Simple Random Logic Programs

Gayathri Namasivayam and Mirosław Truszczyński

Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046, USA

Abstract. We consider random logic programs with two-literal rules and study their properties. In particular, we obtain results on the probability that random "sparse" and "dense" programs with two-literal rules have answer sets. We study experimentally how hard it is to compute answer sets of such programs. For programs that are *constraint-free* and *purely negative* we show that the easy-hard-easy pattern emerges. We provide arguments to explain that behavior. We also show that the hardness of programs from the hard region grows quickly with the number of atoms. Our results point to the importance of purely negative constraint-free programs for the development of ASP solvers.

1 Introduction

The availability of a simple model of a random CNF theory was one of the enabling factors behind the development of fast satisfiability testing programs — *SAT solvers*. The model constrains the length of each clause to a fixed integer, say k, and classifies k-CNF theories according to their *density*, that is, the ratio of the number of clauses to the number of atoms. k-CNF theories with low densities have few clauses relative to the number of atoms. Thus, most of them have many solutions, and solutions are easy to find. k-CNF theories with high densities have many clauses relative to the number of atoms. Thus, most of them are unsatisfiable. Moreover, due to the abundance of clauses, proofs of contradiction are easy to find. As theories in low- and high-density regions are "easy," they played essentially no role in the development of SAT solvers.

There is, however, a narrow range of densities "in between," called the *phase transition*, where random k-CNF theories change *rapidly* from most being satisfiable to most being unsatisfiable. Somewhere in that narrow range is a value d such that random k-CNF theories with density d are satisfiable with the probability 1/2. The problem of determining that value has received much attention. For instance, for 3-CNF theories, the phase-transition density was found experimentally to be about 4.25 [1]. A paper by Achlioptas discusses recent progress on the problem, including some lower and upper bounds on the phase transition value [2]. A key property of 3-CNF theories from the phase transition region is that they are hard.[1] Thus, we have the easy-hard-easy difficulty pattern as

[1] It should be noted that the low- and high-density regions also contain challenging theories, but they are relatively rare [3]).

the function of density. Moreover, deciding satisfiability of programs from the hard region is very hard indeed! Designing solvers that could solve random unsatisfiable 3-CNF theories with 700 atoms generated from the phase-transition region was one of grand challenges for SAT research posed by Selman, Kautz and McAllester [4]. It resulted in major advances in SAT solver technology.

As in the case of the SAT research, work on *random logic programs* is likely to lead to new insights into the properties of *answer sets* of programs, and lead to advances in *ASP solvers* — software for computing them. Yet, the question of models of random logic programs has received little attention so far, with the work of Zhao and Lin [5] being a notable exception. Our objective is to propose a model of simple random logic programs and investigate its properties.

As in SAT, we consider random programs with rules of the same length. For the present study, we further restrict our attention to programs with two-literal rules. These programs are simple, which facilitates theoretical studies. But despite their simplicity, they are of considerable interest. First, every problem in NP can be reduced in polynomial time to the problem of deciding the existence of an answer set of a program of that type [6]. Second, many problems of interest have a simple encoding in terms of such programs [7]. We study experimentally and analytically properties of programs with two-literal rules. We obtain results on the probability that random programs with two-literal rules, both "sparse" and "dense," have answer sets. We study experimentally how hard it is to compute answer sets of such programs. We show that for programs that are *constraint-free* and *purely negative* the easy-hard-easy pattern emerges. We give arguments to explain that phenomenon, and show that the hardness of programs from the hard region grows quickly with the number of atoms. Our results point to the importance of constraint-free purely negative programs for the development of ASP solvers, as they can serve as useful benchmarks when developing good search heuristics. However, unlike in the case of SAT, depending on the parameters of the model, we either do not observe the phase transition or, when we do, it is gradual not sudden.

Even relatively small programs from the hard region are very hard for the current generation of ASP solvers. Interestingly, that observation may also have implications for the design of SAT solvers. If P is a purely negative program, answer sets of P are models of its completion $comp(P)$, a certain propositional theory [8]. For programs with two-literal rules the completion is (essentially) a CNF theory. Our experiments showed that these theories are very hard for the present-day SAT solvers, despite the fact that most of their clauses are binary.

2 Preliminaries

Logic programs consist of *rules*, that is, of expressions of the form

$$a \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n \tag{1}$$

and

$$\leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n, \tag{2}$$

where a, b_i and c_j are atoms. Rules (1) are called *definite*, and rules (2) — *constraints*. A rule is *proper* if no atom occurs in it more than once. A rule is *k-regular* if it consists of k literals (that is, it is a definite rule with $k-1$ literals in the body, or a constraint with k literals in the body).

If r is a rule of type (1) or (2), the expression $b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$ (understood as the conjunction of its literals) is the *body* of r. We denote it by $bd(r)$. The set of atoms $\{b_1, \ldots, b_m\}$ is the *positive* body of r, denoted $bd^+(r)$, and the set of atoms $\{c_1, \ldots, c_n\}$ is the *negative* body of r, denoted $bd^-(r)$. In addition, the *head* of r, $hd(r)$, is defined as a, if r is of type (1), and as $\bot$, otherwise. A program P is *constraint-free* if it contains no constraints. A program P is *purely negative* if for every non-constraint rule $r \in P$, $bd^+(r) = \emptyset$.

A set of atoms M is an *answer set* of a program P if it is the least model of the *reduct* of P with respect to M, that is, the program P^M obtained by removing from P every rule r such that $M \cap bd^-(r) \neq \emptyset$, and by removing all literals of the form *not* c from all other rules of P.

Computing answer sets of propositional logic programs is the basic reasoning task of answer-set programming, and fast programs that can do that, known as *answer-set programming solvers* (*ASP solvers*, for short) have been developed in the recent years [9–13].

3 2-Regular Programs

We assume a fixed set of atoms $At = \{a_1, a_2, \ldots\}$. There are five types of 2-regular rules: $a \leftarrow not\ b$; $a \leftarrow b$; $\leftarrow not\ a, not\ b$; $\leftarrow a, not\ b$; $\leftarrow a, b$. Accordingly, we define five classes of programs, mR_n^-, mR_n^+, mC_n^-, $mC_n^\pm$, and mC_n^+, with atoms from $At_n = \{a_1, \ldots, a_n\}$ and consisting of m *proper* rules of each of these types, respectively. Without the reference to m, the notation refers to all programs with n atoms of the corresponding type (for instance, R_n^+ stands for the class of all programs over At_n consisting of proper rules of the form $a \leftarrow b$).

The maximum value of m for which mR_n^-, mR_n^+ and $mC_n^\pm$ are not empty is $n(n-1)$. The maximum value of m for which mC_n^- and mC_n^+ are not empty is $n(n-1)/2$. Let $0 \leq m_1, m_2, c_2 \leq n(n-1)$ and $0 \leq c_1, c_3 \leq n(n-1)/2$ be integers. By $[m_1R^- + m_2R^+ + c_1C^- + c_2C^\pm + c_3C^+]_n$ we denote the class of programs P that are unions of programs from the corresponding classes. We refer to these programs as *components* of P. If any of the integers m_i and c_i is 0, we omit the corresponding term from the notation. When we do not specify the numbers of rules, we allow any programs from the corresponding classes. For instance, $[R^- + R^+ + C^- + C^\pm + C^+]_n$ stands for the class of all proper programs with atoms from At_n.

Given integers n and m, it is easy to generate uniformly at random programs from each class mR_n^-, mR_n^+, mC_n^-, $mC_n^\pm$, and mC_n^+. For instance, a random program from mR_n^- can be viewed as the result of a process in which we start with the empty program on the set of atoms At_n and then, in each step, we add a randomly generated proper rule of the form $a \leftarrow not\ b$, with repeating rules discarded, until m rules are generated. This approach generalizes easily

to programs from other classes we consider, in particular, to programs from $[m_1R^- + m_2R^+ + c_1C^- + c_2C^{\pm} + c_3C^+]_n$. Our goal is to study properties of such random programs.

We start with a general observation. If $P \in [m_2R^+ + c_1C^- + c_2C^{\pm} + c_3C^+]_n$ ($m_1 = 0$), then either P has no answer sets (if $c_1 \neq 0$) or, otherwise, $\emptyset$ is a unique answer set of P. Thus, in order to obtain interesting classes of programs, we must have $m_1 > 0$. In other words, programs from R_n^- (proper purely negative and constraint-free) play a key role.

4 The Probability of a Program to Have an Answer Set

We study first the probability that a random program in the class $[m_1R^- + m_2R^+ + c_1C^- + c_2C^{\pm} + c_3C^+]_n$ has an answer set. In several places we use results from random graph theory [14, 15]. To this end, we exploit graphs associated with programs. Namely, with a program $P \in [R^- + R^+ + C^{\pm}]_n$ we associate a *directed* graph $D(P)$ with the vertex set At_n, in which a is connected to b with a directed edge (a, b) if $b \leftarrow not\ a$, $b \leftarrow a$ or $\leftarrow b, not\ a$ is a rule of P. For $P \in [R^- + R^+]_n$, the graph $D(P)$ is known as the *dependency* graph of a program. Similarly, with a program $P \in [R^- + R^+ + C^- + C^{\pm} + C^+]_n$ we associate an undirected graph $G(P)$ with the vertex set At_n, in which a is connected to b with an *undirected* edge $\{a, b\}$ if a and b appear together in a rule of P. If $P \in [R^- + R^+ + C^{\pm}]_n$, then $D(P)$ may have fewer edges than P has rules (the rules $a \leftarrow not\ b$, $a \leftarrow b$ and $\leftarrow b, not\ a$ determine the same edge). A similar observation holds for $G(P)$.

These graphs contain much information about the underlying programs. For instance, it is well known that if $P \in [R^- + R^+]_n$ and $D(P)$ has no cycles then P has a unique answer set. Similarly, if $P \in [m_1R^- + m_2R^+ + c_1C^- + c_2C^{\pm} + c_3C^+]_n$ and M is an answer set of P then $\overline{M}$ is an independent set in the graph $G(P_1)$, where P_1 is the component of P from $m_1R_n^-$.

We denote by $\mathcal{AS}^+$ the class of all programs over At that have answer sets. We write $Prob(P \in \mathcal{AS}^+)$ for the probability that a random graph P from one of the classes defined above has an answer set. That probability depends on n (technically, it also depends on the numbers of rules of particular types but, whenever it is so, the relevant numbers are themselves expressed as functions of n). We are interested in understanding the behavior of $Prob(P \in \mathcal{AS}^+)$ for random programs P from the class $[R^- + R^+ + C^- + C^{\pm} + C^+]_n$ (or one of its subclasses). More specifically, we will investigate $Prob(P \in \mathcal{AS}^+)$ as n grows to infinity. If $Prob(P \in \mathcal{AS}^+) \to 1$ as $n \to \infty$, we say that P *asymptotically almost surely*, or *a.a.s* for short, has answer sets. If $Prob(P \in \mathcal{AS}^+) \to 0$ as $n \to \infty$, we say that P a.a.s. has no answer sets.

To ground our results in some intuitions, we first consider the probability that a program from mR_{150}^- has an answer set as a function of the density $d = m/150$ (or equivalently, the number of edges m). The graphs, shown in Figure 1, were obtained experimentally. For each value of d, we generated 1000 graphs from the set mR_{150}^-, where $m = 150d$. The graph on the left shows the behavior of the

probability across the entire range of d. The graph on the right shows in more detail the behavior for small densities.

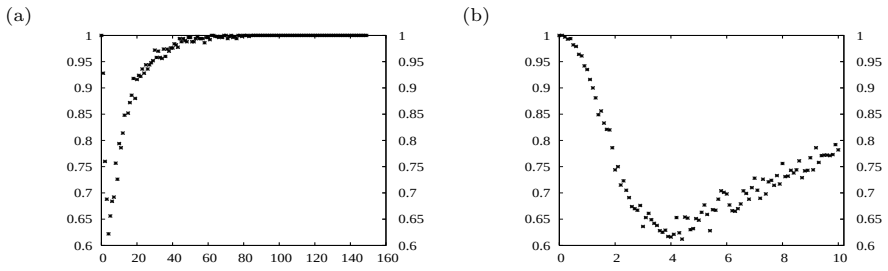

Fig. 1. The probability that a graph from mR^-_{150} ($m = 150d$) has an answer set, as a function of d.

The graphs show that the probability is close to 1 for very small densities, then drops rapidly. After reaching a low point (around 0.6, in this case), it starts getting larger again and, eventually, reaches 1. We also note that the rate of drop is faster than the rate of ascent. We will now present theoretical results that quantify some of these observations. Our results concern the two extremes: programs of low density and graphs of high density.

We start with programs of low density and assume first that they do not have constraints. In this case, the results do not depend on whether or not we allow positive rules.

Theorem 1. *If $m_1 + m_2 = o(n)$ and $P \in [m_1R^- + m_2R^+]_n$, then P a.a.s has a unique answer set.*

Proof. (Sketch) Let P be a random program from $[m_1R^- + m_2R^+]_n$. The directed graph $D(P)$ can be viewed as a random directed graph with n vertices, and $m' = o(n)$ edges ($m' \leq m$, as different rules in P may map onto the same edge). Thus, $D(P)$ a.a.s. has no directed cycles (the claim can be derived from the property of random *undirected* graphs: a random undirected graph with n vertices and $o(n)$ edges a.a.s. has no cycles [15]). It follows that P a.a.s. has a unique answer set. □

If there are constraints in the program, the situation changes. Even a single constraint of the form $\leftarrow$ *not a, not b* renders a sparse random program inconsistent.

Corollary 1. *If $c_1 \geq 1$, $m_1 + m_2 = o(n)$, and P is a random program from $[m_1R^- + m_2R^+ + c_1C^-]_n$, then P a.a.s. has no answer sets.*

Proof. Let P be a random program from $[m_1R^- + m_2R^+ + c_1C^-]_n$. Then, $P = P_1 \cup P_2$, where P_1 is a random program from $[m_1R^- + m_2R^+]_n$ and P_2 is a random program from $c_1C^-_n$. By Theorem 1, P_1 a.s.s. has a unique answer set, say M. Since P_1 has $o(n)$ non-constraint rules, $|M| = o(n)$. The probability that

a randomly selected constraint of the form $\leftarrow not\ a, not\ b$ is violated by M is given by the probability that $\{a, b\} \cap M = \emptyset$. That probability is $\binom{n-o(n)}{2}/\binom{n}{2}$ and it converges to 1 as $n \to \infty$. Thus, the assertion follows. □

If we exclude such constraints then there again is a small initial interval of densities, for which random programs are consistent with high probability.

Corollary 2. *If $c_1 = 0$, $c_2 + c_3 \geq 1$, $(m_1 + m_2)c_2 = o(n)$, $(m_1 + m_2)^2 c_3 = o(n^2)$, and P is a random program from $[m_1 R^- + m_2 R^+ + c_2 C^{\pm} + c_3 C^+]_n$, then P a.a.s. has an answer set.*

Proof. (Sketch) Let P be a random program from $[m_1 R^- + m_2 R^+ + c_2 C^{\pm} + c_3 C^+]_n$. Thus, $P = P_1 \cup P_2 \cup P_3$, where P_1, P_2 and P_3 are random programs from $[m_1 R^- + m_2 R^+]_n$, $c_2 C_n^{\pm}$ and $c_3 C_n^+$, respectively. Since $c_2 > 0$ or $c_3 > 0$, $m_1 + m_2 = o(n)$. By Theorem 1, P_1 a.a.s. has a unique answer set, say M. Moreover, the size of M is at most $m_1 + m_2$. Under the assumptions of the corollary, one can show that a.a.s. each constraint $\leftarrow a, not\ b$ in P_2 has no atoms in M, and a.a.s. each constraint $\leftarrow a, b$ in P_3 has at most one atom in M. Thus, a.a.s. programs P_2 and P_3 are satisfied by M. Consequently, P a.a.s. has M as its unique answer set of P. □

We move on to programs of high density. Our first result concerns programs from R_n^- (proper, purely negative and constraint-free programs with n atoms).

Theorem 2. *Let $0 < c < 1$ be a constant. For every fixed x, a random program from mR_n^-, where $m = \lfloor cN + x\sqrt{c(c-1)N} \rfloor$ and $N = n(n-1)$, a.a.s. has an answer set.*

Proof. (Sketch) To show the assertion, it is enough prove that a random directed graph with n vertices and m edges, where m is as in the statement of the theorem, a.a.s. has a kernel. It is known [16] that a.a.s. a random directed graph with n nodes drawn from the *binomial model* (edges are selected independently of each other and with the same probability c) has a kernel. Moreover, one can show that if $m' > m$, $m' = m + O(n)$, and G_m and $G_{m'}$ are random directed graphs with n nodes, and m and m' edges, respectively, then $Prob(G_m \text{ has a kernel}) \leq Prob(G_{m'} \text{ has a kernel}) + o(1)$. That property can be used instead of convexity in Theorem 2(ii) [14], which allows us to transform properties of graphs from the binomial model into properties of graphs from the *uniform* model that we are considering. Thus, the assertion follows. □

Theorem 2 concerns only a narrow class of dense programs, its applicability being limited by the specific number of rules programs are to have ($m = \lfloor cN + x\sqrt{c(c-1)N} \rfloor$, where $N = n(n-1)$). It also does not apply to "very" dense graphs with $m = n^2 - o(n^2)$ rules. However, based on that theorem and on experimental results (Figure 1), we conjecture that for every $c > 0$, a program from mR_n^-, where $m \geq cn^2$, a.a.s. has an answer set.

We will now consider the effect of adding positive rules (rules of the form $a \leftarrow b$) and constraints. In fact, as soon as we have just slightly more than $n \log n$ positive rules in a random program that program a.a.s. has no answer sets.

Theorem 3. *For every $\epsilon > 0$, if $m_1 \geq 1$, $m_2 \geq (1+\epsilon)n \log n$, and P is a random program from $[m_1 R^- + m_2 R^+ + c_1 C^- + c_2 C^{\pm} + c_3 C^+]_n$, then P a.a.s. has no answer sets.*

Proof. Let $P \in [m_1 R^- + m_2 R^+ + c_1 C^- + c_2 C^{\pm} + c_3 C^+]_n$, where $m_1 \geq 1$. Also, let P_2 be the component of P from $m_2 R^+$. If $D(P_2)$ contains a Hamiltonian cycle, then P has no answer sets. Indeed, $\emptyset$ is not a answer set due to the rule of the form $a \leftarrow not\ b$ that is present in P. Thus, if P has an answer set, say M, then $M \neq \emptyset$. Clearly, P^M contains P_2. By the assumption on $D(P_2)$, the least model of P^M contains all atoms in At_n. Thus, $M = At_n$. But then, P^M contains no atoms (all its rules are either from P_2 or are constraints of the form $\leftarrow a, b$) and so, the least model of P^M is $\emptyset$, a contradiction. Clearly, there is a precise correspondence between programs from $m_2 R^+$ and random directed graphs with n nodes and m edges (no loops). The assertion follows now from the result that states that a random directed graph with n nodes and at least $(1+\epsilon)n \log n$ edges a.a.s. has a Hamiltonian cycle [14]. □

The presence of sufficiently many constraints of the form $\leftarrow a, b$ or $\leftarrow a, not\ b$ also eliminates answer sets. To see that, we recall that if M is an answer set of a program $P = P_1 \cup P_2$, where $P_1 \in R_n^-$ and $P_2 \in [R^+ + C^- + C^{\pm} + C^+]_n$, then M is the complement of an independent set in $G(P_1)$. The following property will be useful. For every real $c > 0$ there is a real $d > 0$ such that a.a.s. a graph with n vertices and $m \geq cn^2$ edges has no independent set with more than $d \log n$ elements [14]. Thus, we get the following result that provides a lower bound on the size of an answer set in a dense random logic program.

Theorem 4. *For every real $c > 0$, there is a real $d > 0$ such that a.a.s. the complement of every answer set of a random program $P = P_1 \cup P_2$, where $P_1 \in mR_n^-$, $P_2 \in [R^+ + C^- + C^{\pm} + C^+]_n$ and $m \geq cn^2$, has size at most $d \log n$.*

We now consider the effect of constraints of the form $\leftarrow a, b$ on the existence of answer sets in programs with many purely negative rules. Intuitively, even a small number of such constraints should suffice to "kill" all answer sets. Indeed, according to Theorem 4, these answer sets are large and contain "almost all" atoms. Formalizing this intuition, we get the following result.

Theorem 5. *For every $c > 0$ there is $d > 0$ such that if $m_1 \geq cn^2$, $c_3 \geq d \log n + 1$, and P is a random program from $[m_1 R^- + m_2 R^+ + c_1 C^- + c_2 C^{\pm} + c_3 C^+]_n$, then P a.a.s. has no answer sets.*

Constraints of the form $\leftarrow a, not\ b$ do not have such a dramatic effect. However, a still relatively small number of such constraints a.a.s. eliminates all answer sets.

Theorem 6. *For every $c > 0$, and for every $\epsilon > 0$, if $m_1 \geq cn^2$, $c_2 \geq n^{1+\epsilon}$, and P is a random program form $[m_1 R^- + m_2 R^+ + c_1 C^- + c_2 C^{\pm} + c_3 C^+]_n$, then a.a.s. P has no answer sets.*

Proof. We set $N = n(n-1)$ and $N_i = i(n-i)$. Let $X \subseteq At_n$ consist of $n-i$ elements, where $0 < i < n$. We will first compute the probability that in a random directed graph with the set of vertices At_n and with m edges, there is no edge starting in $\overline{X}$ and ending in X. That probability is given by $\binom{N-N_i}{m}/\binom{N}{m}$. One can show that it can be bounded from above by $(1-N_i/N)^m$. It follows that the probability that at least one $X \subseteq At_n$ such that $0 < |\overline{X}| \leq k$ has that property is bounded by $\sum_{i=1}^{k}\binom{n}{i}(1-N_i/N)^m$. Let $d > 0$ be a constant. One can show that for every $\epsilon > 0$, if $k \leq d\log n$ and $m \geq n^{1+\epsilon}$, then $\sum_{i=1}^{k}\binom{n}{i}(1-N_i/N)^m \to 0$ as $n \to \infty$. Let us interpret that result in terms of programs. Let d be a constant such that the complement of every answer set in a random program P from $[m_1R^- + m_2R^+ + c_1C^- + c_2C^{\pm} + c_3C^+]_n$ has size at most $d\log n$ (such d exists by Theorem 4) and let ϵ be any fixed positive real. Let Q be the component from $c_2C_n^{\pm}$ of P. Then $D(Q)$ has at least $n^{1+\epsilon}$ edges. Thus, a.a.s. for every set X such that $1 \leq |\overline{X}| \leq d\log n$, there is an edge (a,b) in $D(P)$ that originates in $\overline{X}$ and ends in X. Such edge corresponds to a constraint $\leftarrow b, not\ a$ in Q. Clearly, this constraint is violated by X. Thus, a.a.s. P has no non-empty answer sets. Since $X = At_n$ is not an answer set either (the reduct of P wrt At_n contains no atoms and so, it is inconsistent or its least model is empty), a.a.s. P has no answer sets. □

The case of constraints $\leftarrow not\ a, not\ b$ is less interesting. Large answer sets (having at least $n - d\log n$ atoms) that arise for programs with dense component from R_n^- typically satisfy them and to "kill" all answer sets of such programs with high probability almost all constraints $\leftarrow not\ a, not\ b$ must be present.

5 Hardness of Programs

We will now study the hardness of programs from $[m_1R^- + m_2R^+ + c_1C^- + c_2C^{\pm} + c_3C^+]_n$ for ASP solvers. The bulk of our experimental results concern programs in the class R_n^-. It turns out these programs (for appropriately chosen density) are especially challenging.

Unless stated otherwise, our experiments separate programs that have answer sets (are *consistent*) from those that do not (are *inconsistent*). For each experiment we generate a sample of instances of programs of each of these two types. In the previous section we provided evidence that programs in mR_n^-, where $m \geq cn^2$ (cf. Figure 1 and Theorem 2), a.a.s. have an answer set. Therefore, when experimenting with inconsistent programs, we restrict the number of rules in a program to values for which inconsistent programs appear with probability sufficiently larger than 0 (about 0.05) to allow for building samples of inconsistent programs of sizes large enough to justify drawing conclusions from experiments (typically 100 programs per sample).

In experiments, we used *smodels* (with lookahead) [9] and *clasp* [10]. We took the average number of choice points as reported by these systems as the measure of the *hardness* of a family of programs.

Our first observation is that as we increase m, programs from mR_n^- show the easy-hard-easy pattern. That is, low-density programs are easy for the two

solvers. When m grows, programs get harder. Then, at some point, they start getting easier again. We illustrate that behavior in Figure 2 below. The two graphs show separately the results for consistent and inconsistent programs from the classes mR^-_{100}. Each figure shows together the results (average number of choice points) for *smodels* (the scale on the right) and *clasp* (the scale on the left). The x-axis shows the density, that is, the ratio of the number of rules to the number of atoms in a program. We stress that the scales differ. Thus, the figures are not meant to compare the performance of *smodels* and *clasp*. But they do show that for each solver a similar easy-hard-easy pattern emerges, and that the features of the pattern are remarkably similar for the two solvers.

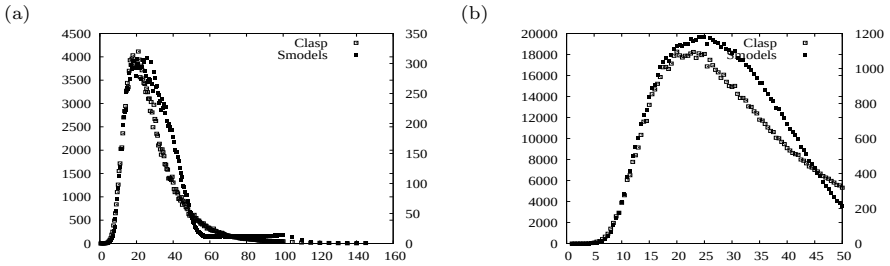

Fig. 2. Average number of choice points for consistent (graph (a)) and inconsistent (graph(b)) programs with 150 atoms; *smodels* (scale on the right) and *clasp* (scale on the left). The x-axis represents the density. Sample sizes are 500 for consistent programs, and 100 for inconsistent programs.

We obtained the same type of a pattern in our experiments with programs with 125, 175 and 200 atoms. However, we observed some minor deviations from that pattern for *smodels* (but not for *clasp*) for programs with 100 atoms. Given our results for $n \geq 125$, it seems plausible that the irregular behavior arises only for some smaller numbers of atoms.

We used the term *hard region* above somewhat informally. To make that concept more precise, we define it now as the maximum interval $[u, v]$ such that for every density $d \in [u, v]$ the average number of choice points is at least 90% of the maximum (peak) average number of choice points. Table 1 shows the hard regions, the density for which the number of choice points reaches the maximum, and the number of choice points at the peak location for consistent and inconsistent instances with $n = 125, 150, 175$ and 200 atoms. The key observations are: (1) the location of the hard region does not seem to depend much on the solver; it is centered around the density of 19 for consistent programs, and 22 for inconsistent ones, (2) inconsistent programs are significantly harder than consistent ones, (3) the peak of hardness is not sharp or, in other words, the hard region extends over a sizable range of densities, and (4) the hardness of programs in the hard region grows very quickly.

We conclude with arguments to explain the presence of the easy-hard-easy pattern we observed for programs in the class R^-_n. First, we note that programs in mR^-, where $m = o(n)$, a.a.s. are stratified (Theorem 1). Computing answer

Inconsistent programs						
	clasp			*smodels*		
n	hard region	peak	choice points at peak	hard region	peak	choice points at peak
125	$[17.5-27]$	22	5261	$[17.5-24]$	21	388
150	$[18-27]$	23	18639	$[19-31]$	24.5	1184
175	$[18.5-27.5]$	22	59704	$[17.5-23.5]$	20.5	3582
200	$[18-28]$	22	189576	$[18-26]$	22.5	14407
Consistent programs						
125	$[15.5-21.5]$	17.5	1231	$[16-25]$	20	130
150	$[16-23]$	17.5	4033	$[16-29.5]$	20	308
175	$[18.5-21.5]$	20	14230	$[17.5-21.5]$	20	1110
200	$[17.5-23]$	19.5	43345	$[18.5-24.5]$	19.5	4232

Table 1. Hard region, peak location, and the number of choice points at the peak location for consistent and inconsistent programs. Results for *clasp* and *smodels*.

sets for such programs is easy. As the density (the number of rules) grows, cycles in the graph $D(P)$ start appearing (that happens roughly when a program has as many rules as atoms). Initially, there are few cycles and the increase in hardness is slow. At some point, however, there are enough cycles in $D(P)$ to make computing answer sets of P hard. To explain why the task gets easier again, we note the following property of binary trees.

Proposition 1. *Let T be a binary tree with m leaves, the height n, and with the number of left edges on any path from the root to a leaf bounded by k. Then $m \leq 2^k \binom{n}{k}$.*

Proof: Let $S(n,k)$ be the maximum number of leaves in such a tree. Then $S(n,k)$ is given by the recursive formula $S(n,k) = S(n-1,k) + S(n-1,k-1)$, for $n \geq k+1$ and $k \geq 1$, with the initial conditions $S(n,0) = 1$ and $S(n,n) = 2^n$, for $n \geq 0$. The assertion can now be proved by an easy induction. □

We denote by $\mathcal{S}$ the class of complete solvers with the following three properties: (1) they compute answer sets (or determine that no answer set exists) by generating a sequence of partial assignments so that if an answer set exists then it occurs among the generated assignments; (2) they use boolean constraint propagation to force truth assignments on unassigned atoms and trigger backtracking if contradictions are found; and (3) the generated assignments can be represented by a binary tree, whose nodes are atoms, and where the left (right) edge leaving an atoms corresponds to assigning that atom *false* (*true*). This class of solvers includes in particular solvers that use chronological backtracking, as well as those that perform backjumping (we note that in that latter case, some nodes corresponding to decision atoms may have only one child).

Proposition 2. *Let $P \in R_n^-$ be such that the maximum size of an independent set in $G(P)$ equals β. Then, the number of assignments generated by any solver in the class $\mathcal{S}$ is $O((2n)^{\beta+1})$.*

Proof: The tree representing the space of assignments generated by a solver from $\mathcal{S}$ for P has height at most n and at most $\beta+1$ left edges on every path. Indeed, if there are ever $\beta+1$ left edges on a path in the tree, then $\beta+1$ atoms are set

to false. Atoms in that set do not form an independent set in $G(P)$, and so for some two of them, say a and b, the rule $a \leftarrow not\ b$ is in P. Boolean propagation forces a or b to be true, while both of these atoms are false. Thus, a backtrack will occur (the current path will not be extended). The assertion follows now by Proposition 1, as $\binom{n}{k} \leq n^k$. □

We noted earlier that when $m \geq cn^2$, $\beta = O(\log n)$. Thus, when $m \geq cn^2$, the size of the search space is bounded by $n^{O(1)}2^{O(\log^2 n)}$, which is asymptotically much smaller than $O(2^n)$. Furthermore, with m getting closer to $n(n-1)$, β gets even smaller and so, the search space gets smaller, too.

Finally, we note (due to space limits, we do not discuss these results in detail) that adding even a small number of positive rules or constraints to programs from mR_n^- generally makes the resulting programs easier. These results suggest that from the perspective of benchmarking and insights into search heuristics, proper purely negative constraint-free programs are especially important.

6 Benchmarks for SAT Solvers

Deciding whether a logic program has an answer set is in the class NP. Thus, there are polynomial-time methods to reduce the task of computing stable models of a program to that of computing models of a CNF theory. Unfortunately, all known reductions lead to theories whose size is superlinear with respect to that of the original program [17].

However, linear-size reductions exist for programs that are *tight* [18]. Namely, answer sets of a tight program P are precisely models of the Clark's *completion* of P [8]. Purely negative programs are tight. In particular, programs in R_n^- are tight. Moreover, if $P \in R_n^-$, then the completion of P has especially simple form. It can be written as the collection of the following clauses: (1) $a \vee b$, where $a \leftarrow not\ b \in P$, and (2) $\neg a \vee \neg b_1 \vee \ldots \vee \neg b_k$, where $a \leftarrow not\ b_i$, $1 \leq i \leq k$, are *all* rules in P with a as the head.

Theories of that type obtained from programs form R_n^-, constitute an interesting class of benchmarks for SAT solvers. They are simple in that most of their clauses consist of two literals and all other clauses are disjunctions of atoms. Moreover, as the density grows, there is no phase transition, unlike in the case of the standard model. Instead, we observe the familiar easy-hard-easy property, with the hard region correlated well with the one we observed for *clasp* and *smodels* (Figure 3).

7 Discussion

We proposed and considered a model of random logic programs with fixed-length rules. We focused on the case of proper programs with two-literal rules. Our model is parameterized by the number of atoms and five integers that specify the numbers of rules of each possible type. Due to its simplicity, the model lends itself to theoretical investigations. To the best of our knowledge, our paper provides

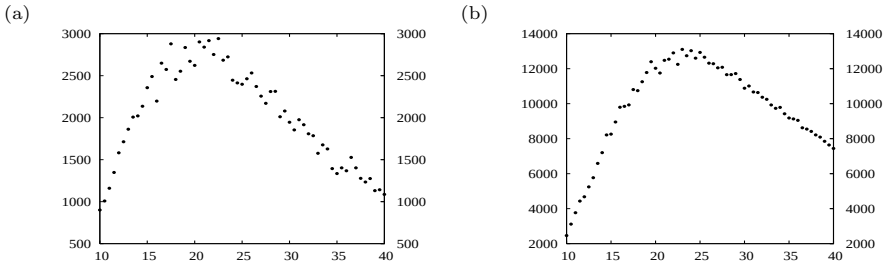

Fig. 3. Easy-hard-easy pattern shown by *minisat* [19] on the completions of programs from $mR^{-}n$, where $n = 150$, satisfiable theories in graph (a), inconsistent ones in graph (b).

first non-trivial theoretical results on the properties of random programs. Our experimental results show that while simple, the model allows us to generate relatively small programs that are hard for the current ASP solvers. Computing answer sets of proper purely negative constraint-free programs with 600 atoms generated from the hard region seems to be infeasible at present. We also noted that completions of hard programs from our model are challenging benchmarks for SAT solvers. One of the main outcomes of our paper is the emergence of proper purely negative constraint-free programs as the core class for generating benchmarks and a key to theoretical studies of properties of random programs.

The model we proposed for the case of two-literal rules can be generalized to programs consisting of $k \geq 3$ rules. We believe that most properties we identified in this paper generalize, too. In particular, our preliminary experiments show the same easy-hard-easy pattern for proper purely negative constraint-free programs with three-literal rules. Moreover, programs from the hard region are harder than hard-region two literal ones.

There are several differences between our work and that of Zhao and Lin [5]. First, we consider the fixed rule-length model (and more narrowly, only the case of two-literal programs). Second, we can specify in our model the composition of programs in terms of the numbers of rules of particular types. That facilitates studies of the effect these rules have when added to the "core" consisting of proper purely negative constraint-free programs. Third, we focus on, what we believe, is the key class of random logic programs — the class of programs that are proper purely negative and constraint-free. Despite these differences, one of specializations of our model (that allows for constraints) is quite closely related to Zhao and Lin model and shows similar properties. We also note that the first and the last of the issues discussed above differentiate our approach from an unpublished work by Wong, Schlipf and Truszczyński [20].

Finally, we note that for the class R_n^-, as well as for several other classes of programs we can define in our framework, we do not observe the phase transition. That is, unlike in SAT, increasing the density (the number of rules) does not result in a sudden transition from consistent to inconsistent programs. In fact there is no density for which programs are a.a.s inconsistent. We believe it is due to the nonmonotonicity of the semantics of answer sets. For some classes

of programs, namely those with sufficiently many constraints, a transition from consistent to inconsistent programs can be observed (Zhao and Lin's model shows such transition, too). However, the transition is relatively slow.

References

1. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distributions of SAT problems. In: Proceedings of AAAI-92, Los Altos, CA, American Association for Artificial Intelligence, Morgan Kaufmann (1992)
2. Achlioptas, D.: Random satisfiability. In Biere, A., Heule, M., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. IOS Press (2009) 245–270
3. Gent, I., Walsh, T.: Easy problems are sometimes hard. Artificial Intelligence **70** (1994) 335–345
4. Selman, B., Kautz, H., McAllester, D.: Ten challenges in propositional reasoning and search. In: Proceedings of the 15th International Joint Conference on Artificial Inteligence, IJCAI-97, Morgan Kaufmann (1997) 50–54
5. Zhao, Y., Lin, F.: Answer set programming phase transition: a study on randomly generated programs. In Palamidessi, C., ed.: Proceedings of the 19th International Conference on Logic Programming, ICLP 2003. Volume 2916 of Lecture Notes in Computer Science., Springer (2003) 239–253
6. Marek, W., Truszczyński, M.: Autoepistemic logic. Journal of the ACM **38** (1991) 588–619
7. Huang, G.S., Jia, X., Liau, C.J., You, J.H.: Two-literal logic programs and satisfiability representation of stable models: a comparison. In: In Proceedings of the 15th Canadian Conference on AI. Volume 2338 of LNCS., Springer (2002) 119–131
8. Clark, K.: Negation as failure. In Gallaire, H., Minker, J., eds.: Logic and data bases. Plenum Press, New York-London (1978) 293–322
9. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234
10. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: *Clasp* : A conflict-driven answer set solver. In Baral, C., Brewka, G., Schlipf, J., eds.: Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Proceedings. Volume 4483 of LNCS., Springer (2007) 260–265
11. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7(3)** (2006) 499–562
12. Lin, F., Zhao, Y.: ASSAT: Computing answer sets of a logic program by SAT solvers. In: Proceedings of the 18th National Conference on Artificial Intelligence (AAAI 2002), AAAI Press (2002) 112–117
13. Lierler, Y., Maratea, M.: Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In: Proceedings of LPNMR-04. LNCS (2004) 346–350
14. Bollobás, B.: Random Graphs. Academic Press (1985)
15. Janson, S., Luczak, T., Ruciński, A.: Random Graphs. Wiley-Interscience (2000)
16. de la Vega, W.F.: Kernels in random graphs. Discrete Math. **82** (1990) 213–217
17. Janhunen, T.: Representing normal programs with clauses. In de Mántaras, R.L., Saitta, L., eds.: Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, IOS Press (2004) 358–362
18. Fages, F.: Consistency of Clark's completion and existence of stable models. Journal of Methods of Logic in Computer Science **1** (1994) 51–60

19. Eén, N., Sörensson, N.: An extensible SAT solver. In: Theory and Applications of Satisfiability Testing, 6th International Conference, SAT-2003. Volume 2919 of LNCS., Springer (2003) 502–518
20. Wong, D., Schlipf, J., Truszczyński, M.: On the distribution of programs with stable models. Unpublished, presented at the Dagstuhl Seminar 05171, Nonmonotonic Reasoning, Answer Set Programming and Constraints (2005)