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)
…

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.

Real World cont.

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

I.e., ((not x_1) or x_7)
((not x_1) or x_6)
etc.

10 pages later:

:

185 −9 0
185 −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 −185 0
186 −187 0
186 −188 0

…

**I.e., (x_177 or x_169 or x_161 or x_153 …
x_33 or x_25 or x_17 or x_9 or x_1 or (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 10086 10087 10088 10089 10090
 10098 10099 10100 10101 10102 10103 10104
 10105 10106 10107 10108 −55 −54 53 −52 −51 50
 10047 10048 10049 10050 10051 10235 −10236 0
10237 −10008 0
10237 −10009 0
10237 −10010 0

…

Finally, 15,000 pages later:

-7 260 0
7 -260 0
1072 1070 0
-15 -14 -13 -12 -11 -10 0
-15 -14 -13 -12 -11 10 0
-15 -14 -13 -12 11 -10 0
-15 -14 -13 -12 11 10 0
-7 -6 -5 -4 -3 -2 0
-7 -6 -5 -4 -3 2 0
-7 -6 -5 -4 3 -2 0
-7 -6 -5 -4 3 2 0
185 0

$$2^{50000} \approx 3.160699437 \cdot 10^{15051}$$

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

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.

Tractable Problem Sub-structure

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

Backdoors: intuitions

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

Strong backdoors (apply to satisfiable or inconsistent instances):

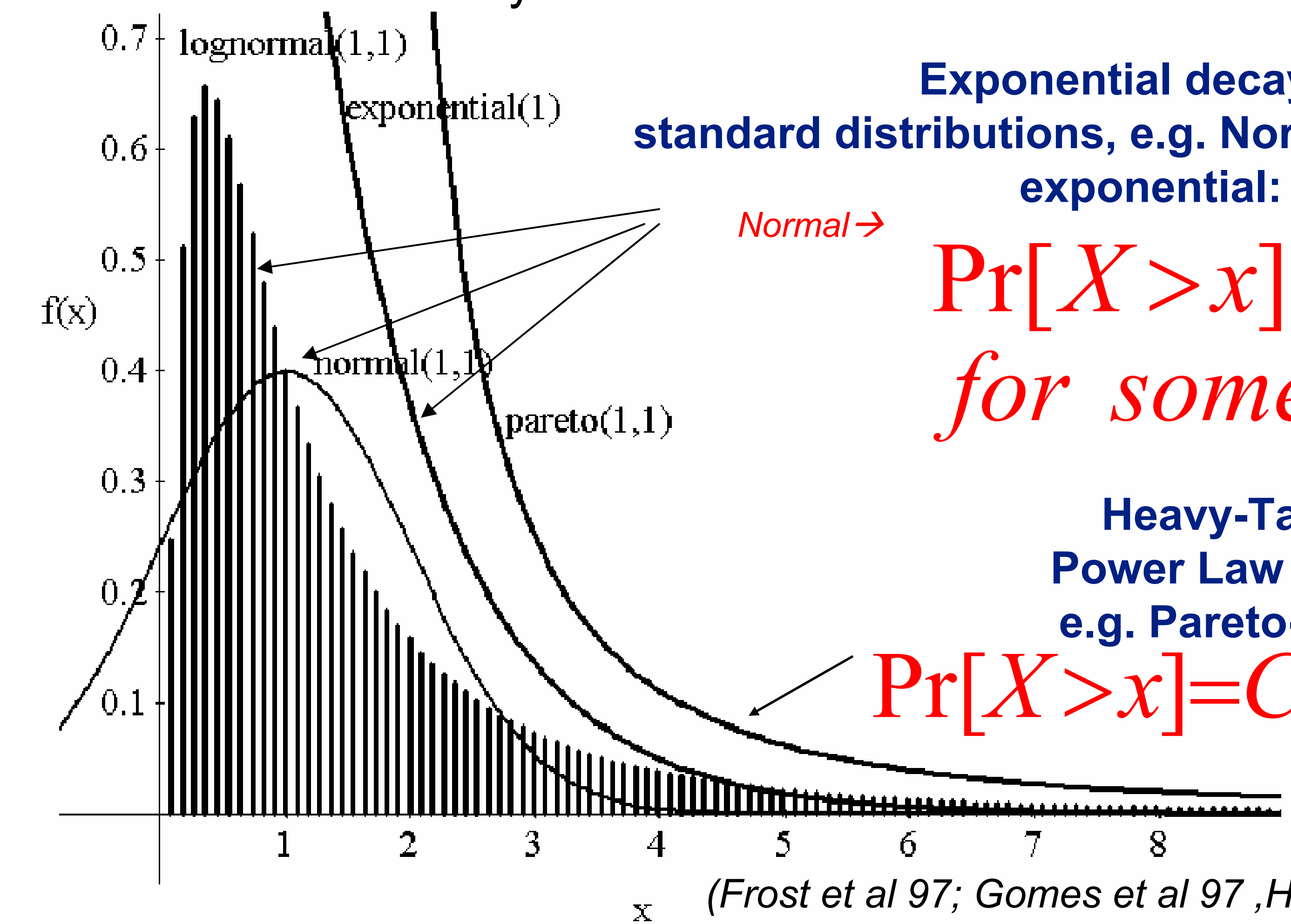
Definition **[strong backdoor]** *A nonempty subset S of the variables is a* strong backdoor in C for A *if for all $a_S : S \to D$, A returns a satisfying assignment or concludes unsatisfiability of $C[a_S]$.*

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:

Normal→ $$\Pr[X > x] \approx Ce^{-x^2}, \quad for\ some\ C > 0$$

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;

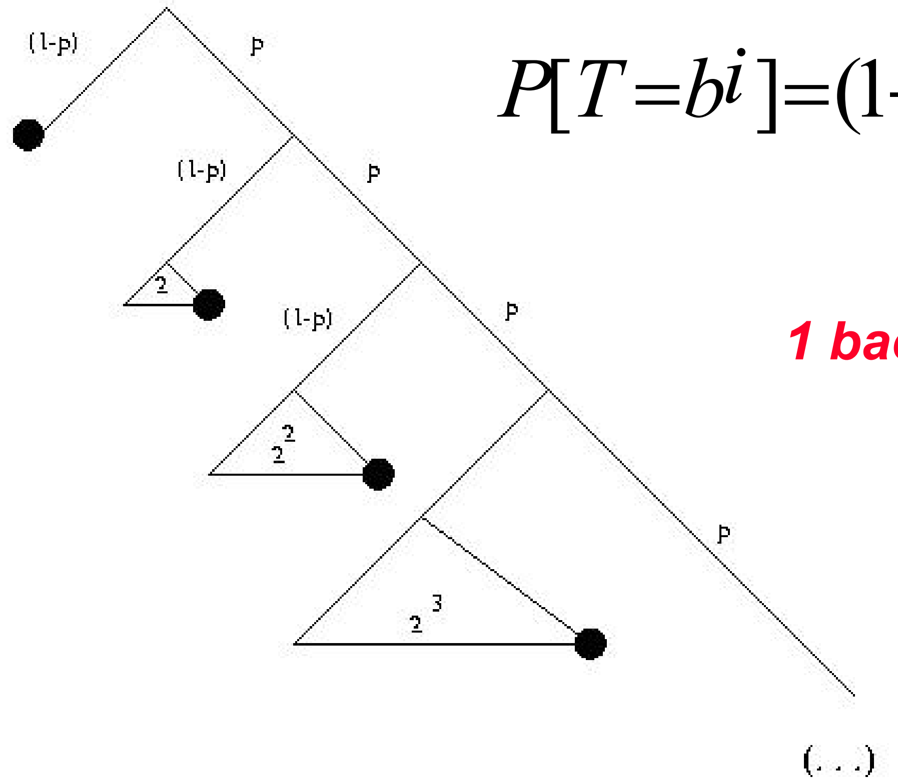
➔But also imply the existence of a wide range of solution times, often from very short runs to very long

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

(see paper for formal proofs)

Regime 1:
finite expected time, finite variance

$$p \leq 1/b^2$$

Regime 2:
finite expected time, infinite variance

$$1/b^2 < p < 1/b$$

Regime 3:
infinite expected time, infinite variance

$$p \geq 1/b$$

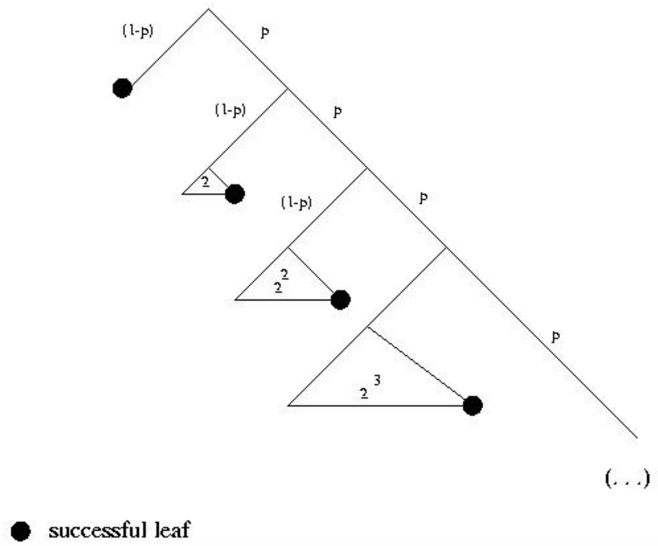
Tail:

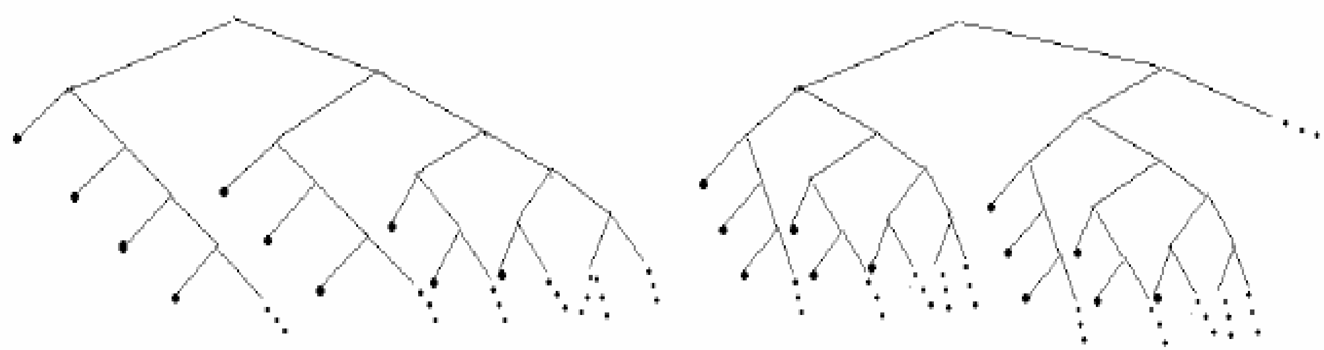
when $p > 1/b^2$ **we have**

p –probability of not finding the backdoor

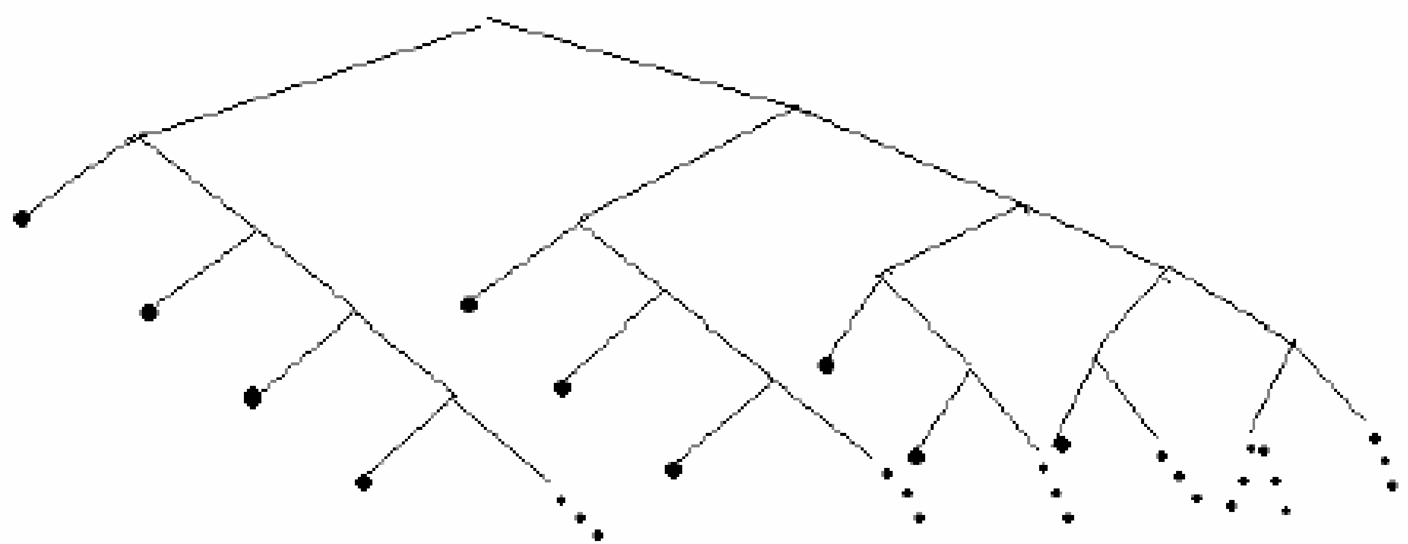
$$P[T > L] > p^2 L^{\log_b p} = C L^{\alpha} \qquad \alpha < 2$$

More than 1 backdoor





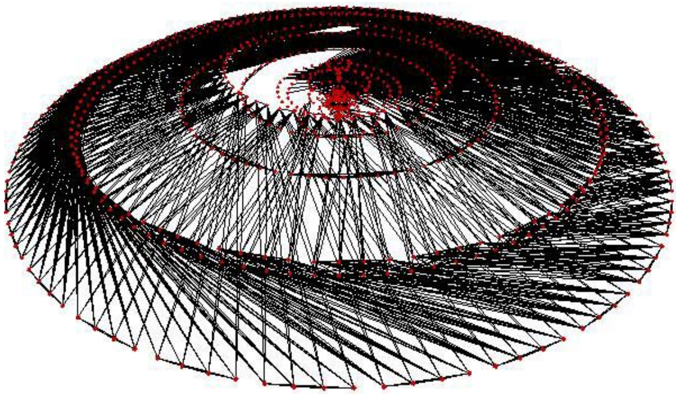
Backdoors provides detailed formal model for heavy-tailed search behavior.



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

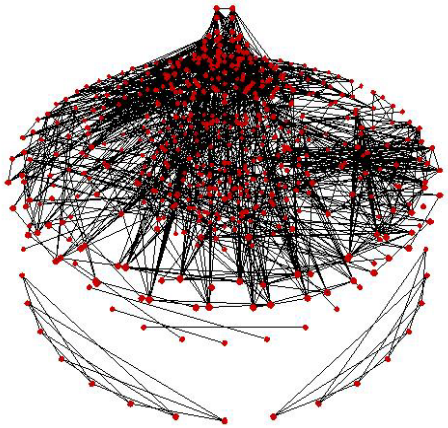
Can formally relate size of backdoor and strength of heuristics (captured by its failure probability to identify backdoor variables) to occurrence of heavy tails in backtrack search.

Backdoors: Visualization

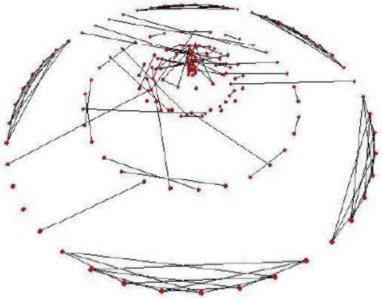


Initial Constraint Graph

Logistics planning problem formula
843 vars, 7,301 constraints – 16 backdoor variables
(visualization by Anand Kapur)



After setting 5 backdoor vars



After setting 12 backdoor vars

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.

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

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**!**

Job Scheduling Hard Problem Instance

Job#	Machine Order / Duration									
1	0 29	1 78	2 09	3 36	4 49	5 11	6 62	7 56	8 44	9 21
2	0 43	2 90	4 75	9 11	3 69	1 28	6 46	5 46	7 72	8 30
3	1 91	0 85	3 39	2 74	8 90	5 10	7 12	6 89	9 45	4 33
4	1 81	2 95	0 71	4 99	6 09	8 52	7 85	3 98	9 22	5 43
5	2 14	0 06	1 22	5 61	3 26	4 69	8 21	7 49	9 72	6 73
6	2 84	1 02	5 52	3 95	8 48	9 72	0 47	6 65	4 06	7 25
7	1 46	0 37	3 61	2 13	6 32	5 21	9 32	8 89	7 30	4 55
8	2 31	0 86	1 46	5 74	4 32	6 88	8 19	9 48	7 36	3 79
9	0 76	1 69	3 76	5 51	2 85	9 11	6 40	7 89	4 26	8 74
10	1 85	0 13	2 61	6 07	8 64	9 76	5 47	3 52	4 90	7 45

The backdoor set corresponds to defining the correct order between two jobs on a given machine

Synthetic Plannnig 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;

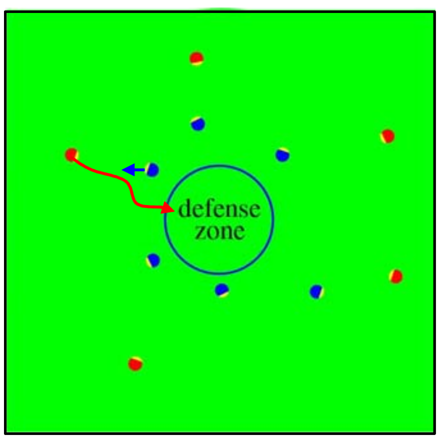
Joint work with Joerg Hoffmann

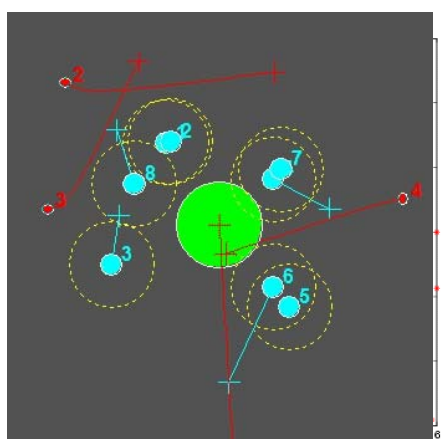
Research questions –

provide semantics to the backdoors sets;

Intuition:-

Backdoors capture critical problem resources (bottlenecks).





Improve MILP efficiency using randomization techniques to find backdoors (see algorithm for exploiting backdoors)

Exploiting Backdoors

Algorithms

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* heuristicly 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***

Deterministic Generalized Iterative Deepening

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

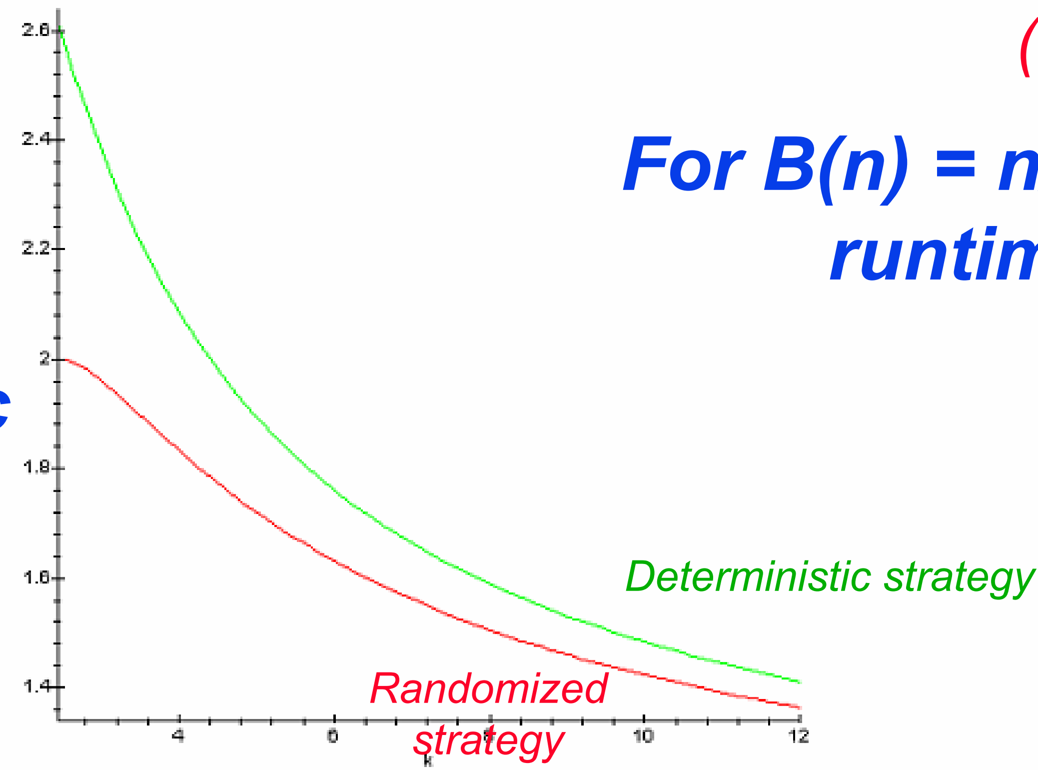
Algorithm 4.2 Given a CSP C with n variables,

Repeat $n(\frac{(n/B(n)-1)}{(b-1)})^{B(n)}$ 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

Runtime exponents.



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

c

k

Complete Randomized Depth First Search with Heuristic

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 **(DFS, *H*, *A*)**

Polytime Restart Strategy for (DFS, H, A)

Theorem 4.3 *If the size of a backdoor of a CSP C is $B \leq \frac{c \log N}{\log h + \log d}$ 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

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

DFS,H,A

B(n) = 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