— Tutorial —

Backdoors to Satisfaction

Barry O'Sullivan

b.osullivan@cs.ucc.ie

Cork Constraint Computation Centre
University College Cork, Ireland

http://osullivan.ucc.ie

CP 2010

ucc

Coláiste na hOllscoile Corcaigh, Éire
University College Cork, Ireland

— Tutorial —

Back"⋆" Galactica

Barry O'Sullivan

b.osullivan@cs.ucc.ie

Cork Constraint Computation Centre
University College Cork, Ireland

http://osullivan.ucc.ie

CP 2010

ucc
Coláiste na hOllscoile Corcaigh, Éire
University College Cork, Ireland

Supported by:
Science Foundation Ireland Grant 05/IN/I886.

Main Technical Sections of the Tutorial

1. Backdoors – [Williams et al., IJCAI-03] and [Kilby et al., AAAI-05]
   - Polynomial-time Sub-Solvers
   - Weak and Strong Backdoors
   - Backdoors in Realworld Problems
   - Computing Weak and Strong Backdoors
2. Exploiting Backdoors – [Williams et al., IJCAI-03]
   - Deterministic Strategy
   - Randomized strategy
   - Heuristic strategy
3. Backdoors and Problem Hardness – [Ruan et al., AAAI-04]
   - Backdoor Size and Hardness
   - Backdoor Keys and Key Fraction
4. Some Directions of Study
   - Backdoors and Parameterised Complexity
   - Backdoors and Machine Learning

About this Tutorial

Motivation

Understanding the importance of structure in solving real world constraint satisfaction problems is both interesting and important. The discovery and exploitation of sets of backdoor variables is of particular interest.

About this Tutorial

Motivation

Understanding the importance of structure in solving real world constraint satisfaction problems is both interesting and important. The discovery and exploitation of sets of backdoor variables is of particular interest.

The Promise

This tutorial will give you an introduction to the key concepts in relation to backdoor variables and some interesting directions for future work.

About this Tutorial

Motivation

Understanding the importance of structure in solving real world constraint satisfaction problems is both interesting and important. The discovery and exploitation of sets of backdoor variables is of particular interest.

The Promise

This tutorial will give you an introduction to the key concepts in relation to backdoor variables and some interesting directions for future work.

Scope

I will focus on a small number of key papers by Williams et al. (IJCAI 2003), Ruan et al. (AAAI 2004), and Kilby et al. (AAAI 2005), along with some of my own work.

Predictive versus Descriptive - My Terminology

Predictive Backdoors

Explain precisely the structural conditions under which a problem can become tractable, e.g. tree-structured CSPs (Freuder), and cycle cutsets (Dechter).

Predictive versus Descriptive - My Terminology

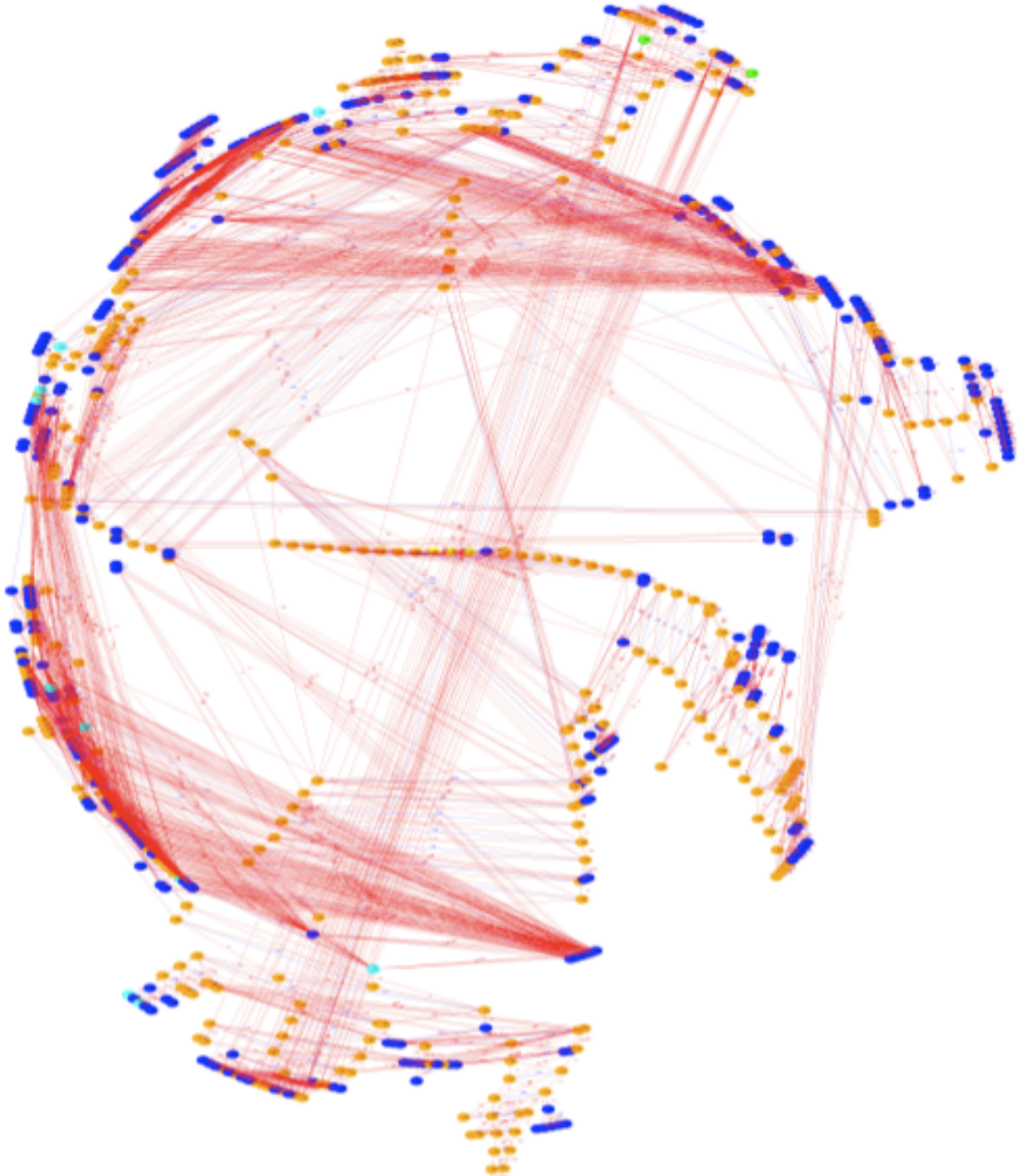
Predictive Backdoors

Explain precisely the structural conditions under which a problem can become tractable, e.g. tree-structured CSPs (Freuder), and cycle cutsets (Dechter).

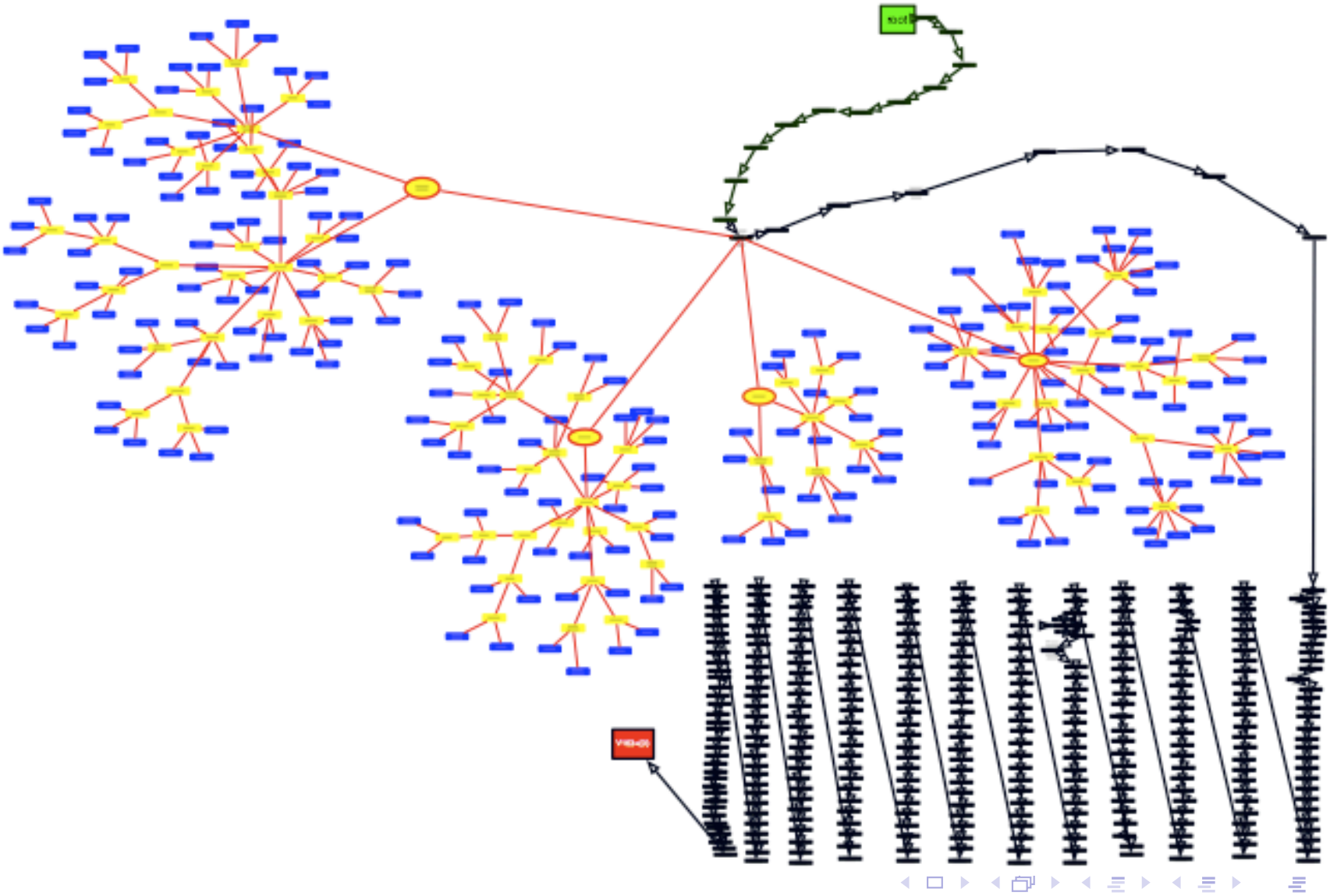
Descriptive Backdoors

Explain why we observe short runs in practice when we are not (necessarily) in a tractable class.

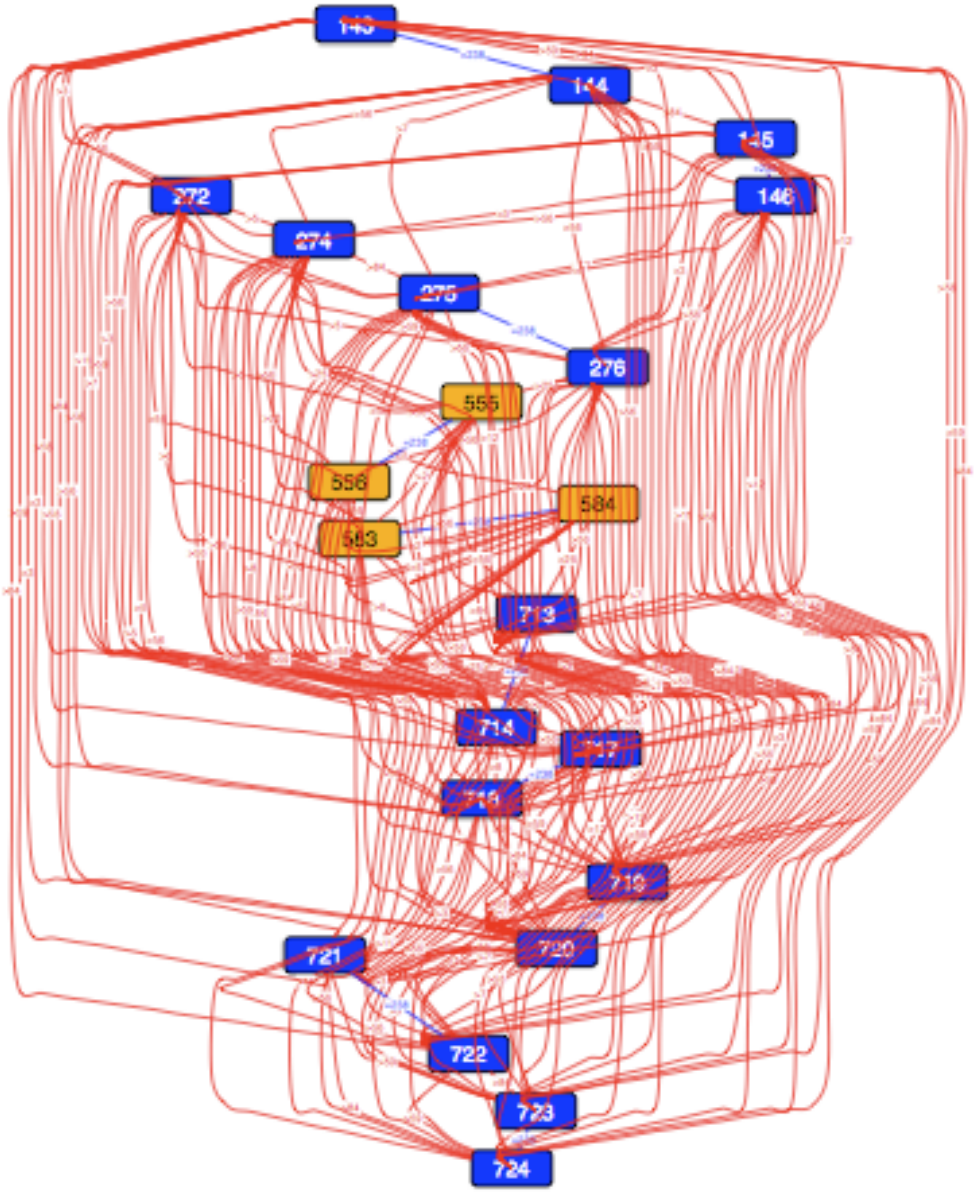
A Real-world Problem: RLFAP-11



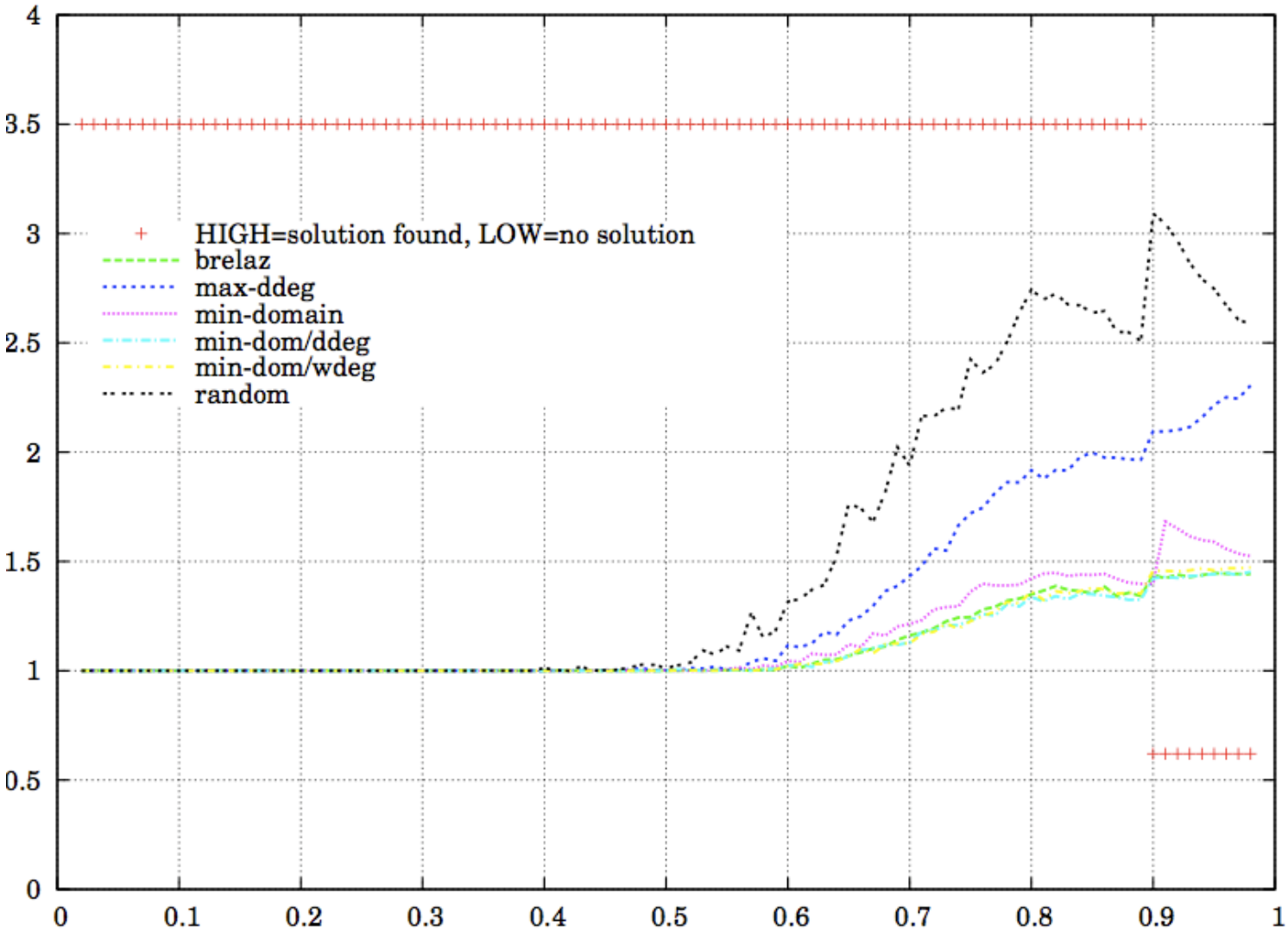
MAC Search Tree for RLFAP-11



Why is RLFAP-11 hard for some solvers?

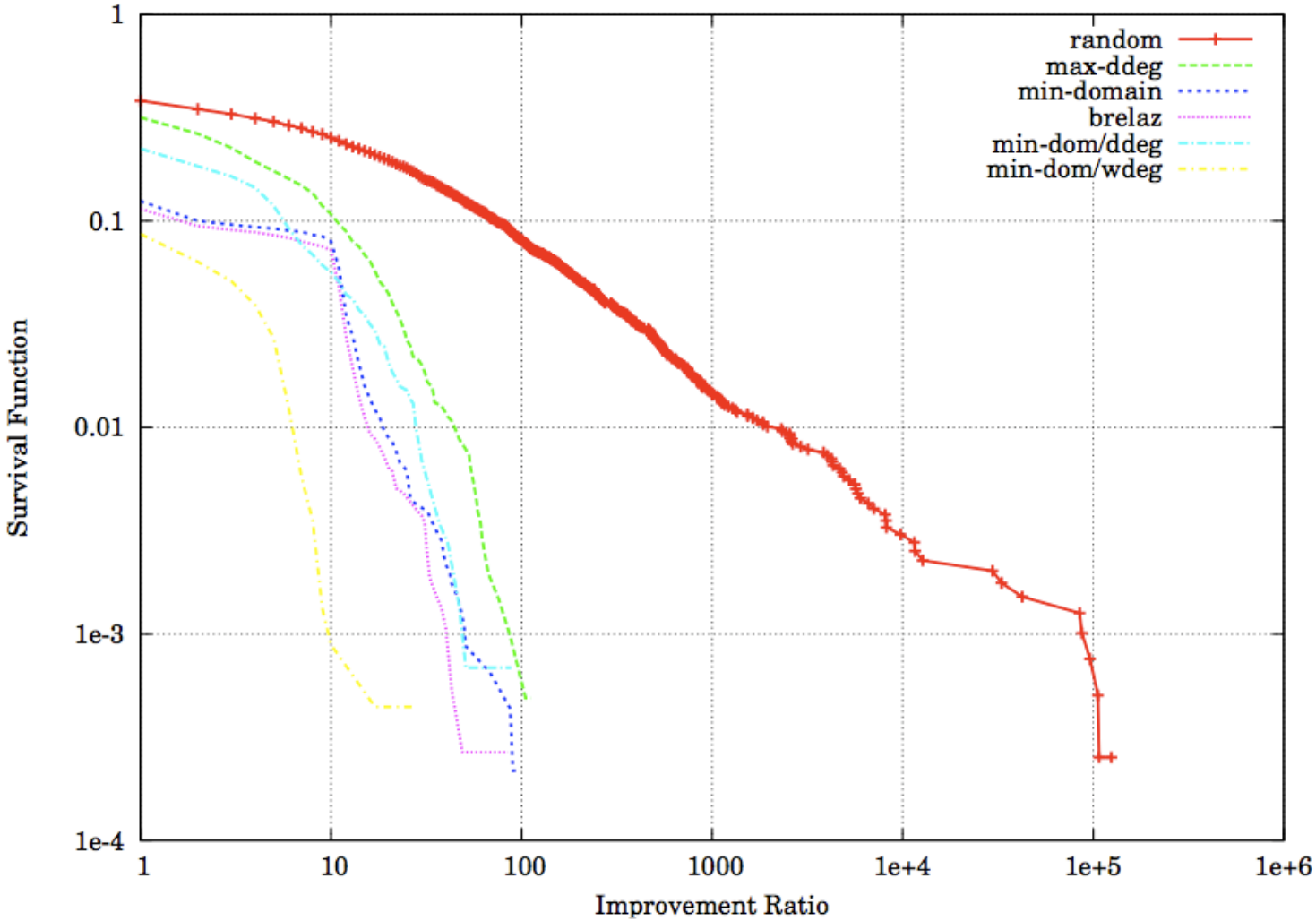


What about unsatisfiable problems?



(Optimal) Refutations in uniform random problems.

What about unsatisfiable problems?



(Optimal) Refutations in real-world problems.

Erratic Sample Means in Backtrack Search

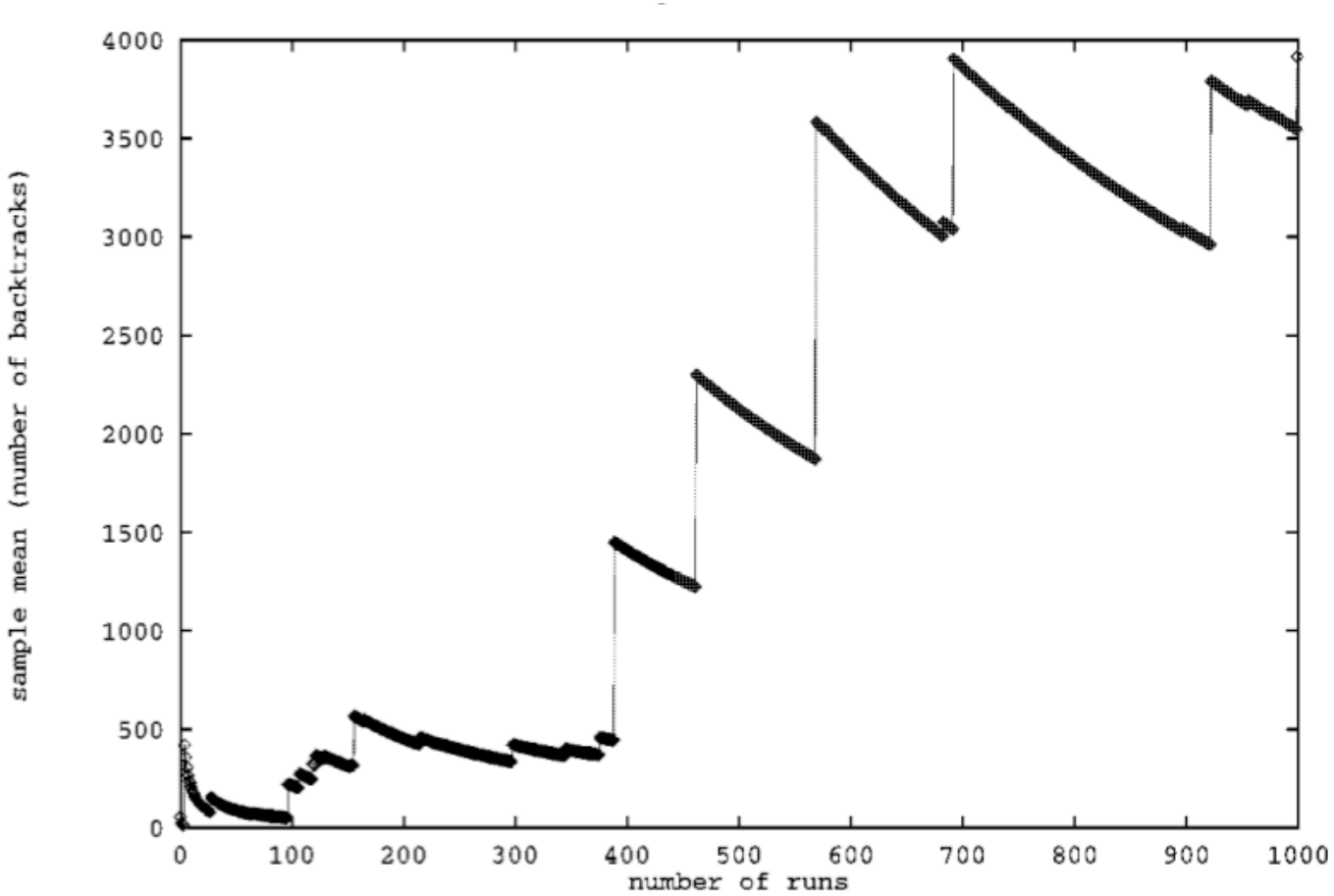


Figure: Gomes et al. discovered heavy-tailed runtime distributions [JAR, '01].

Key Papers by Others

[Williams et al., IJCAI-03]

Ryan Williams, Carla P. Gomes, Bart Selman: Backdoors To Typical Case Complexity. IJCAI 2003: 1173-1178.

[Ruan et al., AAAI-04]

Yongshao Ruan, Henry A. Kautz, Eric Horvitz: The Backdoor Key: A Path to Understanding Problem Hardness. AAAI 2004: 124-130.

[Kilby et al., AAAI-05]

Philip Kilby, John K. Slaney, Sylvie Thiébaux, Toby Walsh: Backbones and Backdoors in Satisfiability. AAAI 2005: 1368-1373.

Outline

1. Backdoors – [Williams et al., IJCAI-03] and [Kilby et al., AAAI-05]
   - Polynomial-time Sub-Solvers
   - Weak and Strong Backdoors
   - Backdoors in Realworld Problems
   - Computing Weak and Strong Backdoors
2. Exploiting Backdoors – [Williams et al., IJCAI-03]
   - Deterministic Strategy
   - Randomized strategy
   - Heuristic strategy
3. Backdoors and Problem Hardness – [Ruan et al., AAAI-04]
   - Backdoor Size and Hardness
   - Backdoor Keys and Key Fraction
4. Some Directions of Study
   - Backdoors and Parameterised Complexity
   - Backdoors and Machine Learning

The Constraint Satisfaction Problem

- A CSP, C, is characterized by a set $V = \{x_1, \ldots, x_n\}$ of variables, with domains $D_1, \ldots, D_n$, respectively, which list the possible values for each variable.
- A constraint is defined over a subset of the variables $S_i \subseteq V$, called its scope.
- The set of tuples satisfying the constraint S_i is a relation defined as a subset of the Cartesian product of the domains of the variables in its scope.
- We assume variables have the same domain, and say that $d = |D|$.
- An assignment a is a function from variables to D. A partial assignment assigns a subset of V.
- A solution is an assignment to all variables that satisfies all the constraints.

Simplifying a CSP

- $C[x \mapsto v]$ denotes the simplified CSP we get by setting variable x to value v.
- Let $a_S :: (S \subseteq V) \mapsto D$ be a partial assignment. We use the notation $C[a_S]$ to denote the simplified CSP we get when making each assignment in a_S.
- We have said little about how the simplification is performed at this point – we'll come back to this shortly.

The Method of Simplification: A Polynomial Sub-Solver

Sub-solver

A sub-solver A given as input a CSP C satisfies the following:

- **Trichotomy:** A either rejects the input C, or "decides" C correctly as either satisfiable or unsatisfiable.
- **Efficiency:** In runs in polynomial time
- **Trivial Solvability:** A can determine if C is trivially true or trivially false.
- **Self-reducibility:** If A determines C, then for any variable x and value v, then A determines $C[x \mapsto v]$.

What is a Backdoor?

Definition (Weak Backdoor – for satisfiable instances!)

A non-empty subset S of the variables is a weak backdoor in C with respect to sub-solver A if there exists an $a_S :: (S \subseteq V) \mapsto D$, A returns a satisfying assignment of $C[a_S]$.

What is a Backdoor?

Definition (Weak Backdoor – for satisfiable instances!)

A non-empty subset S of the variables is a weak backdoor in C with respect to sub-solver A if there exists an $a_S :: (S \subseteq V) \mapsto D$, A returns a satisfying assignment of $C[a_S]$.

Intuition

A (weak) backdoor is a set of variables that *when assigned correctly* simply the problem to an extent that the sub-solver than solve the remaining problem.

What is a Backdoor?

Definition (Weak Backdoor – for satisfiable instances!)

A non-empty subset S of the variables is a weak backdoor in C with respect to sub-solver A if there exists an $a_S :: (S \subseteq V) \mapsto D$, A returns a satisfying assignment of $C[a_S]$.

Intuition

A (weak) backdoor is a set of variables that *when assigned correctly* simply the problem to an extent that the sub-solver than solve the remaining problem.

Example

For binary CSP, if A is an arc-consistency propagator, a cycle cutset defines a backdoor.

What is a Strong Backdoor?

Definition (Strong Backdoor – for (un)satisfiable instances)

A non-empty subset S of the variables is a strong backdoor in C with respect to sub-solver A if for all $a_S :: (S \subseteq V) \mapsto D$, A returns a satisfying assignment or decides the unsatisfiability of $C[a_S]$.

What is a Strong Backdoor?

Definition (Strong Backdoor – for (un)satisfiable instances)

A non-empty subset S of the variables is a strong backdoor in C with respect to sub-solver A if for all $a_S :: (S \subseteq V) \mapsto D$, A returns a satisfying assignment or decides the unsatisfiability of $C[a_S]$.

Example

For an unsatisfiable problem the variables in an unsatisfiable core defines a strong backdoor.

Backbones & their relationship to (Strong) Backdoors

Backbone — it gets a little like "Back$\star$ Gallatica"

S is a backbone of C if there is a unique partial assignment $a_S :: (S \subseteq V) \mapsto D$ such that $C[a_S]$ is satisfiable.

Backbones & their relationship to (Strong) Backdoors

Backbone — it gets a little like "Back⋆ Gallatica"

S is a backbone of C if there is a unique partial assignment $a_S :: (S \subseteq V) \mapsto D$ such that $C[a_S]$ is satisfiable.

Comparison

Backbone variables are frozen in all solutions: if there are no solutions, the backbone is empty; if there is one solution, that solution defines the backdoor.

A (strong) backdoor is sufficient to decide the problem instance.

Independent Variables define a Backdoor

Necessarily a Backdoor

Problems are often modelled such that a subset of the variables *determine* the remainder. We refer to these

Backdoors in Realworld Problems

instance	# vars	# clauses	backdoor	fract.
logistics.d	6783	437431	12	0.0018
3bitadd_32	8704	32316	53	0.0061
pipe_01	7736	26087	23	0.0030
qg_30_1	1235	8523	14	0.0113
qg_35_1	1597	10658	15	0.0094

Figure: Examples of small backdoors in real-world instances [Williams et al., IJCAI-03]

Computing Weak Backdoors

Algorithm 1 MinWeakBackdoor (F, I)

Input: Formula F, Initial weak backdoor set I - i.e. running *satz* on $F \cup I$ requires no branching.

Output: A set of literals W forming a minimal backdoor, and a model M consistent with the backdoor

1. $W \leftarrow \emptyset$; $M \leftarrow \emptyset$
2. **while** $I \neq \emptyset$
3. Choose literal $l \in I$ randomly
4. $I \leftarrow I \setminus \{l\}$
5. Run *satz* on $F \cup W \cup I$
6. **if** *satz* requires branching,
7. **then** $W \leftarrow W \cup \{l\}$; $M \leftarrow$ *satz* solution
8. **endwhile**
9. **return** W, M

[Kilby et al., AAAI-05]

Computing Weak Backdoors

Algorithm 2 SatzWeak (F)

Input: Formula F

Output: A minimal weak backdoor W and a consistent model M

1. Solve F using *satz* , saving branching literals in B
2. $W, M \leftarrow$ MINWEAKBACKDOOR (F, B)
3. **return** W, M

[Kilby et al., AAAI-05]

Computing Strong Backdoors

Algorithm 5 StrongBackdoor (F, Max-card)

Input: Formula F, the maximum cardinality Max-card.

Output: A set of strong backdoors $\mathcal{S}$, and a set of weak backdoors $\mathcal{W}$.

1. $\mathcal{S} \leftarrow \mathcal{W} \leftarrow \emptyset$
2. **for each** subset X of the problem variables of size up to Max-card
3. **for each** distinct set of literals L corresponding to the variables in X
4. Run *satz* on $F \cup L$
5. **if** branching is not required,
   and the formula is satisfiable
6. **then** $\mathcal{W} \leftarrow \mathcal{W} \cup L$
7. **if** no literal set required branching
8. **then** $\mathcal{S} \leftarrow \mathcal{S} \cup X$
9. **return** $\mathcal{S}, \mathcal{W}$

[Kilby et al., AAAI-05]

Outline

1. Backdoors – [Williams et al., IJCAI-03] and [Kilby et al., AAAI-05]
   - Polynomial-time Sub-Solvers
   - Weak and Strong Backdoors
   - Backdoors in Realworld Problems
   - Computing Weak and Strong Backdoors
2. Exploiting Backdoors – [Williams et al., IJCAI-03]
   - Deterministic Strategy
   - Randomized strategy
   - Heuristic strategy
3. Backdoors and Problem Hardness – [Ruan et al., AAAI-04]
   - Backdoor Size and Hardness
   - Backdoor Keys and Key Fraction
4. Some Directions of Study
   - Backdoors and Parameterised Complexity
   - Backdoors and Machine Learning

Can exploiting backdoors help improve search?

The Key Question

Can it pay off to search and exploit backdoors? We, of course, don't assume we know the backdoor ahead of time.

Can exploiting backdoors help improve search?

The Key Question

Can it pay off to search and exploit backdoors? We, of course, don't assume we know the backdoor ahead of time.

Yes!

We'll discuss three strategies...

1. Deterministic strategy
2. Randomized strategy
3. Heuristic strategy

Deterministic Strategy

When does this strategy work?

For any CSP where the instance has a small fraction of backdoor variables with respect to the sub-solver, e.g. a backdoor of constant size.

Deterministic Strategy

When does this strategy work?

For any CSP where the instance has a small fraction of backdoor variables with respect to the sub-solver, e.g. a backdoor of constant size.

A Deterministic Algorithm

Given a CSP C with n variables:

For $i = 1, \ldots, n$, for all subsets S of the variables with $|S| = i$, perform a backtrack search over the variables in S for an assignment that results in C being solved by sub-solver A.

Deterministic Strategy

When does this strategy work?

For any CSP where the instance has a small fraction of backdoor variables with respect to the sub-solver, e.g. a backdoor of constant size.

A Deterministic Algorithm

Given a CSP C with n variables:

For $i = 1, \ldots, n$, for all subsets S of the variables with $|S| = i$, perform a backtrack search over the variables in S for an assignment that results in C being solved by sub-solver A.

For a constant sized backdoor this algorithm runs in polynomial time!

Deterministic Strategy – The Complexity Result

Theorem (Running Time of the Deterministic Algorithm)

If C has a variable domain size d and a backdoor of at most $B(n)$, then the deterministic algorithm runs in:

$$\mathcal{O}(p(n) \cdot (\frac{d \cdot n}{\sqrt{B(n)}})^{B(n)}),$$

for some polynomial $p(n)$.

Deterministic Strategy – The Complexity Result

Proof.

Given a sub-solver running in $T(n)$ the running time is bounded by:

$$\begin{aligned} T(n) \cdot \sum_{i=1}^{B(n)} \binom{n}{i} d^i &\leq T(n) \cdot q(n) \cdot \binom{n}{B(n)} \cdot d^{B(n)} \\ &\leq T(n) \cdot q(n) \cdot \frac{(d \cdot n)^{B(n)}}{B(n)!} \\ &\leq p(n) \cdot \frac{(d \cdot n)^{B(n)}}{B(n)^{B(n)/2}} \\ &= p(n) \cdot (\frac{d \cdot n}{\sqrt{B(n)}})^{B(n)} \end{aligned}$$

for $p(n) = T(n) \cdot q(n)$. □

Deterministic Strategy – A Special Case

Corollary

If C has a backdoor of size $B(n) = \mathcal{O}(\log\ n)$*, then C can be solved in*

$$\left(\frac{n}{\sqrt{\log\ n}}\right)^{\mathcal{O}(\log\ n)}$$

time – representing a huge saving in effort.

Randomized strategy

The Idea

Repeatedly choose random subsets of variables that are larger than $B(n)$, an upper bound on backdoor size, searching these subsets for a backdoor.

Randomized strategy

The Idea

Repeatedly choose random subsets of variables that are larger than $B(n)$, an upper bound on backdoor size, searching these subsets for a backdoor.

Algorithm

Given a CSP C over n variables, repeat the following

$$max(n \cdot (\frac{n}{(b-1) \cdot B(n-1)})^{B(n)}, 1)$$

times:

Randomly choose a subset S of the n variables of size $b \cdot B(n)$. Perform a backtrack search search over the variables in S. Return a satisfying assignment if found.

Randomized strategy

Where does $B(n)$ come from?

Guess by using a progression $1, \alpha, \alpha^2, \ldots,$ until we decide C.

Randomized strategy

Where does $B(n)$ come from?

Guess by using a progression $1, \alpha, \alpha^2, \ldots,$ until we decide C.

Property of the Algorithm

If C has a backdoor of size $B(n)$, the randomized algorithm finds a solution with a probability approaching 1.

Randomized strategy

Why repeat $max(n \cdot (\frac{n}{(b-1)\cdot B(n-1)})^{B(n)}, 1)$ times?

Probablity a random S contains the entire backdoor of size $n > j \geq B(n)$ is at least:

$$\begin{aligned} \binom{n-B(n)}{j-B(n)} / \binom{n}{j} &\geq (\frac{(j-B(n))}{(n-B(n)))})^{B(n)} \\ &= (\frac{(b-1)}{(b-1)\cdot(B(n)-1))})^{B(n)} \end{aligned}$$

setting $j = b \cdot B(n)$. Therefore, repeating n times, the algorithm succeeds with probability at least

$$1 - \frac{1}{n}.$$

Heuristic strategy

Setting

A DFS algorithm using a heuristic H to select variables. H has a probability of $1/h$ of selecting a backdoor variable. A polynomial time restart strategy can exist.

Theorem (Polynomial Restart Strategy)

If the size of a backdoor of a CSP C is

$$B \leq \frac{c \log n}{\log h + \log d}$$

for some constant c, then (DFS, H, A) has a restart strategy that solves C in polynomial time.

Summary: Time bounds for CSPs with Backdoors

$B(n)$	deterministic	randomized	heuristic
n/k	small $exp(n)$	smaller $exp(n)$	tiny $exp(n)$
$O(\log n)$	$\left(\frac{n}{\sqrt{\log n}}\right)^{O(\log n)}$	$\left(\frac{n}{\log n}\right)^{O(\log n)}$	$poly(n)$
$O(1)$	$poly(n)$	$poly(n)$	$poly(n)$

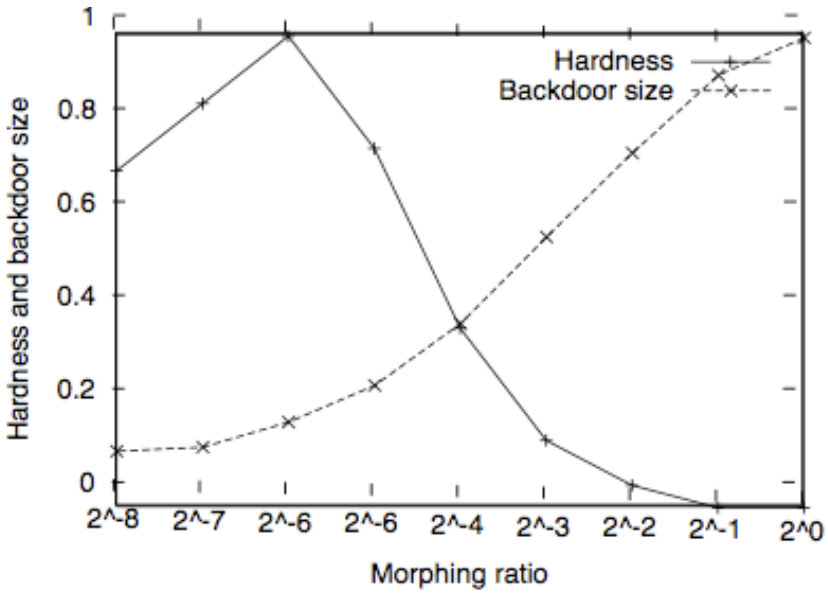
[Williams et al., IJCAI-03]

Outline

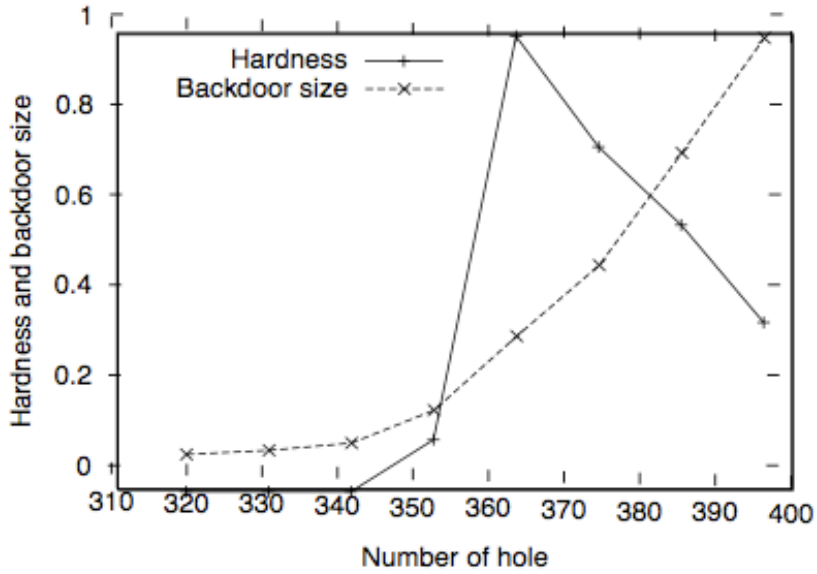
1. Backdoors – [Williams et al., IJCAI-03] and [Kilby et al., AAAI-05]
   - Polynomial-time Sub-Solvers
   - Weak and Strong Backdoors
   - Backdoors in Realworld Problems
   - Computing Weak and Strong Backdoors
2. Exploiting Backdoors – [Williams et al., IJCAI-03]
   - Deterministic Strategy
   - Randomized strategy
   - Heuristic strategy
3. Backdoors and Problem Hardness – [Ruan et al., AAAI-04]
   - Backdoor Size and Hardness
   - Backdoor Keys and Key Fraction
4. Some Directions of Study
   - Backdoors and Parameterised Complexity
   - Backdoors and Machine Learning

Hardness versus Backdoor Size – Ensemble Level

Comparison using two benchmark suites: Morphed Graph Colouring and Quasigroups with Holes. [Ruan et al., AAAI-04]



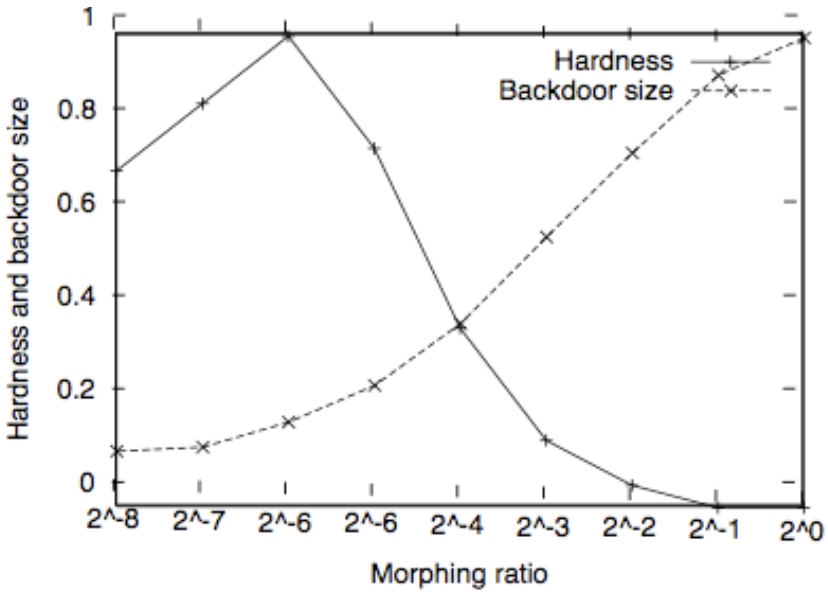
(a) Morphed GC



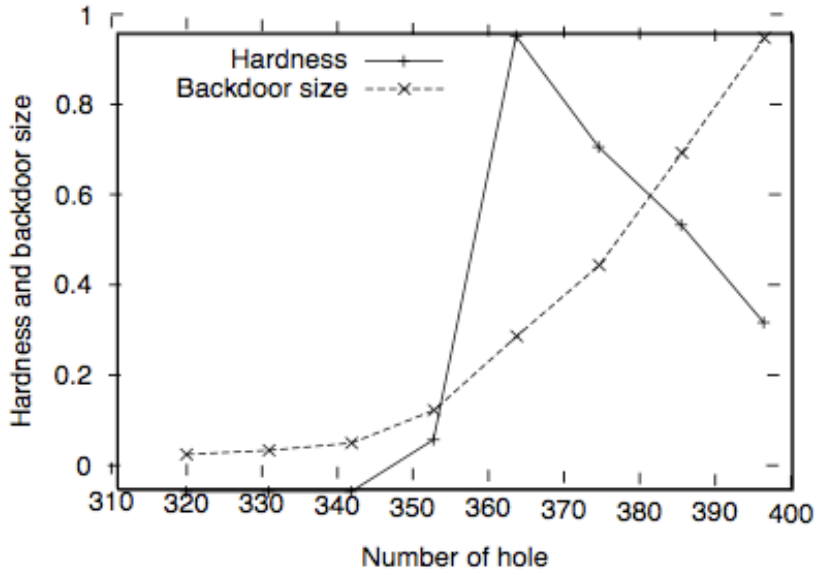
(b) QWH

Hardness versus Backdoor Size – Ensemble Level

Comparison using two benchmark suites: Morphed Graph Colouring and Quasigroups with Holes. [Ruan et al., AAAI-04]



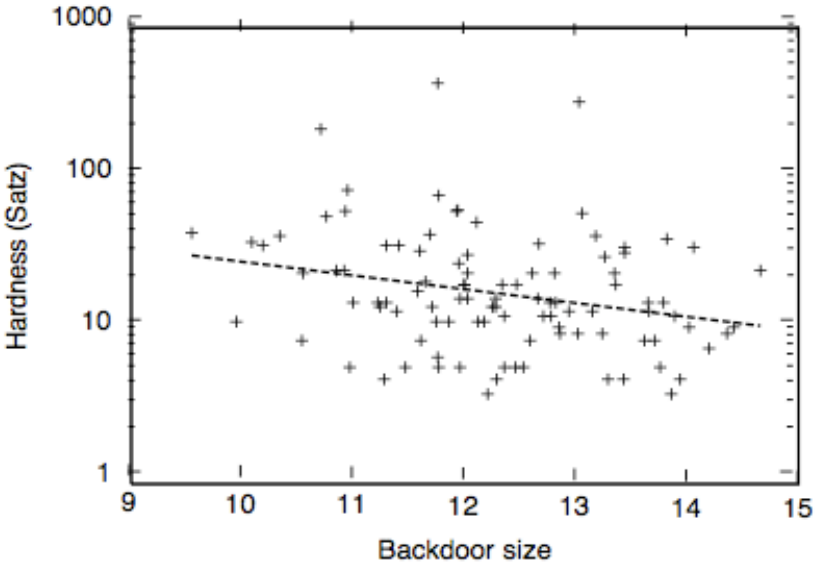
(c) Morphed GC



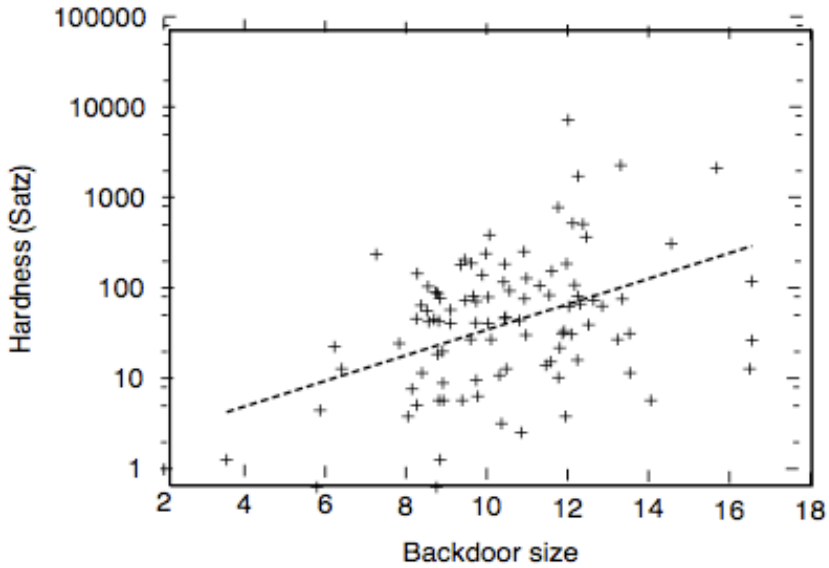
(d) QWH

No correlation between hardness and backdoor size.

Hardness versus Backdoor Size – Instance Level



(e) Morphed GC



(f) QWH

[Ruan et al., AAAI-04]– 100 instances per plot.

Backdoors don't capture dependencies

Intuition

Backdoor size alone does not capture dependencies amongst backdoor variables. The fewer dependent variables, the smaller the probability that some of the backdoors will be set to wrong values.

Backdoors don't capture dependencies

Intuition

Backdoor size alone does not capture dependencies amongst backdoor variables. The fewer dependent variables, the smaller the probability that some of the backdoors will be set to wrong values.

Dependencies must be captured

We need a notion that captures such dependencies – this is the backdoor key.

Backdoor Keys and Key Fractions

Definition (Dependent Variable)

A variable $v \in V$ is a dependent variable of formula ϕ with respect to a partial truth assignment A_B if $\phi[A_B]$ determines v, i.e. there is a unique value assignment $v \mapsto x$ such that $\phi[A_B \cup \{v \mapsto x\}]$ is satisfiable.

Backdoor Keys and Key Fractions

Definition (Dependent Variable)

A variable $v \in V$ is a dependent variable of formula ϕ with respect to a partial truth assignment A_B if $\phi[A_B]$ determines v, i.e. there is a unique value assignment $v \mapsto x$ such that $\phi[A_B \cup \{v \mapsto x\}]$ is satisfiable.

Definition (Backdoor Key)

A backdoor variable v is in the backdoor key set of B with respect to a backdoor truth assignment A_B if and only if v is a dependent variable in ϕ with respect to the partial truth assignment $A_{B-\{v\}}$.

Backdoor Keys and Key Fractions

Definition (Dependent Variable)

A variable $v \in V$ is a dependent variable of formula ϕ with respect to a partial truth assignment A_B if $\phi[A_B]$ determines v, i.e. there is a unique value assignment $v \mapsto x$ such that $\phi[A_B \cup \{v \mapsto x\}]$ is satisfiable.

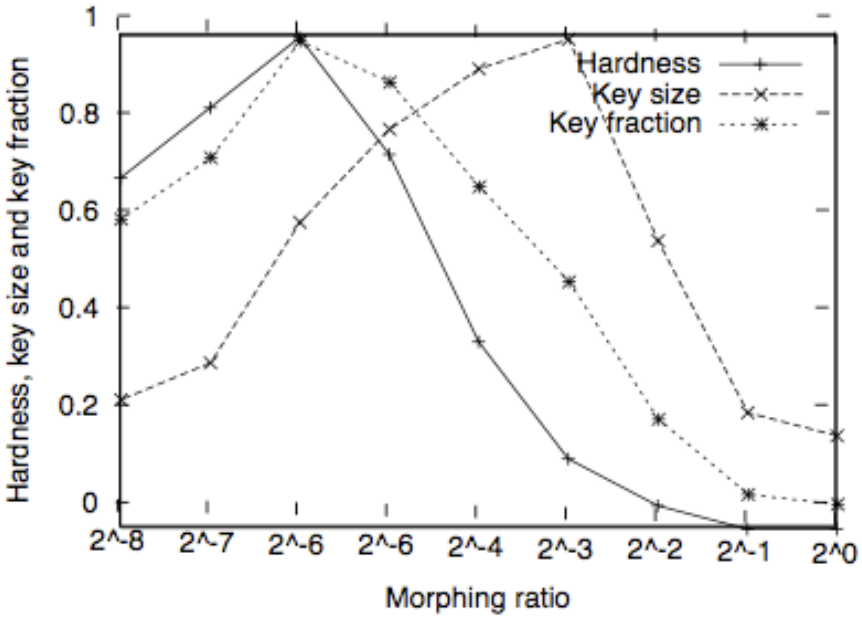
Definition (Backdoor Key)

A backdoor variable v is in the backdoor key set of B with respect to a backdoor truth assignment A_B if and only if v is a dependent variable in ϕ with respect to the partial truth assignment $A_{B-\{v\}}$.

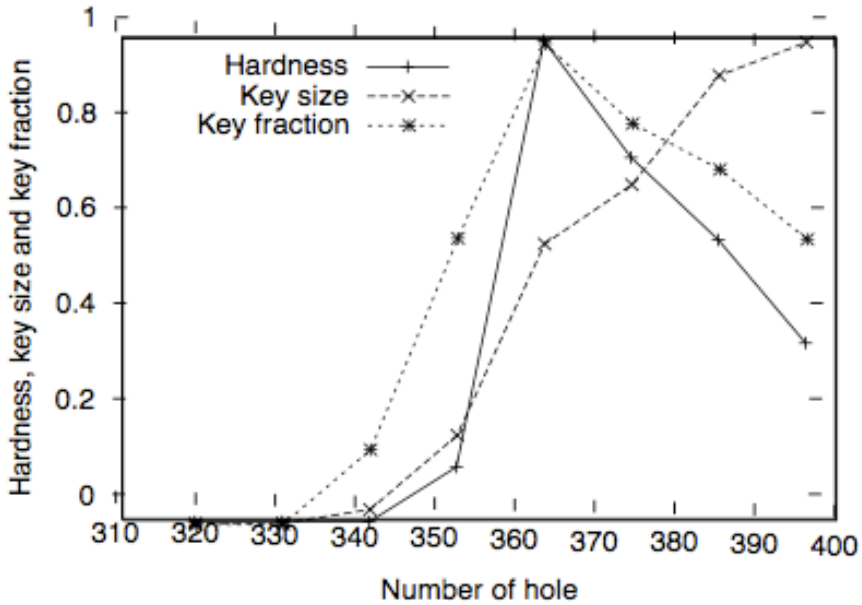
Definition (Backdoor Key Fraction)

A backdoor key fraction is the ratio if the size of the backdoor key set to the size of the corresponding backdoor set.

Hardness vs Backdoor Key Fraction – Ensemble Level



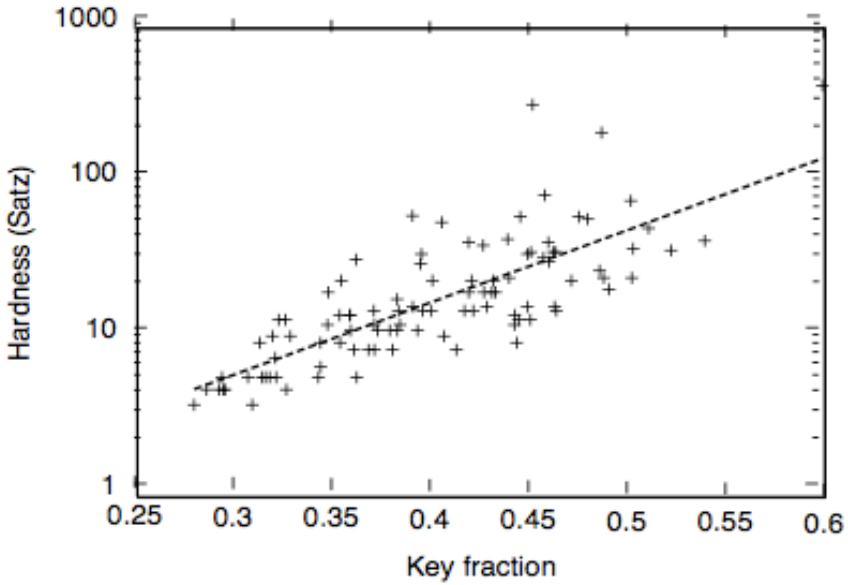
(g) Morphed GC



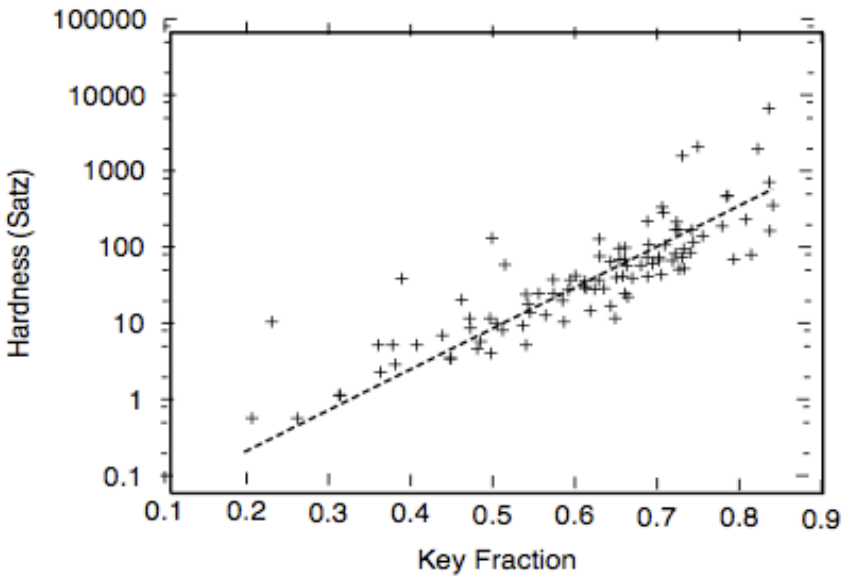
(h) QWH

[Ruan et al., AAAI-04]

Hardness vs Backdoor Key Fraction – Instance Level



(i) Morphed GC



(j) QWH

[Ruan et al., AAAI-04]

Correlation between Hardness and Backdoor Keys

Measures	Satz-rand						zChaff					
	QWH			GCP			QWH			GCP		
	F	K	B	F	K	B	F	K	B	F	K	B
Correlation Coefficient	0.86	0.79	0.42	0.78	0.56	-0.26	0.70	0.70	0.42	0.54	0.53	0.01
RMSE	0.41	0.47	0.71	0.24	0.37	0.38	0.39	0.68	0.87	0.37	0.37	0.38
MAE	0.29	0.37	0.57	0.29	0.28	0.28	0.29	0.53	0.69	0.31	0.31	0.32

Figure: Linear regression results for instance hardness: F is key fraction, K is key size, and B is backdoor size [Ruan et al., AAAI-04]

Backdoors and Backbone variables are different

Weak backdoors

Problem set	Mean backbone size	Mean w_F	Mean $\mid W_F^* \mid$	Mean Overlap
uf20[1]	13.7	0.76	4.4	0
uf50[1]	30.8	1.6	18.8	6.5
uf100[2]	53.6	4.6	10.2	2.7
RTI[2]	53.8	4.5	9.8	2.5

Strong backdoors

Problem set	Mean backbone size	Mean s_F	Mean $\mid S_F^* \mid$	Mean Overlap
uf20[1,3]	15.5	1.1	3.1	0
uf50[1,3]	43.4	2.0	16.4	13.0

[1] These entries are based on systematic search
[2] These entries are based on local search
[3] Only problems where a strong backdoor of size < 4 was found. (Hence mean backbone size differs from weak backdoor table)

[Kilby et al., AAAI-05]

Outline

1. Backdoors – [Williams et al., IJCAI-03] and [Kilby et al., AAAI-05]
   - Polynomial-time Sub-Solvers
   - Weak and Strong Backdoors
   - Backdoors in Realworld Problems
   - Computing Weak and Strong Backdoors
2. Exploiting Backdoors – [Williams et al., IJCAI-03]
   - Deterministic Strategy
   - Randomized strategy
   - Heuristic strategy
3. Backdoors and Problem Hardness – [Ruan et al., AAAI-04]
   - Backdoor Size and Hardness
   - Backdoor Keys and Key Fraction
4. Some Directions of Study
   - Backdoors and Parameterised Complexity
   - Backdoors and Machine Learning

Fixed Parameter Algorithms

Traditional complexity theory

The running time of an algorithm that solves an NP-Hard problem is exponential in the input size n, e.g. SAT is $\mathcal{O}(2^n)$, unless P=NP.

The fixed parameter algorithm view (informal)

- Downey & Fellows, 1997
- Running time of an algorithm is exponential in a parameter k independent of n;
- Only polynomially dependent on n;

Example

For vertex cover $O(1.3^k + n)$, where k is the maximum number of vertices incident to all edges in the given graph of n vertices.

Cyclic cutset (Feedback Vertex Set)

Problem statement

Input: A CSP Z over n variables
Parameter: The cycle-cutset size, k.
Question: Is it possible to remove at most k variables from Z so that the resulting CSP is acylic.

What do we know?

The problem is FPT and can be solved in time

$$\mathcal{O}(5^k k^2 + poly(n))$$

.

Importance?

A cycle cutset is a backdoor in a binary CSP [Dechter].

Generic Backdoor Computation

Problem statement

Input: An instance Z of CSP of SAT, a polynomially solvable class P of the given problem.

Parameter: The size of the backdoor, k

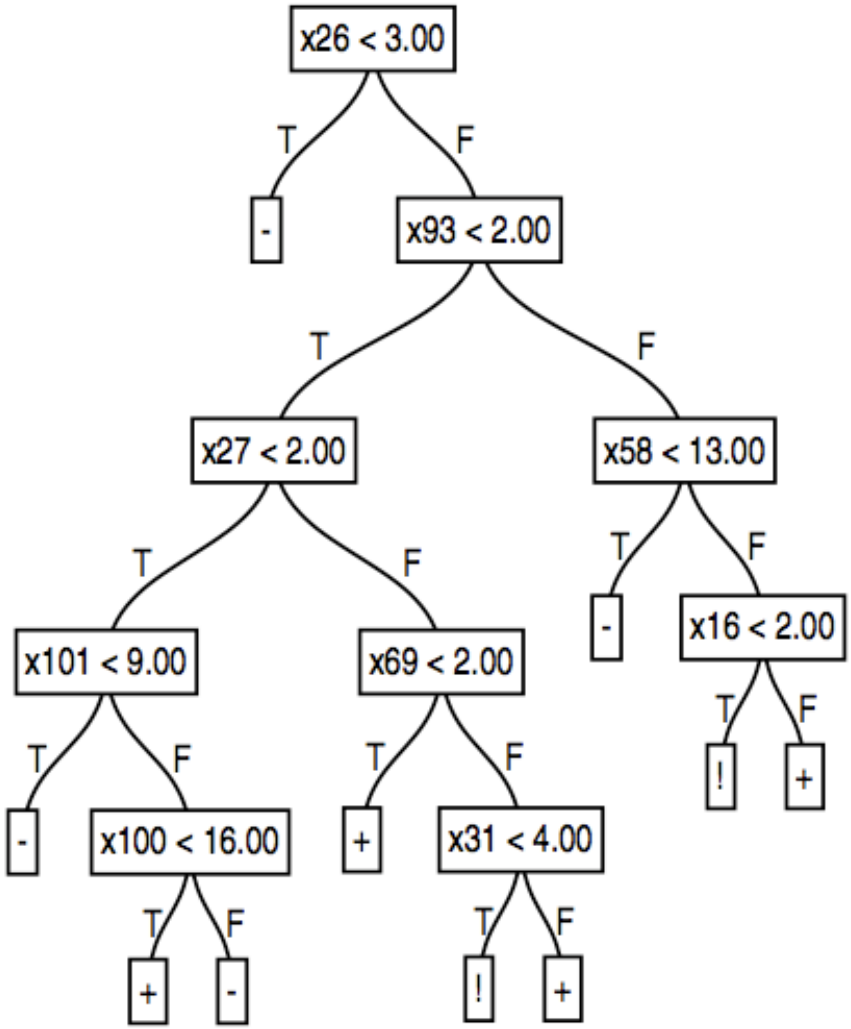
Question: Is it possible to remove at most k variables from Z so that the resulting instance belongs to P?

What do we know?

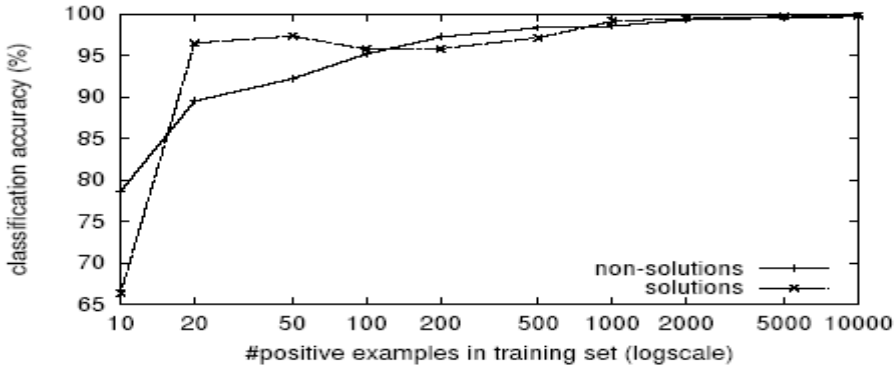
Some classes of this problem are FPT.

[Razgon and O'Sullivan, Journal of Computer and System Sciences, 2009.]

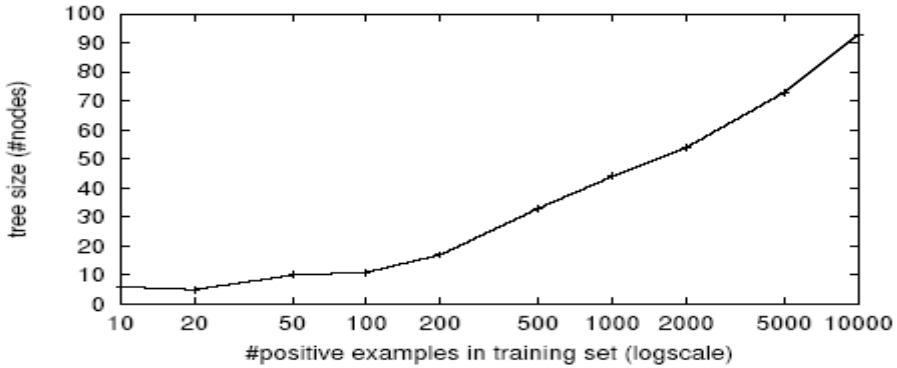
Discovery of Backdoors using Machine Learning?



Looks like a backdoor



(a) Classification accuracy.



(b) Size of the decision tree.

Methodology

- We classified 28,000 solutions and 90,000 non-solutions uniformly distributed throughout the solution space.
- Very small trees, very high classification accuracy (99.9%).
- Tree size is tiny (largest can be stored in a 23kb file) compare with 6.6Mb file to represent the problem, or 3.4Mb if we represent is as an automaton!

Classification of CSP/SAT

Problem Size Features:
1. **Number of clauses**: denoted c
2. **Number of variables:** denoted v
3. **Ratio:** c/v

Variable-Clause Graph Features:
4-8. **Variable nodes degree statistics:** mean, variation coefficient, min, max and entropy.
9-13. **Clause nodes degree statistics:** mean, variation coefficient, min, max and entropy.

Variable Graph Features:
14-17. **Nodes degree statistics:** mean, variation coefficient, min and max.

Balance Features:
18-20. **Ratio of positive and negative literals in each clause:** mean, variation coefficient and entropy.
21-25. **Ratio of positive and negative occurrences of each variable:** mean, variation coefficient, min, max and entropy.
26-27. **Fraction of binary and ternary clauses**

Proximity to Horn Formula:
28. **Fraction of Horn clauses**
29-33. **Number of occurrences in a Horn clause for each variable:** mean, variation coefficient, min, max and entropy.

DPLL Probing Features:
34-38. **Number of unit propagations:** computed at depths 1, 4, 16, 64 and 256.
39-40. **Search space size estimate:** mean depth to contradiction, estimate of the log of number of nodes.

Local Search Probing Features:
41-44. **Number of steps to the best local minimum in a run:** mean, median, 10th and 90th percentiles for SAPS.
45. **Average improvement to best in a run:** mean improvement per step to best solution for SAPS.
46-47. **Fraction of improvement due to first local minimum:** mean for SAPS and GSAT.
48. **Coefficient of variation of the number of unsatisfied clauses in each local minimum:** mean over all runs for SAPS.

Figure: Features from [Xu, Hutter, Hoos, Leyton-Brown, CP 2007].

Classification of CSP/SAT

Classifier	Class	Crafted			Industrial			Random 3SAT			Random		
		Base	All	+t	Base	All	+t	Base	All	+t	Base	All	+t
Forest	SAT	78.9	82.5	81.1	93.3	94.1	**94.9**	**98.2**	**99.4**	**99.8**	93.2	96.2	**99.2**
	UNSAT	81.4	83.9	84.4	92.7	92.8	92.9	96.3	**97.2**	99.3	**90.7**	**94.7**	**97.9**
	ALL	**80.5**	83.4	83.1	93.0	**93.5**	94.0	97.1	**98.1**	99.5	92.0	**95.5**	**98.6**
DT	SAT	**82.2**	**84.5**	**83.4**	87.8	89.7	93.3	98.0	97.3	98.0	96.0	95.3	98.5
	UNSAT	78.0	83.5	85.3	**97.1**	**94.7**	93.8	**96.6**	96.8	**99.5**	88.6	93.4	97.5
	ALL	79.3	**83.9**	**84.6**	91.0	91.5	93.5	**97.2**	97.1	**99.6**	**92.3**	94.4	98.0
MLP	SAT	71.8	72.4	71.5	92.6	92.5	95.0	88.4	93.9	88.4	90.9	92.1	**99.2**
	UNSAT	79.4	81.2	79.6	94.9	92.6	**96.3**	90.8	92.4	99.4	82.7	86.7	97.8
	ALL	76.4	77.6	76.4	93.5	92.5	**95.6**	89.8	93.0	99.4	86.8	89.5	98.5
1-NN	SAT	74.7	79.1	78.1	**95.7**	**94.7**	94.6	–	–	–	–	–	–
	UNSAT	80.2	81.7	80.7	94.0	88.6	87.6	–	–	–	–	–	–
	ALL	78.1	80.7	79.8	**95.0**	92.0	91.5	–	–	–	–	–	–
Bayes	SAT	58.6	65.3	69.0	80.2	86.5	87.3	64.2	64.3	87.6	**99.9**	**99.1**	**99.2**
	UNSAT	**84.8**	**85.9**	**85.6**	75.0	76.7	76.9	94.1	93.8	96.9	57.6	56.5	91.7
	ALL	69.4	75.2	69.4	78.1	82.1	82.6	74.9	74.9	92.4	66.0	64.4	95.4

Figure: Classification accuracy for SAT, even industrial problems, can be very high.

Backdoor Features

- You've heard it here first....
- Learning methods seem to be identifying important structure in the problem, not just specific variables.
- Ongoing work: analyzing the features that give rise to strong classification accuracy.
- I term these backdoor features.

Wrap-up

1. Backdoors – [Williams et al., IJCAI-03] and [Kilby et al., AAAI-05]
   - Polynomial-time Sub-Solvers
   - Weak and Strong Backdoors
   - Backdoors in Realworld Problems
   - Computing Weak and Strong Backdoors
2. Exploiting Backdoors – [Williams et al., IJCAI-03]
   - Deterministic Strategy
   - Randomized strategy
   - Heuristic strategy
3. Backdoors and Problem Hardness – [Ruan et al., AAAI-04]
   - Backdoor Size and Hardness
   - Backdoor Keys and Key Fraction
4. Some Directions of Study
   - Backdoors and Parameterised Complexity
   - Backdoors and Machine Learning

— Tutorial —
Backdoors to Satisfaction

Barry O'Sullivan

b.osullivan@cs.ucc.ie

Cork Constraint Computation Centre
University College Cork, Ireland

http://osullivan.ucc.ie

CP 2010