result (Theorems 5 and 6 of [45] hold only for forced instances below the threshold) it appears here that forced and unforced instances have a similar hardness.

To further evaluate the hardness of forced satisfiable instances of model RB, we can perform some experiments where the local search algorithm starts with a biased initial assignment which agrees with the hidden assignment on 10%, 30% and 50% variables respectively. Please note that this happens with exponentially small

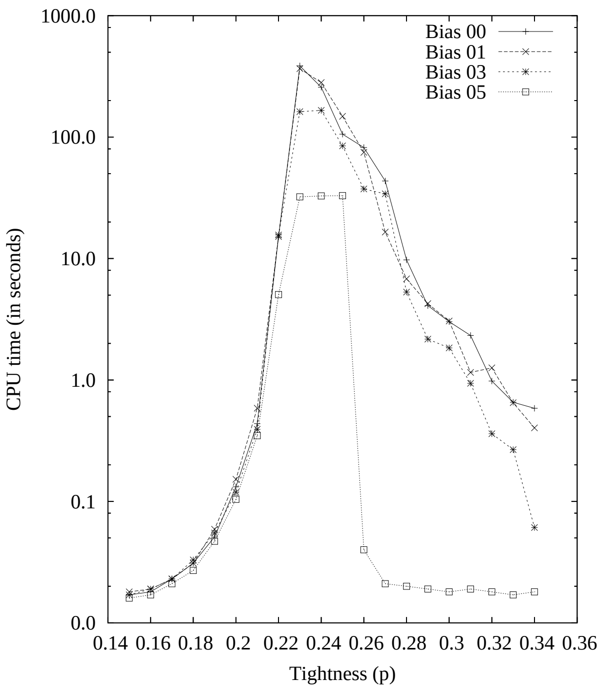

Fig. 14. Median search cost (100 instances) of solving forced instances in RB(2,40,0.8,3,p) using TABU. Initial assignments agree with forced solutions on 10%, 30% and 50% of the variables.

probability. More precisely, for model RB, it is easy to show that for any constant $0 < c \leq 1$, the probability that a random initial assignment agrees with the hidden assignment on at least cn variables is

$$\sum_{i=cn}^{n} \binom{n}{i} \left(\frac{1}{n^{\alpha}}\right)^{i} \left(1 - \frac{1}{n^{\alpha}}\right)^{n-i} < \sum_{i=cn}^{n} 2^{n} \left(\frac{1}{n^{\alpha}}\right)^{cn} < n \left(\frac{2}{n^{\alpha c}}\right)^{n},$$

which converges to 0 exponentially fast as n tends to infinity.

In Figure 14, it appears that unlike random 3-SAT with two complementary hidden solutions [2], the biased initial assignment does not render forced instances of model RB much easier to solve. There is a significant difference, only when the bias reaches 50%.

Finally, we will mention that some instances of model RB were used for different solver competitions. Some forced ternary CSP instances of class RB(3,$\{20, 24, 28\}$, 1,1,≈ 0.63) were used as benchmarks of the 2005 CSP solver competition. No CSP solver succeeded in solving one instance for $n = 28$ within 10 minutes. Forced binary CSP instances of class RB(2,n,0.8,$0.8/ln\frac{4}{3}$,0.25) with n ranging from 40 to 59 were also used as benchmarks of this competition, but no solver succeeded in solving one instance at $n = 53$ (within 10 minutes). These binary instances were also encoded into SAT (using the direct encoding method) and submitted to the SAT competition 2004 (`http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.htm`). About 50% of the competing solvers have succeeded in solving the SAT instances corresponding to $n = 40$ ($d = 19$ and $m = 410$) whereas only one solver has been successful for $n = 50$ ($d = 23$ and $m = 544$).

8 Related Work

In this paper, we have tried to emphasize the nice theoretical and practical properties of models RB. But, as already mentioned in the Section 2, other models of random CSP instances exist. However, we believe that all these models do not offer a framework which is as simple as the one proposed by model RB.

As a related work, we can mention the recent progress on generating hard satisfiable SAT instances. In [5,21], it is proposed to build random satisfiable 3-SAT instances on the basis of a spin-glass model from statistical physics. Instances are generated while using different probabilities in order to select clauses with 0, 1 or 2 negative literals. Another approach, quite easy to implement, has also been proposed in [2]: any 3-SAT instance is forced to be satisfiable by forbidding the clauses violated by both an assignment and its complement.

Finally, let us mention [1] which propose to build random instances with a specific structure, namely, instances of the Quasigroup With Holes (QWH) problem. The hardest instances belong to a new type of phase transition, defined from the number of holes, and coincide with the size of the backbone.

9 Conclusions and Future Work

In this paper, we have first shown that the model(s) RB (and RD) can be used to produce, very easily, hard random instances, i.e., instances whose hardness grows exponentially with the problem size. Importantly, we have demonstrated that the same result holds for instances that are forced to be satisfiable, as long as the existence of the exact phase transition is guaranteed. Then, we have shown that for a wide range of parameter values, practice meets theory. Among other things, our experimental results have confirmed the phase transitions and thresholds predicted by theory, the hardness of forced and unforced satisfiable instances as well as the non heavy-tailed regime for instances of Model RB.

We believe that such results should be of use and value for experimental studies on random CSP instances. We also think that although there are some other ways to generate hard satisfiable instances, e.g. QWH [1] or 2-hidden [2] instances, the simple and natural method presented in this paper, based on model RB, should be well worth further investigation. We propose some interesting directions for future work, including:

(1) extending model RB to allow the easy generation of hard random instances involving constraints of different arities and/or global constraints.
(2) investigating if, for model RB, forced instances are almost as hard as unforced ones when there are more than one forced solution.
(3) proving complexity lower bounds for forced and unforced satisfiable instances of Model RB.

Acknowledgments

We would like to thank the AIJ anonymous referees for their helpful comments and suggestions. We also wish to thank the anonymous referees of IJCAI'05 for their comments on an earlier version of this paper.

References

[1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of AAAI'00*, pages 256–301, 2000.

[2] D. Achlioptas, H. Jia, and C. Moore. Hiding satisfying assignments: two are better than one. In *Proceedings of AAAI'04*, pages 131–136, 2004.

[3] D. Achlioptas, L.M. Kirousis, E. Kranakis, D. Krizanc, M.S.O. Molloy, and Y.C. Stamatiou. Random constraint satisfaction: a more accurate picture. In *Proceedings of CP'97*, pages 107–120, 1997.

[4] D. Achlioptas and C. Moore. The asymptotic order of the random k-SAT threshold. In *Proceedings of FOCS'02*, pages 779–788, 2002.

[5] W. Barthel, A.K. Hartmann, M. Leone, F. Ricci-Tersenghi, M. Weigt, and R. Zecchina. Hiding solutions in random satisfiability problems: a statistical mechanics approach. *Physical review letters*, 88(18), 2002.

[6] E. Ben-Sasson and A. Wigderson. Short proofs are narrow - resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

[7] C. Bessiere and J. Régin. Arc consistency for general constraint networks: preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, 1997.

[8] C. Bessiere, J.C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185, 2005.

[9] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

[10] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proceedings of IJCAI'91*, pages 331–337, 1991.

[11] S.A. Cook and D.G. Mitchell. Finding hard instances of the satisfiability problem: a survey. *DIMACS series in discrete mathematics and theoretical computer science*, 35, 1997.

[12] R. Debruyne and C. Bessiere. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.

[13] O. Dubois, Y. Boufkhad, and J. Mandler. Typical random 3-sat formulae and the satisfiability threshold. In *Proceedings of SODA'00*, pages 126–127, 2000.

[14] A.M. Frieze and M. Molloy. The satisfiability threshold for randomly generated binary constraint satisfaction problems. In *Proceedings of Random'03*, pages 275–289, 2003.

[15] Y. Gao and J. Culberson. Consistency and random constraint satisfaction models with a high constraint tightness. In *Proceedings of CP'04*, pages 17–31, 2004.

[16] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. Random constraint satisfaction: flaws and structure. *Journal of Constraints*, 6(4):345–372, 2001.

[17] I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The Constrainedness of Search. In *Proceedings of AAAI-96*, pages 246–252, 1996.

[18] C.P. Gomes, C. Fernández, B. Selman, and C. Bessiere. Statistical regimes across constrainedness regions. In *Proceedings of CP'04*, pages 32–46, 2004.

[19] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[20] R. Impagliazzo, L. levin, and M. Luby. Pseudo-random number generation from one-way functions. In *Proceedings of STOC'89*, pages 12–24, 1989.

[21] H. Jia, C. Moore, and B. Selman. From spin glasses to hard satisfiable formulas. In *Proceedings of SAT'04*, 2004.

[22] A.C. Kaporis, L.M. Kirousis, and E.G. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. In *Proceedings of ESA'02*, pages 574–585, 2002.

[23] C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In *Proceedings of CP'03*, pages 480–494, 2003.

[24] C. Lecoutre, F. Boussemart, and F. Hemery. Implicit random CSPs. In *Proceedings of ICTAI'03*, pages 482–486, 2003.

[25] C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-based techniques versus conflict-directed heuristics. In *Proceedings of ICTAI'04*, pages 549–557, 2004.

[26] C. Lecoutre and R. Szymanek. Generalized arc consistency for positive table constraints. In *Proceedings of CP'06*, pages 284–298, 2006.

[27] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[28] B. Mazure, L. Sais, and E. Gregoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.

[29] D.G. Mitchell. Some random CSPs are hard for resolution. *Submitted*, 2000.

[30] D.G. Mitchell. Resolution complexity of random constraints. In *Proceedings of CP'02*, pages 295–309, 2002.

[31] M. Molloy. Models for random constraint satisfaction problems. *SIAM Journal of computing*, 32(4):935–949, 2003.

[32] P. Morris. The breakout method for escaping from local minima. In *Proceedings of AAAI'93*, pages 40–45, 1993.

[33] NIST. *Secure Hash Standard.* National Institute of Standards and Technology, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf, 2002. FIPS 180-2.

[34] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.

[35] R. Rivest. *The MD5 Message-Digest Algorithm.* MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992. Request for Comments 1321.

[36] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of CP'94*, pages 10–20, 1994.

[37] B. Selman and H. Kautz. Domain-independent extensions to GSAT: solving large structured satisfiability problems. In *Proceedings of IJCAI'93*, pages 290–295, 1993.

[38] B.M. Smith. Constructing an asymptotic phase transition in random binary constraint satisfaction problems. *Theoretical Computer Science*, 265:265–283, 2001.

[39] B.M. Smith and M.E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155–181, 1996.

[40] J.R. Thornton. *Constraint weighting local search for constraint satisfaction.* PhD thesis, Griffith University, Australia, 2000.

[41] C. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.

[42] K. Xu. *A study on the phase transitions of SAT and CSP (in chinese).* PhD thesis, Beihang University, 2000.

[43] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. A simple model to generate hard satisfiable instances. In *Proceedings of IJCAI'05*, pages 337–342, 2005.

[44] K. Xu and W. Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12:93–103, 2000.

[45] K. Xu and W. Li. Many hard examples in exact phase transitions with application to generating hard satisfiable instances. Technical report, CoRR Report cs.CC/0302001, 2003. Revised version in *Theoretical Computer Science*, 355(2006):291-302.