



A Compressed Breadth-First Search for Satisfiability

DoRon B. Motter and Igor L. Markov
University of Michigan, Ann Arbor



Motivation

- SAT is a fundamental problem in CS thry & apps
- “Efficient” SAT solvers abound (GRASP, Chaff)
- Many small instances are still difficult to solve
- We are pursuing **novel algorithms** for SAT facilitated by data structures with compression
 - Zero-suppressed Binary Decision Diagrams (ZDDs)
- **Existing algorithms** can be implemented w ZDDs
 - The DP procedure: Simon and Chatalic, *IJCAI 2000*
 - DLL: Aloul, Mneimneh and Sakallah, *DATE 2002*



Outline

- Background
 - Partial truth assignments and implied clause classification
 - Representing collections of subsets with Zero-Suppressed BDDs (ZDDs)
- Cassatt: a simple example
- Cassatt: algorithm overview
 - Outer loop: process one variable at a time
 - Processing a given variable
 - Efficiency improvements using ZDDs
- Empirical results and conclusions



Partial Truth Assignments

- SAT instance: $\{V, C\}$
 - V : set of variables $\{a, b, \dots, n\}$
 - C : set of clauses
 - Each clause is a set of literals over V
- Partial truth assignment to some $V' \subseteq V$
 - If it makes all literals in some clause false
 - call it **invalid**
 - Otherwise, call the assignment **valid**

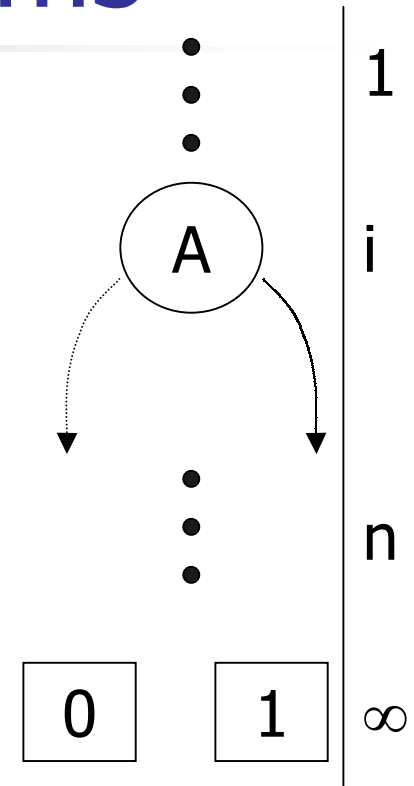


Clause Classification

- With respect to a valid truth assignment, no clauses evaluate to false
- ⇒ Every clause must be either
 - Unassigned
 - No literals in this clause are assigned
 - Satisfied
 - At least one literal in this clause is true
 - Open
 - At least one literal assigned, and all such literals are false
- {Open clauses} \Leftrightarrow partial truth assignment
- ⇒ Store sets of open clauses instead of assignments

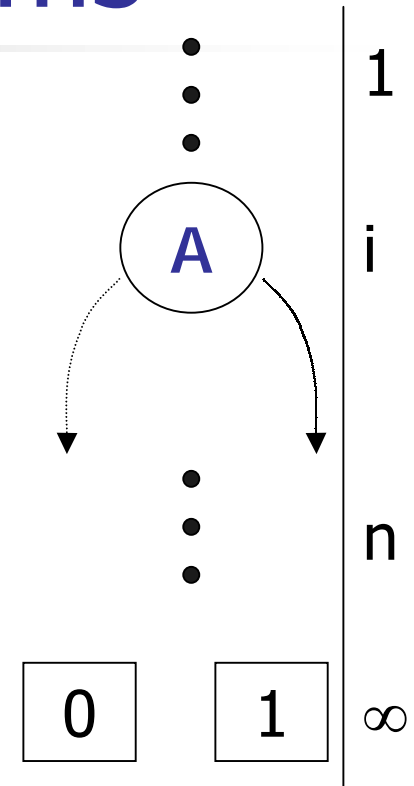
Binary Decision Diagrams

- BDD: A directed acyclic graph (DAG)
 - Unique source
 - Two sinks: the 0 and 1 nodes
- Each node has
 - Unique label
 - Level number
 - Two children at lower levels
 - T-Child and E-Child
- BDDs can represent Boolean functions
 - Evaluation is performed by a single DAG traversal



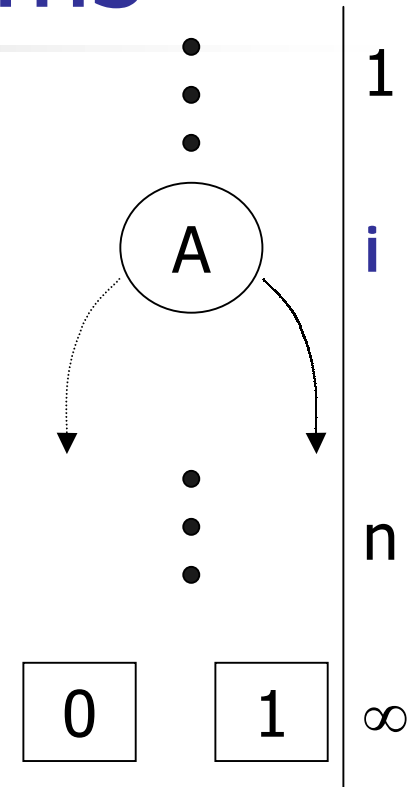
Binary Decision Diagrams

- BDD: A directed acyclic graph (DAG)
 - Unique source
 - Two sinks: the 0 and 1 nodes
- Each node has
 - Unique label
 - Level number
 - Two children at lower levels
 - T-Child and E-Child
- BDDs can represent Boolean functions
 - Evaluation is performed by a single DAG traversal



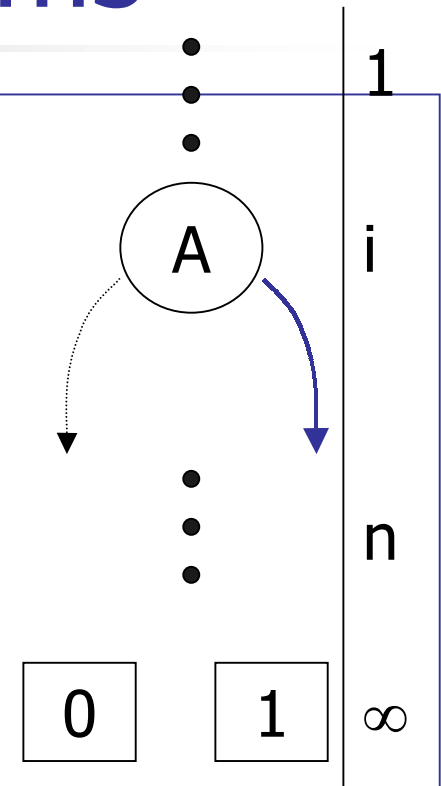
Binary Decision Diagrams

- BDD: A directed acyclic graph (DAG)
 - Unique source
 - Two sinks: the 0 and 1 nodes
- Each node has
 - Unique label
 - Level number
 - Two children at lower levels
 - T-Child and E-Child
- BDDs can represent Boolean functions
 - Evaluation is performed by a single DAG traversal



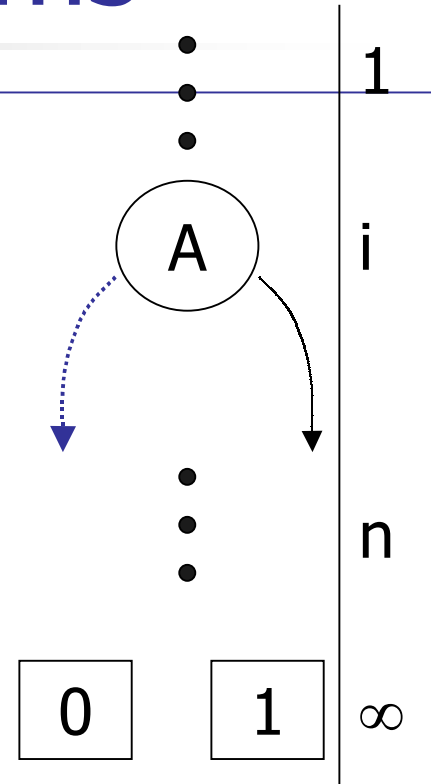
Binary Decision Diagrams

- BDD: A directed acyclic graph (DAG)
 - Unique source
 - Two sinks: the 0 and 1 nodes
- Each node has
 - Unique label
 - Level number
 - Two children at lower levels
 - T-Child and E-Child
- BDDs can represent Boolean functions
 - Evaluation is performed by a single DAG traversal



Binary Decision Diagrams

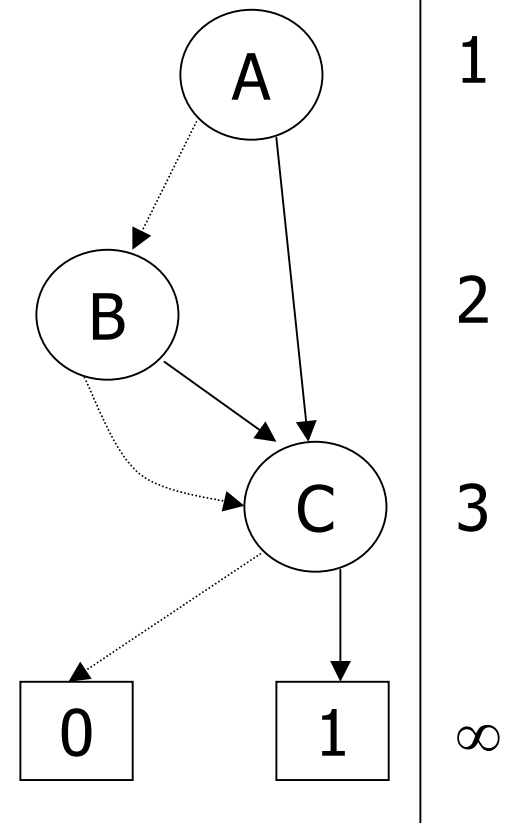
- BDD: A directed acyclic graph (DAG)
 - Unique source
 - Two sinks: the 0 and 1 nodes
- Each node has
 - Unique label
 - Level number
 - Two children at lower levels
 - T-Child and E-Child
- BDDs can represent Boolean functions
 - Evaluation is performed by a single DAG traversal



ZDD: Example

- Collection of subsets:

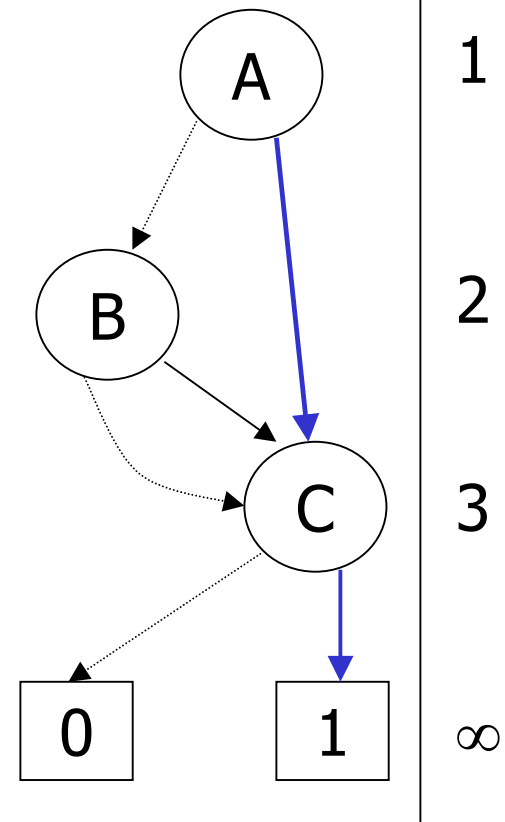
- $\{1, 3\}$
- $\{2, 3\}$
- $\{3\}$



ZDD: Example

- Collection of subsets:

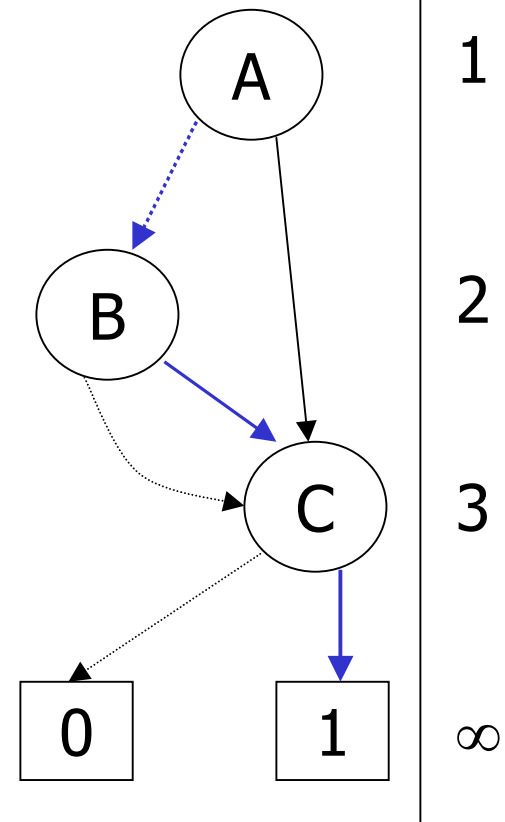
- $\{1, 3\}$
- $\{2, 3\}$
- $\{3\}$



ZDD: Example

- Collection of subsets:

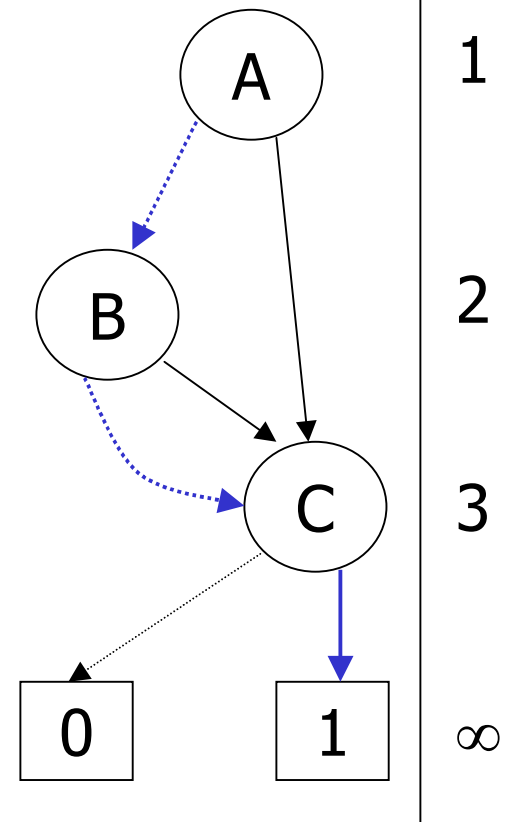
- {1, 3}
- {2, 3}
- {3}



ZDD: Example

- Collection of subsets:

- $\{1, 3\}$
- $\{2, 3\}$
- $\{3\}$





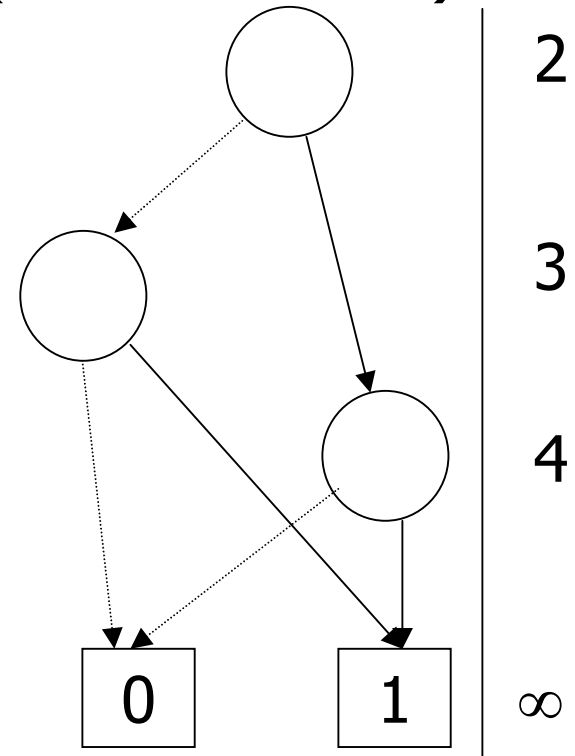
Zero-Suppressed BDDs (ZDDs)

- Zero-suppression rule
 - Eliminate nodes whose T-Child is 0
 - No node with a given index \Rightarrow assume a node whose T-child is 0
- ZDDs can store a collection of subsets
 - Encoded by the collection's characteristic function
 - 0 is the empty collection \emptyset
 - 1 is the one-collection of the empty set $\{\emptyset\}$
- Zero-suppression rule enables **compact representations of sparse or regular collections**

Cassatt: Example

$$(b + c + \sim d)(a+b)(\sim a + b + d)(a + \sim b + \sim c)$$

- $a \leftarrow 1$
 - activates clause 3 (satisfies 2, 4)
- $a \leftarrow 0$
 - activates clauses 2, 4 (sat 3)
- "Cut" clauses
 - 2, 3, 4





Cassatt: Example

$$(b + c + \sim d)(a + b)(\sim a + b + d)(a + \sim b + \sim c)$$

- $b \leftarrow 1$
 - satisfies clauses 2, 3 (and 1)
- $b \leftarrow 0$
 - activates 1, satisfies 4, violates 2
- “Cut” clauses
 - 1, 3, 4

$$\boxed{1} \mid \infty$$



Cassatt: Example

$$(b + c + \sim d)(a + b)(\sim a + b + d)(a + \sim b + \sim c)$$

- $c \leftarrow 1$
 - violates 4, satisfies 1
- $c \leftarrow 0$
 - satisfies 4
- “Cut” clauses
 - 1, 3

$$\boxed{1} \mid \infty$$



Cassatt: Example

$$(b + c + \sim d)(a + b)(\sim a + b + d)(a + \sim b + \sim c)$$

- $d \leftarrow 1$
 - violates 1, satisfies 3
- $d \leftarrow 0$
 - violates 3, satisfies 1
- “Cut” clauses
 - \emptyset

$$\boxed{1} \mid \infty$$



Cassatt: Algorithm Overview

- Maintain collection of subsets of open clauses
 - Analogous to maintaining all “promising” partial solutions of increasing depth
 - Enough information for BFS on the solution tree
- This collection of sets is called the **front**
 - Stored and manipulated in compressed form (ZDD)
 - Assumes a clause ordering (global indices)
 - Clause indices correspond to node levels in the ZDD
- Algorithm: expand one variable at a time
 - When all variables are processed **two cases possible**
 - The front is $\emptyset \Rightarrow$ Unsatisfiable
 - The front is $\{\emptyset\} \Rightarrow$ Satisfiable



Cassatt: Algorithm Overview

```
Front  $\leftarrow$  1      # assign  $\{\emptyset\}$  to front
foreach  $v \in$  Vars
    Front2  $\leftarrow$  Front
    Update(Front,  $v \leftarrow 1$ )
    Update(Front2,  $v \leftarrow 0$ )
    Front  $\leftarrow$  Front  $\cup_s$  Front2
if Front == 0 return Unsatisfiable
if Front == 1 return Satisfiable
```



Processing a Single Variable

- Given:
 - Subset S of open clauses
 - Assignment of 0 or 1 to a single variable x
- Do they imply that some clauses must be violated?
 - I.e., does it correspond to a partial valid truth assignment ? (otherwise, can prune it)
- What subset S' of clauses corresponds to the new truth assignment?
- In our BFS algorithm, we consider both 0 and 1



Detecting Violated Clauses

- Variables are processed in a static order
 - ⇒ Within each clause, some literal must be processed last
 - ⇒ The **end** literal of a clause is known beforehand
- For all literals in clause C to be false, it is *necessary and sufficient* that
 - Clause C must be open
 - The **end** literal of C must be assigned false



New Set of Open Clauses

- Given:
 - Subset S of open clauses
 - Assignment of 0 or 1 to a single variable x
 - The combination of the two is valid
- What subset S' corresponds to the new truth assignment?



New Set of Open Clauses

- Given:
 - Subset S of open clauses
 - Assignment of 0 or 1 to a single variable x
- In the table below, select
 - Row: current status of a clause $C \in S$
 - Column: location of literal l in C (l corresp. to x)

	Beginning	Middle	End	None
Satisfied	Impossible	No Action	No Action	No Action
Open	Impossible	if($t(l) = 0$) $S' \leftarrow S' \cup C$	No Action	$S' \leftarrow S' \cup C$
Unassigned	if($t(l) = 0$) $S' \leftarrow S' \cup C$	Impossible	Impossible	No Action



Gaining Efficiency Using ZDDs

- Use ZDD to store the collection of all subsets of open clauses (front)
 - Achieves data compression (in some cases, with exponential compression ratio)
 - Improves memory requirements of BFS
- Use ZDD algorithms to consider all subsets in the ZDD at the same time
 - Implicit (symbolic) manipulation of compressed data



Gaining Efficiency Using ZDDs

- Given:
 - Assignment of 0 or 1 to a single variable x
- Consider its effect on all clauses
 - It **violates** some clauses
 - x corresponds to the end literal $/$ of some clause C_i and $/$ is assigned false
 - It **satisfies** some clauses
 - x appears in C_i and its literal $/$ is assigned true
 - It **activates** some clauses
 - x corresponds to the beginning literal $/$ for C_i and $/$ is assigned false

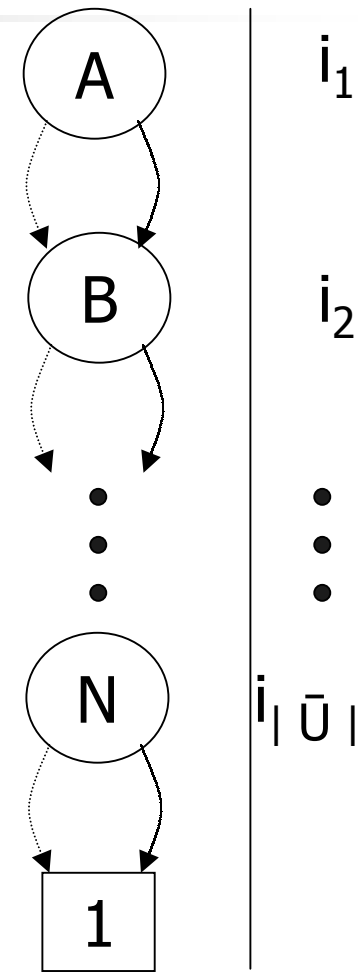


Newly Violated Clauses

- Given:
 - Subset U of violated clauses
- Each set S in the ZDD containing $u \in U$ must be removed
 - This branch cannot yield satisfiability
- Efficient implementation in terms of ZDD ops
 - Form the ZDD containing all possible subsets of \bar{U} : the set-complement to U
 - Intersect this with the original **front**

Newly Violated Clauses

- Build the ZDD containing all subsets of \bar{U}
 - For each element in \bar{U}
 - add a **don't care** node at that level
 - Size is $O(\bar{U})$
 - Exponential compression in this simple case





Newly Satisfied Clauses

- Given:
 - Set F of newly-satisfied clauses
- If $f \in F$ is in some subset of the **front**
 - It has now been satisfied
 - Any occurrence of f in the ZDD must be removed
- Implementation
 - The ZDD Existential Abstraction operation



Newly Activated Clauses

- Given:
 - Set A of activated clauses
- Each $a \in A$ must be added to every set in the **front**
- Implementation:
 - The ZDD Cartesian Product operation



Pseudocode

```
Front  $\leftarrow$  1      # assign  $\{\emptyset\}$  to front
foreach  $v \in$  Vars
    Front2  $\leftarrow$  Front
    Update(Front,  $v \leftarrow$  1)
    Update(Front2,  $v \leftarrow$  0)
    Front  $\leftarrow$  Front  $\cup_s$  Front2
if Front == 0 return Unsatisfiable
if Front == 1 return Satisfiable
```




Pseudocode

Update(ZDD Z , $v \leftarrow \text{value}$)

Find the set U of violated clauses

$$Z \leftarrow Z \cap \mathbf{2}^{\sim U}$$

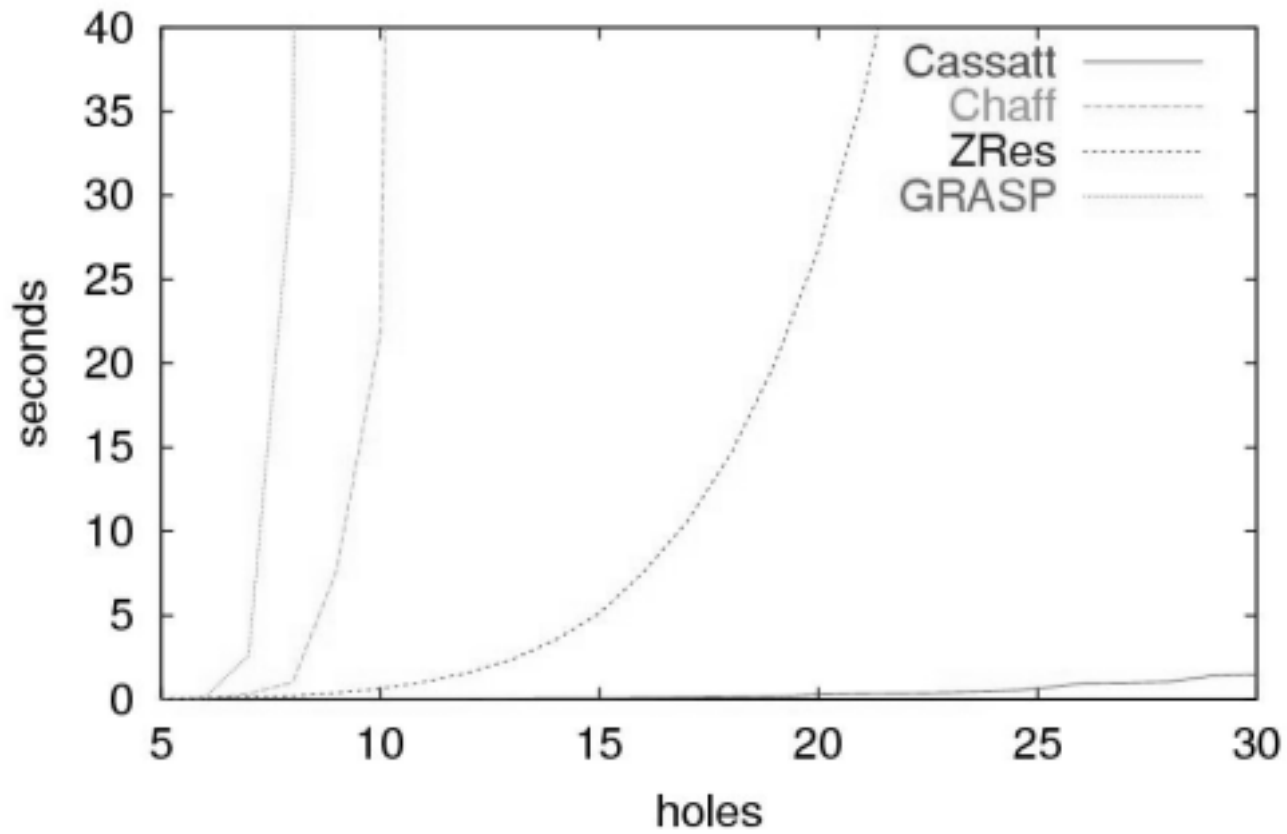
Find the set F of satisfied clauses

$$Z \leftarrow \text{ExistentialAbstract}(Z, F)$$

Find the set A of activated clauses

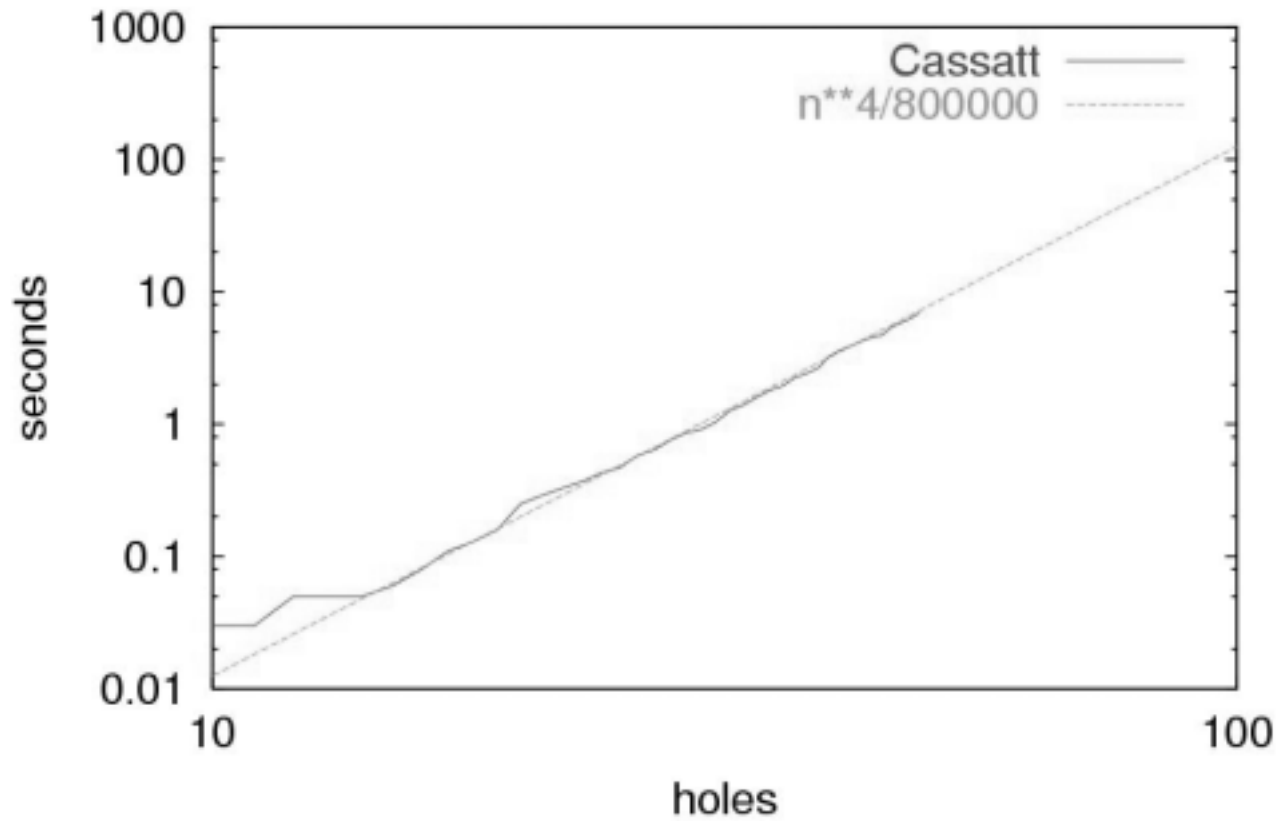
$$Z \leftarrow \text{CartesianProduct}(Z, A)$$

Results





Results





Summary of Results

- Proposed a novel algorithm for SAT
 - BFS with compression
 - Efficiency is due to exponential compression via ZDDs
- Implementation and empirical results
 - Solves pigeon-hole instances in poly-time
 - Outperforms Zres of Simon and Chatalic
 - Beats best DLL solvers on Urquhart instances
 - not better than Zres
 - Reasonable but not stellar performance on DIMACS benchmarks



Future Work

- Improved efficiency via Boolean Constraint Propagation
 - BCP is a part of all leading-edge SAT solvers
- Exploring the effects of clause and variable ordering on memory/runtime
- Implications of Cassatt in terms of proof systems



Questions?
