Backdoors in Combinatorial Problems

Carla P. Gomes
Cornell University

Joint work with:

Ryan Williams and Bart Selman
Carnegie Mellon University and Cornell University
Motivation

Study of Exact / Complete Solvers for hard combinatorial problems

In particular the study of the interplay between:

• certain special structural features of problems instances
• polynomial time inference/consistency methods
• randomization and restart strategies
Backtrack Search: Main Underlying Mechanism of Compete/Exact Methods

Exact / Complete Solvers for hard combinatorial problems

Mathematical Programming (MP)
Constraint Programming (CP)

Tree search methods
Branch & Bound;
Branch & Cut;
Branch & Price;
Intelligent Backtrack search methods
DPPL (SAT)
...

Carla P. Gomes
MURI-UCLA-2005
Defying NP-Completeness

Current state of the art complete or exact solvers can handle very large problem instances of hard combinatorial:

→ We are dealing with formidable search spaces of exponential size --- to prove optimality we have to implicitly search the entire search;

→ the problems we are able to solve are much larger than would predict given that such problems are in general NP complete or harder

But first, what is BIG?
“Real World” (begin)

From “SATLIB”:

http://www.satlib.org/benchm.html

SAT-encoded bounded model checking instances
(contributed by Ofer Shtrichman)

In Bounded Model Checking (BMC) [BCCZ99], a rather newly introduced problem in formal methods, the task is to check whether a given model \( M \) (typically a hardware design) satisfies a temporal property \( P \) in all paths with length less or equal to some bound \( k \). The BMC problem can be efficiently reduced to a propositional satisfiability problem, and in fact if the property is in the form of an invariant (Invariants are the most common type of properties, and many other temporal properties can be reduced to their form. It has the form of 'it is always true that ... '), it has a structure which is similar to many AI planning problems.
I.e., \((\neg x_1) \lor x_7\)  
\((\neg x_1) \lor x_6\)  
 etc.

The instance bmc-ibm-6.cnf, IBM LSU 1997:

```
p cnf .  
  1  7  0 
  1  6  0 
  1  5  0 
  1  4  0 
  1  3  0 
  1  2  0 
  1  8  0 
  9  15 0 
  9  14 0 
  9  13 0 
  9  12 0 
  9  11 0 
  9  10 0 
  9  16 0 
 17  23 0 
 17  22 0 
```
10 pages later:

\[
\begin{align*}
&185 \rightarrow 9 0 \\
&185 \rightarrow 1 0 \\
&177 169 161 153 145 137 129 121 113 105 97 \\
&89 81 73 65 57 49 41 \\
&33 25 17 9 1 \rightarrow 185 0 \\
&186 \rightarrow 187 0 \\
&186 \rightarrow 188 0
\end{align*}
\]

\[\ldots\]

I.e., \((x_{177} \text{ or } x_{169} \text{ or } x_{161} \text{ or } x_{153} \ldots x_{33} \text{ or } x_{25} \text{ or } x_{17} \text{ or } x_{9} \text{ or } x_{1} \text{ or } (\text{not } x_{185}))\]

clauses / constraints are getting more interesting…
4000 pages later:

```
10236  -10050  0
10236  -10051  0
10236  -10235  0
10008  10009  10010  10011  10012  10013  10014
10015  10016  10017  10018  10019  10020  10021
10022  10023  10024  10025  10026  10027  10028
10029  10030  10031  10032  10033  10034  10035
10036  10037  10038  10039  10040  10041  10042
10043  10044  10045  10046  10047  10048  10049
10050  10051  10235  -10236  0
10237  -10008  0
10237  -10009  0
10237  -10010  0
```

...
Finally, 15,000 pages later:

The Chaff SAT solver (Princeton) solves this instance in a few seconds.

\[ 2^{50000} \approx 3.160699437 \cdot 10^{15051} \]
Gap between theory and practice

The good scaling behavior of state of the art complete solvers seems to defy the worst-case complexity results for NP complete problems!

How can we explain this gap between theory and practice?

What makes this possible?
Inference and Search

• Inference at each node of search tree:

  MP uses LP relaxations and cutting planes;
  CP - domain reduction constraint propagation and no-good learning.

• Search

  Different search enhancements in terms of variable and value selection strategies, probing, randomization etc, while guaranteeing the completeness of the search procedure.
Real World Problems are also characterized by

Hidden *tractable* substructure in real-world problems.

Can we make this more precise?

Proposal:

*We consider particular structures we call backdoors.*

Connections to Domitilla’s and Reza’s talk!!!
Outline

• **Backdoors**
  Intuitions and formal definition

• **Connections between backdoors and heavy-tailedness**
  Small backdoor sets provide a formal model for the power laws in combinatorial search

• **Backdoors in problem instances**
  Real world problems have surprisingly small backdoor sets

• **Algorithms for exploiting Backdoors**
  Nice results showing that we can have provably optimal complete randomized algorithms for general constraint satisfaction problems, when the backdoor set is small;

• **Conclusions**
Backdoors
Real World Problems are characterized by Hidden *Tractable* Substructure

**BACKDOORS**

Subset of the “critical” variables such that once assigned a value the instance simplifies to a tractable class.

Explain how a solver can get “lucky” and solve very large instances
Backdoors to tractability

Informally:

A backdoor to a given problem is a subset of its variables such that, once assigned values, the remaining instance simplifies to a tractable class.

Formally:

We define notion of a “sub-solver”
(handles tractable substructure of problem instance)

backdoors and strong backdoors
Constraint Satisfaction/Optimization Formulations (CSP)

Given:

- A finite set of variables;
- A finite set of constraints;
- With each variable is associated a non-empty finite domain.
- A constraint on k variables $X_1, \ldots, X_k$ is a relation $R(X_1, \ldots, X_k) \subseteq D_1 \times \ldots \times D_k$.

A solution to a CSP is an assignment of values to all the variables, satisfying (optimally) all the constraints.
Defining a sub-solver

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

- *(Trichotomy)* A either rejects the input C, or “determines” C correctly (as unsatisfiable or satisfiable, returning a solution if satisfiable),

- *(Efficiency)* A runs in polynomial time,

- *(Trivial solvability)* A can determine if C is trivially true (has no constraints) or trivially false (has a contradictory constraint),

- *(Self-reducibility)* if A determines C, then for any variable \( x \) and value \( v \), then A determines \( C[v/x] \).

Definition is general enough to encompass many polynomial time propagation methods. Also those for which there does not exist a clean syntactical characterization of the tractable subclass. Valid for different encoding e.g., Mixed Integer Programming, Constraint Programming and Satisfiability.
**Defining backdoors**

**Backdoors (for satisfiable instances):**

**Definition**  
A nonempty subset $S$ of the variables is a backdoor in $C$ for $A$ if for some $a_S : S \rightarrow D$, $A$ returns a satisfying assignment of $C[a_S]$.

**Strong backdoors (apply to satisfiable or inconsistent instances):**

**Definition**  
A nonempty subset $S$ of the variables is a strong backdoor in $C$ for $A$ if for all $a_S : S \rightarrow D$, $A$ returns a satisfying assignment or concludes unsatisfiability of $C[a_S]$. 

Carla P. Gomes  
MURI-UCLA- 2005  
19
On the connections between backdoors and heavy-tailedness
Heavy-tailed distributions

Certain problems, when solved by randomized backtracking, yield a runtime distribution that is heavy-tailed.

Exponential decay for standard distributions, e.g. Normal, Logonormal, exponential:

\[ \Pr[X > x] \approx Ce^{-x^2}, \quad x > 0 \]

Heavy-Tailed Power Law Decay, e.g. Pareto-Levy:

\[ \Pr[X > x] =Cx^{-\alpha}, x > 0 \]

(Frost et al 97; Gomes et al 97, Hoos 1999, Walsh 99)
Fat and Heavy-tailed distributions

- Explain very long runs of complete solvers;
- But also imply the existence of a wide range of solution times, often from very short runs to very long

How to explain short runs?

Backdoors
Connection Between Heavy-Tails and Backdoors

$T$ - the number of leaf nodes visited up to and including the successful node; $b$ - branching factor

$$P[T = b^i] = (1 - p) p^i \quad i \geq 0$$

1 backdoor

(b = 2) successful leaf

(Chen, Gomes, and Selman)
Three Regimes of Behavior

Regime 1:
finite expected time, finite variance

Regime 2:
finite expected time, infinite variance

Regime 3:
infinite expected time, infinite variance

Tail:
when \( p > \frac{1}{b^2} \) we have

\[
P[T > L] > p^2 L \log b^p = CL\alpha \quad \alpha < 2
\]

(see paper for formal proofs)

\[
p \leq \frac{1}{b^2}
\]

\[
\frac{1}{b^2} < p < \frac{1}{b}
\]

\[
p \geq \frac{1}{b}
\]
More than 1 backdoor
Backdoors provides detailed formal model for heavy-tailed search behavior.

Theorem 3. (Heavy-tail lower bound) If $B \in o\left(\frac{N}{\log N}\right)$ and the probability of heuristic failure $(1 - 1/\theta)$ is less than $1/\theta^\alpha$ for $\alpha \in (0, 2)$, then the tail probability of the search cost on $V(B)$ is lower bounded by a heavy-tail.

Can formally relate size of backdoor and strength of heuristics (captured by its failure probability to identify backdoor variables) to occurrence of heavy tails in backtrack search.
**Logistics planning problem formula**
843 vars, 7,301 constraints – 16 backdoor variables

*Visualization by Anand Kapur*
Backdoors in problems instances
Backdoors can be surprisingly small:

Backdoors explain how a solver can get “lucky” on certain runs, when the backdoors are identified early on in the search.

<table>
<thead>
<tr>
<th>instance</th>
<th># vars</th>
<th># clauses</th>
<th>backdoor</th>
<th>fract.</th>
</tr>
</thead>
<tbody>
<tr>
<td>logistics.d</td>
<td>6783</td>
<td>437431</td>
<td>12</td>
<td>0.0018</td>
</tr>
<tr>
<td>3bitadd_32</td>
<td>8704</td>
<td>32316</td>
<td>53</td>
<td>0.0061</td>
</tr>
<tr>
<td>pipe_01</td>
<td>7736</td>
<td>26087</td>
<td>23</td>
<td>0.0030</td>
</tr>
<tr>
<td>qg_30_1</td>
<td>1235</td>
<td>8523</td>
<td>14</td>
<td>0.0113</td>
</tr>
<tr>
<td>qg_35_1</td>
<td>1597</td>
<td>10658</td>
<td>15</td>
<td>0.0094</td>
</tr>
</tbody>
</table>
Job Shop Scheduling Problem

Job-Shop Scheduling: 10 jobs on 10 machines.
Proposed by Fischer and Tompson in 1963.
Solved by Carlier and Pinson in 1990!
The backdoor set corresponds to defining the correct order between two jobs on a given machine.
Synthetic Planning Domains

Connections to work by M. Earl and R. D’Andrea (Cornell University)

Most recently:

*We are building synthetic planning domains with small $O(\log n)$ backdoors sets or even smaller, near constant size backdoors;*

Research questions –

*provide semantics to the backdoors sets;*

Intuition:-

*Backdoors capture critical problem resources (bottlenecks).*

Joint work with Joerg Hoffmann

Improve MILP efficiency using randomization techniques to find backdoors (see algorithm for exploiting backdoors)
Exploiting Backdoors
We cover three kinds of strategies for dealing with backdoors:

- A **deterministic algorithm**
- A **complete randomized algorithm**
  - *Provably better performance over the deterministic one*
- A **heuristically guided complete randomized algorithm**
  - *Assumes existence of a good heuristic for choosing variables to branch on*
  - *We believe this is close to what happens in practice*
Algorithm 4.1  Given a CSP $C$ with $n$ variables, For $i = 1, \ldots, n$, For all subsets $S$ of the $n$ variables with $|S| = i$, Perform a standard backtrack search (just on the variables in $S$) for an assignment that results in $C$ being solved by sub-solver $A$. 
**Randomized Generalized Iterative Deepening**

**Assumption:**
There exists a backdoor whose size is bounded by a function of $n$ (call it $B(n)$)

**Idea:**
Repeatedly choose random subsets of variables that are slightly larger than $B(n)$, searching these subsets for the backdoor
Randomized Generalized Iterative Deepening

Algorithm 4.2 Given a CSP $C$ with $n$ variables,
Repeat $n \left( \frac{n/B(n)-1}{b-1} \right) B(n)$ times (and at least once):

Randomly choose a subset $S'$ of the $n$ variables, of size $b \cdot B(n)$. Perform a standard backtrack search on variables in $S'$. If $C$ is ever solvable by $A$, return the satisfying assignment.
Deterministic Versus Randomized

Suppose variables have 2 possible values (e.g. SAT)

For $B(n) = n/k$, algorithm runtime is $c^n$

Det. algorithm outperforms brute-force search for $k > 4.2$
Assume we have the following.

**DFS**, a generic depth first search randomized backtrack search solver with:

- *(polytime)* sub-solver \( A \)
- Heuristic \( H \) that (randomly) chooses variables to branch on, in polynomial time
  - \( H \) has probability \( 1/h \) of choosing a backdoor variable (*h is a fixed constant*)

Call this ensemble \( (\text{DFS, } H, A) \)
Polytime Restart Strategy for (DFS, H, A)

Theorem 4.3 If the size of a backdoor of a CSP $C$ is $B \leq \frac{c \log N}{\log h + \log a}$ for some constant $c$, then (DFS, H, A) has a restart strategy that solves $C$ in polynomial time.

Essentially:

If there is a small backdoor,

then (DFS, H, A) has a restart strategy that runs in polytime.
### Runtime Table for Algorithms

<table>
<thead>
<tr>
<th>$B(n)$</th>
<th>deterministic</th>
<th>randomized</th>
<th>heuristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n/k$</td>
<td>small $\exp(n)$</td>
<td>smaller $\exp(n)$</td>
<td>tiny $\exp(n)$</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>$\left(\frac{n}{\sqrt{\log n}}\right)^{O(\log n)}$</td>
<td>$\left(\frac{n}{\log n}\right)^{O(\log n)}$</td>
<td>$\poly(n)$</td>
</tr>
<tr>
<td>$O(1)$</td>
<td>$\poly(n)$</td>
<td>$\poly(n)$</td>
<td>$\poly(n)$</td>
</tr>
</tbody>
</table>

DFS, H, A

$B(n) = \text{upper bound on the size of a backdoor, given n variables}$

When the backdoor is a constant fraction of $n$, there is an exponential improvement between the randomized and deterministic algorithm.
Summary

Notion of a “backdoor” set of variables.

1) Captures the combinatorics of a problem instance, as dealt with in practice.
2) Provides insight into restart strategies.
3) Backdoors can be surprisingly small in practice.
4) Search heuristics + randomization can be used to find them, provably efficiently.

Current/Future Work:
Capturing Characteristic Combinatorial Structure in Synthetic Domains
The End

😊

www.cs.cornell.edu/gomes