A generator of hard 2QBF formulas and ASP programs

Giovanni Amendola and Francesco Ricca
University of Calabria, Italy

Mirek Truszczynski
University of Kentucky, USA

Introduction

Models of random instances of search problems

- Much attention in AI in the last twenty years
  - SAT [Gent and Walsh, 1994; Mitchell et al., 1992;Selman et al., 1996]
  - QBF [Gent and Walsh, 1999; Chen, Interian, 2005]
  - CP [Mitchell, 2002]
  - ASP [Zhao and Lin, 2003; Namasivayam and T, 2009]
- Intriguing phase transition phenomenon
  - Sharp transition from SAT to UNSAT
  - “Easy-hard-easy” pattern [Mitchell et al., 1992]

Introduction

Applications of Random Formula/Program Generators

- Solvers Performance Assessment
  - $\rightarrow$ Used to improve CDCL implementations [Silva et al., 2009]
  - $\rightarrow$ For testing efficacy of heuristics [Elffers et al.,2016, Järvisalo et al.,2012]
  - $\rightarrow$ In solver competitions
- Solver correctness testing [Brummayer et al., 2010]
  - $\rightarrow$ Fuzz testing for solver implementation, and defect testing in design

Introduction

Recent models of 2QBFs and ASP programs

Amendola, Ricca, and T, 2017

- Multi-component model
- Controlled model
- Combinations of the two

Features of the models

- Non-normal form boolean formulas
- Natural representations as *disjunctive* ASP programs
- Phase transition and easy-hard-easy pattern
- Instances better solved by “industrial“ SAT solvers

Introduction

Contributions:

- A generator for the models of random formulas/programs
  - → CNF formulas from the well-known fixed-length model [Mitchellet al., 2002]
  - → QBFs from the Chen-Interian model [Chen and Interian 2005]
  - → Multi-component and Controlled model formulas [Amendola et al., 2017]
  - → Supports standard output formats for SAT, QBF and ASP
  - → Implemented in Java: portable and easy to extend
- A methodology for generating instances
  - → Set the desired level of hardness
  - → Set the desired level of frequency of satisfiability

The fixed-length clause model for SAT

Random k-CNF Model

- $C(k, n, m)$: The set of all k-CNF formulas with m clauses over (some fixed) set of n propositional variables
- Select one uniformly at random
- *Select m k-literal clauses over a set of n variables uniformly, independently and with replacement*

The fixed-length clause models for QBF

The Chen-Interian Model

- Let X and Y be sets of variables s.t. $X \cap Y = \emptyset$, and $A = |X|$ and $E = |Y|$
- $C(a, e; A, E; m)$: all $(a + e)$-CNF formulas with m clauses, each with a literals over X and e literals over Y
- $Q(a, e; A, E; m)$: all QBFs $\forall X \exists Y F$, where $F \in C(a, e; A, E; m)$
- *Generate QBFs from $Q(a, e; A, E; m)$, by generating clauses from $C(a, e; A, E; m)$ uniformly, independently and with replacement*

The Controlled model

- $Q^{ctd}(k, A, E)$
- The matrix consists of pairs of clauses $x \vee C$, $-x \vee C$

  $\rightarrow$ One pair for each universal variable x

  $\rightarrow$ C — a random $(k-1)$-clause over existential veriables
- $Q^{ctd}(k, A, E) \subseteq Q(1, k-1; A, E; 2A)$

The multi-component models: SAT & QBF

Multi-component model of propositional formulas

Let $\mathcal{F}$ be a class (or random model) of formulas

- t-$\mathcal{F}$: the class of all disjunctions of t formulas from $\mathcal{F}$
- t-$\mathcal{Q}$: the class of all QBFs $\forall X \exists Y F$, where $F \in t$-$\mathcal{F}$

Example (SAT)

Classical. An instance of $C(2, 3, 2)$ is

$$(a \vee b) \wedge (a \vee -c)$$

i.e., $C(2, 3, 2)$ is the class of 2-CNFs of 2 clauses with 3 vars!

Multi-component. An instance $3\text{-}C(2, 3, 2)$ is

$$\underbrace{((a \vee b) \wedge (a \vee -c))}_{\text{2CNF component 1}} \vee \underbrace{((c \vee a) \wedge (-a \vee -c))}_{\text{2CNF component 2}} \vee \underbrace{((-c \vee -a) \wedge (-b \vee c))}_{\text{2CNF component 3}}$$

The multi-component models: SAT & QBF

- Phase transition shows up again
- With the same values for its low and high boundaries as in the single-component model

The multi-component models: ASP

From formulas to programs

- Our results on QBFs naturally imply a model of random disjunctive logic programs
- Adapting the Eiter-Gottlob reduction of disjunctive logic programming in QBF [Eiter and Gottlob, 1995]
- Based on *conjunctions* of t DNF formulas

  $\rightarrow D(e, a; E, A; m)$ that are *dual* to $C(e, a; E, A; m)$
- The encoding is natural and simple

  $\rightarrow$ Much more compact than Tseitin transformation needed for formulas!

Command line and example

```
$ java -jar RandomGenerator.jar -h

SYNOPSIS: MainGenerator [-option]
-generator=[BasicGenerator,CIGenerator,SATGenerator,
            ControlledCIGenerator] Select generator type
-out=[PrintProgram,PrintQBF,PrintQCIR,MultiOutput,
       PrintSAT] Select output format
-o =<filename> Specify filename, mandatory with
              MultiOutput Generator, default STDOUT
-formats=<OutputFormat1, ..., OutputFormatn>
       Specify a comma-separated list of output formats for
       MultiOutput, e.g., PrintProgram,PrintQBF
-E=<n> Number of existential variables, default 1
-A=<n> Number of universal variables, default 1
-c=<n> Number of clauses/rules,
       ignored by ControlledCIGenerator, default 1
-k=<n> Clause/rule size, only for BasicGenerator, default 1
-e=<n> Number of existentials in each clause/rule
       only for CI, default 1
-a=<n> Number of universals in each clause/rule
       only for CI, default 1
-w=<n> Number of components, default 1
```

Command line and example

```
$ java -jar RandomGenerator.jar -generator=CIGenerator
      -out=PrintQBF -o=10-CI-2-3-20-40-80 -w=10 -a=2 -e=3
      -A=20 -E=40 -c=80

$ java -jar RandomGenerator.jar
      -generator=ControlledCIGenerator
      -out=MultiOutputGenerator
      -format=PrintProgram,PrintQBF,PrintQCIR
      -o=4-Qctd-4-20-10 -w=10 -a=1 -e=3 -A=20 -E=10
```

Generating formulas

Observation

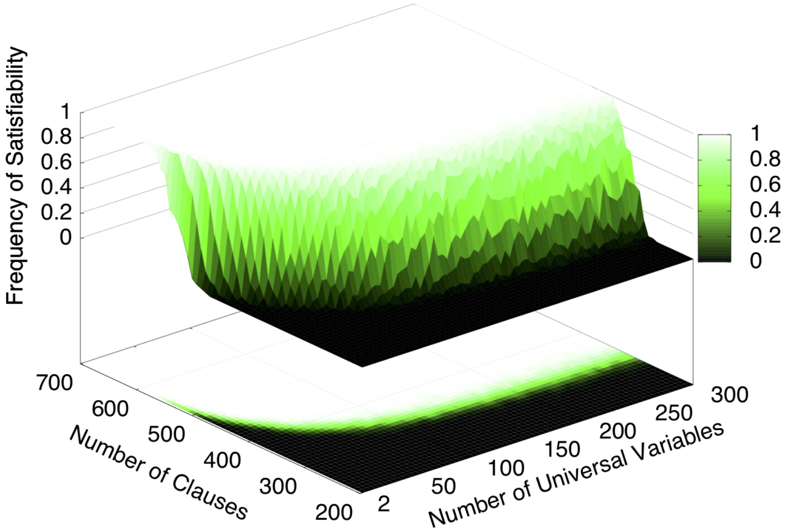
- Different goals $\rightarrow$ different parameters
- Not an obvious choice

Key underlying property

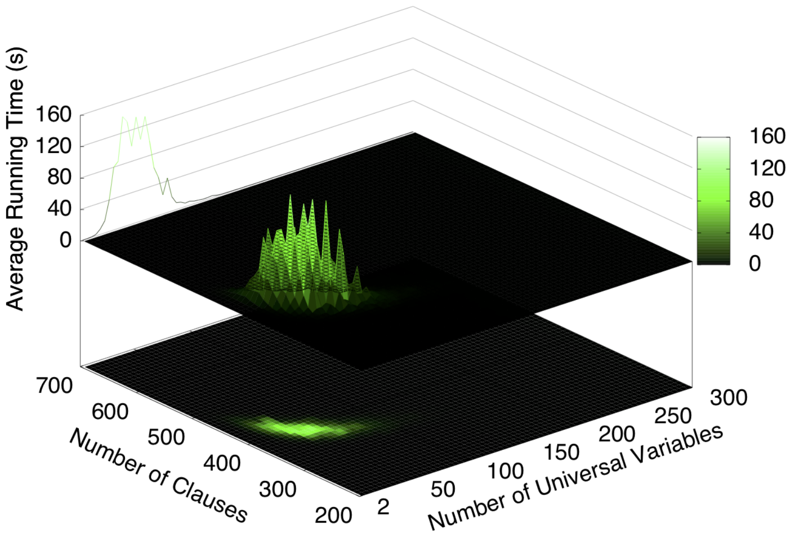
- The location of the phase transition

  $\rightarrow$ To select instances of the desired "satisfiability"
- Solver-independent

Phase transition and hardness



Phase transition and hardness



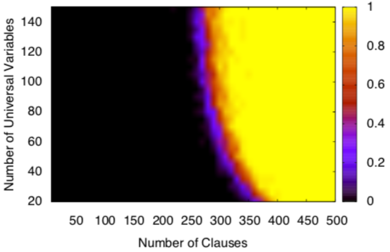
Guidelines

Multi-component Chen-Interian

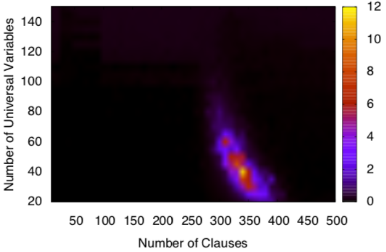
- Fix *a* and *e* to define the structure of a clause
- Run the tool for each pair of values of *A* and *E* with different numbers *m* of clauses/rules
- Identify phase transition
- Select the value of *m* that yields the desired difficulty
- Eventually increase *t* to get super-hard instances

...and similarly for the other models

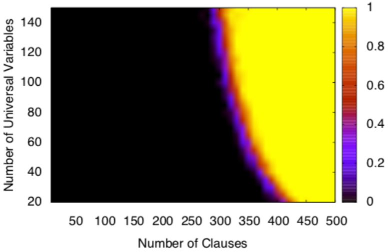
Guidelines



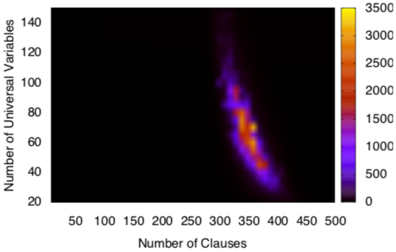
(b) Frequency of SAT for 1-$Q(1,3,A,60,m)$



(b') Execution Time (s) for 1-$Q(1,3,A,60,m)$



(c) Frequency of SAT for 2-$Q(1,3,A,60,m)$



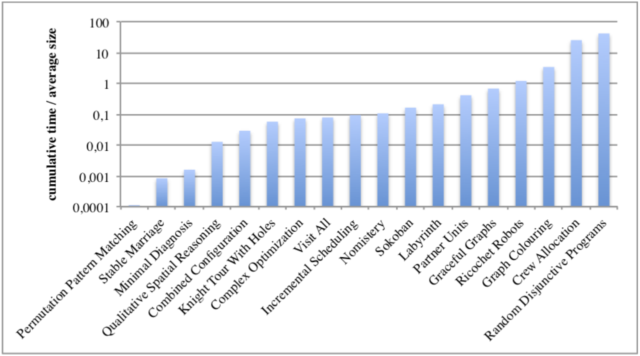
(c') Execution Time (s) for 2-$Q(1,3,A,60,m)$

Figure 3: Phase transition and hardness in (multicomponent) Chen-Interian formulas.

Usecases

ASP Competition 2017

- The smallest in size but among the hardest to solve
- No solver could solve all these instances (of <100 vars!)



Usecases

ASP Competition 2017

- The smallest in size but among the hardest to solve
- No solver could solve all these instances (of <100 vars!)

QBF EVal 2016-2017-2018

- Among the hardest instances of 2QBF

  $\rightarrow$ less than 100 vars, max 9 components, >76% tagged as hard!!
- Used in the Hard Instances Track in 2018
- Helped identify buggy participants

Conclusion

A new generator for hard 2QBF and ASP programs

- Based on Multi-component and Controlled models [Amendola, Ricca and T, 2017]

  → The first models for *disjunctive* ASP programs

- Useful for development and testing of practical solvers

  → Supports standard formats (ASPCore 2, QCIR, (Q)DIMACS)

  → Used in ASP and QBF competitions

- Implemented in Java and available on the Web:

  `www.mat.unical.it/ricca/RandomLogicProgramGenerator`

Thanks for your attention!