Impact of Community Structure on SAT Solver Performance*

by

Zack Newsham[1], Vijay Ganesh[1],

Sebastian Fischmeister[1], Gilles Audemard[2], and Laurent Simon[3]

[1]University of Waterloo, [2]University of Artois and [3]University of Bordeaux

*Best Student Paper Award at SAT 2014, Vienna, Austria.

Problem Statement

Why are CDCL Solvers efficient for Industrial Instances

- Boolean SAT/SMT Solvers are key tools in many approaches to software engineering
- Conflict-driven clause learning (CDCL) Boolean SAT solvers are remarkably efficient for large industrial instances
- This is true for industrial instances from a diverse set of applications
- These instances may have tens of millions of variables and clauses

Problem Statement

Why are CDCL Solvers efficient for Industrial Instances

- While there is more than 50 years of research on CDCL solvers, only 4 techniques are universally key to their efficiency [Sakallah]

1. Conflict-driven clause-learning with back-jumping (Marques-Silva et al.//GRASP)
2. VSIDS branching heuristic (Moskewicz et al.//zCHAFF)
3. Conflict clause deletion (Goldberg et al.//BERKMIN)
4. Restarts with phase-saving (Gomes et al.)

- Why is this so?
- This is surprising and deserves an explanation, especially given that Boolean satisfiability is NP-complete. More precisely,
  - Why are large industrial SAT instances “easy” to solve?
  - Why are the above-mentioned techniques universally effective?

Motivation

Why are CDCL Solvers efficient for Industrial Instances

- Resolving the mystery of “efficiency of CDCL solvers” is like doing physics, as opposed to benchmarking-based technique development
- This question has stumped both theoreticians and practitioners alike
- May lead to better solvers (we already have a preliminary result)
- Motivation for better analysis tools to analyze CDCL-like backtracking algorithms (We have already built visualization and evolution tools to view the runtime behavior of the SAT solver)

Previous Work

Why are CDCL Solvers efficient for Industrial Instances

- Best theoretical result about CDCL solvers to-date
  - The best result to-date states that conflict-driven clause-learning is a proof system that polynomially simulate general resolution [PD09, Beame et al.]
  - Doesn't explain the remarkable efficiency of CDCL on industrial instances
- Results on the practical front to-date
  - Phase-change at clause-var ratio of 4.25 for randomly-generated instances
  - Empirically observed *only* for randomly-generated instances
  - Doesn't explain the power of CDCL solvers for industrial instances

Structure in Industrial Instances

Why are CDCL SAT Solvers efficient for Industrial Instances

- CDCL solvers exploit structure Inherent to SAT instances
- Relevant questions

1. What structure of industrial instances do solvers exploit?
2. What evidence connects so-called structure and solver performance?
3. How does the solver go about exploiting this structure?

- In this talk we discuss an answer to Questions 1 and 2. (Note that for an answer to be useful, it doesn't have to precisely demarcate the line between "easy" and "hard" instances.)

Community Structure and SAT Solver Performance
Our Results and Take-home Message

- Take-home Message
  - Community structure (whose quality is measured using metric called Q) of SAT instances strongly affect solver performance
  - Informally
    - View SAT instances as variable-incidence graphs
    - A community is sub-graph that has more internal edges than outgoing ones
    - A community structure characterizes how “well clustered” a graph is
- Empirical Results
  - Result #1: Strong correlation between community structure and LBD (Literal Block Distance) in Glucose solver
  - Result #2: Hard random instances have low Q ($0.05 \leq Q \leq 0.13$)
  - Result #3: Number of communities and Q of SAT instances are more predictive of CDCL solver performance than other measures we considered

Community Structure and SAT Solver Performance
Graphs and their Community Structure

- Our result (also by Ansotegui, Levy et al. [AL14])
  - **Community structure, specifically, modularity (Q) and number of communities correlate with CDCL solver performance**

- Informal definition of community in a graph
  - A sub-graph tightly connected internally, but weakly connected externally

- Industrial SAT instances community structure (Ansotegui, Levy et al.[AL12, AL13])

Structure in Industrial Instances

Previous Approaches to Characterizing Structure

- Number of variables (N_v) and clauses (N_c), and functions over N_v and N_c
  - For example, clause-variable ratio
  - Lack explanatory and predictive power for industrial SAT instances
  - I mentioned this point earlier, but it is worth repeating in the context of characterizing input structure

- Large successful predictive model by Xu, Hoos et al.
  - Basis for a machine learning based predictor [XHHL08]

- Wanted
  - Small predictive set of features that forms the basis for a complete explanation (we don't have it yet)

Rest of the Talk

- Motivation and problem statement
- Motivated need for precisely capturing structure of industrial instances
- Take-home message and outline of results
- Brief description of conflict-driven clause-learning SAT solvers
- Outline of our ultimate goal
- Definition of community structure in graphs
- Results
- Strengths and weaknesses of our results, leading to future directions

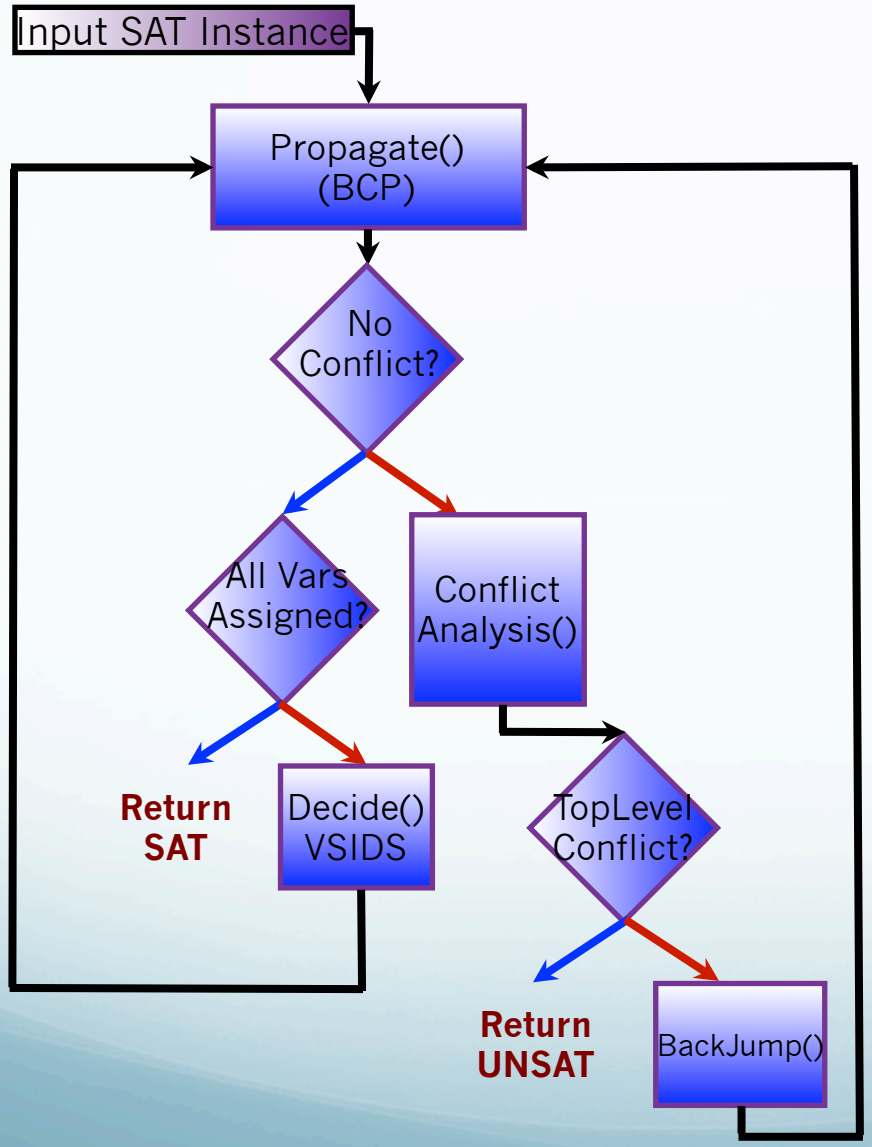
DPLL SAT Solver Architecture
The Basic Solver

```
DPLL(Θcnf, assign) {

Propagate unit clauses;

If conflict return FALSE;

If (complete assignment) return TRUE;

Decide(); //pick decision variable x


return
        DPLL(Θcnf | x=0, assign[x=0])
    || DPLL(Θcnf | x=1, assign[x=1]);

}
```

- **Propagate**
  - Boolean Constant Propagation
  - Unit propagation
  - Propagate inferences due to unit clauses
- **Detect Conflict:**
  - Partial non-satisfying assignment
- **Decide (Branch):**
  - Pick a variable & assign some value
- **Backtracking:**
  - Implicitly done through the recursion

Modern CDCL SAT Solver Architecture

Key Steps and Data-structures



Key steps

- Decide()
- Propagate()
- Conflict analysis and learning()
- Backjump()
- Forget()
- Restart()

CDCL: Conflict-Driven Clause-Learning

- Conflict analysis is a key step
- Learn a conflict clause (CC)
- Prunes the search space

Decide: VSIDS branching heuristic

- Crucial to performance
- Maintain a score for each var
- Additively bump recent CC vars
- Multiplicative decay all

Key data-structures (State)

- Stack of partial assignments
- Input clause database
- Conflict clause database
- Conflict analysis graph
- Decision level (DL) of a variable

Modularity (Q-factor) and Communities in Graphs
Community Structure in Graphs

- Modularity of Q lies between 0 and 1
- Q measures quality
  - Higher Q implies "good community structure", i.e., *highly separable communities*
  - Lower Q implies "bad community structure", i.e., *one giant hairy ball*
- $Q_{simple} = \frac{\text{Number of edges inside communities}}{\text{Total number of edges}}$
- Problem
  - The community containing entire graph has $Q_{simple} = 1$

Modularity (Q-factor) and Communities in Graphs
Community Structure in Graphs

- Problem
  - The community containing the entire graph has $Q_{simple} = 1$
- Solution
  - For all community structures over given graph G
    - Compute Q_{simple}
    - Randomize G and compute $Q_{randomized}$
    - Compute $Q = Q_{simple} - Q_{randomized}$
  - Q of graph G = Optimal($Q_{simple} - Q_{randomized}$)
- The modularity (Q) is the optimal structure furthest from a random-looking version of G

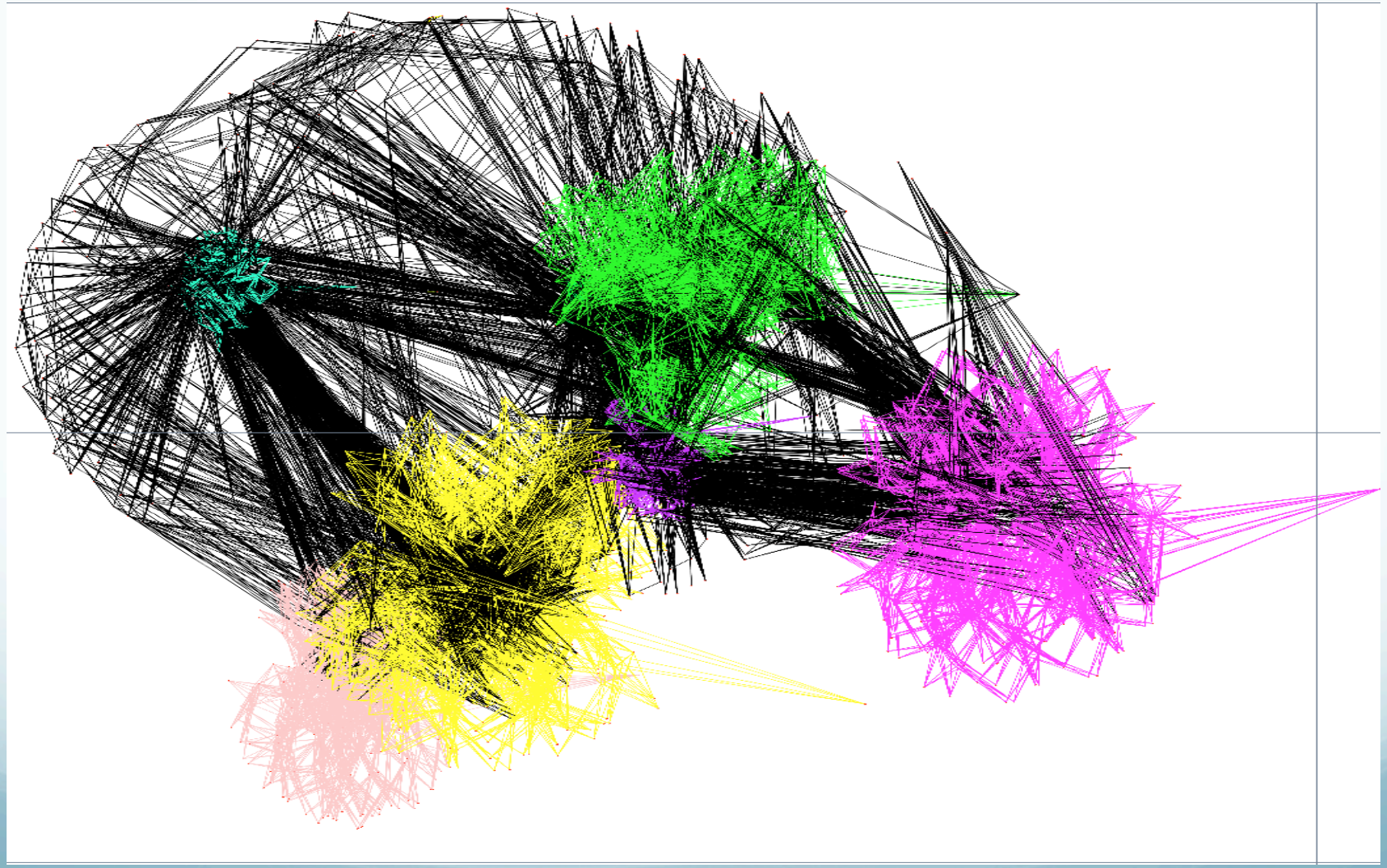
A Note on Community Structure in Graphs

Applications

- Community structure [GN03,CNM04,OL13] is used to study all kinds of complex networks, such as,
  - Social networks, e.g., Facebook
  - Internet
  - Protein networks
  - Neural network of the human brain
  - Citation graphs
  - Business networks
  - Populations
  - And more recently the graph of logical formulas

Community Structure in Graphs

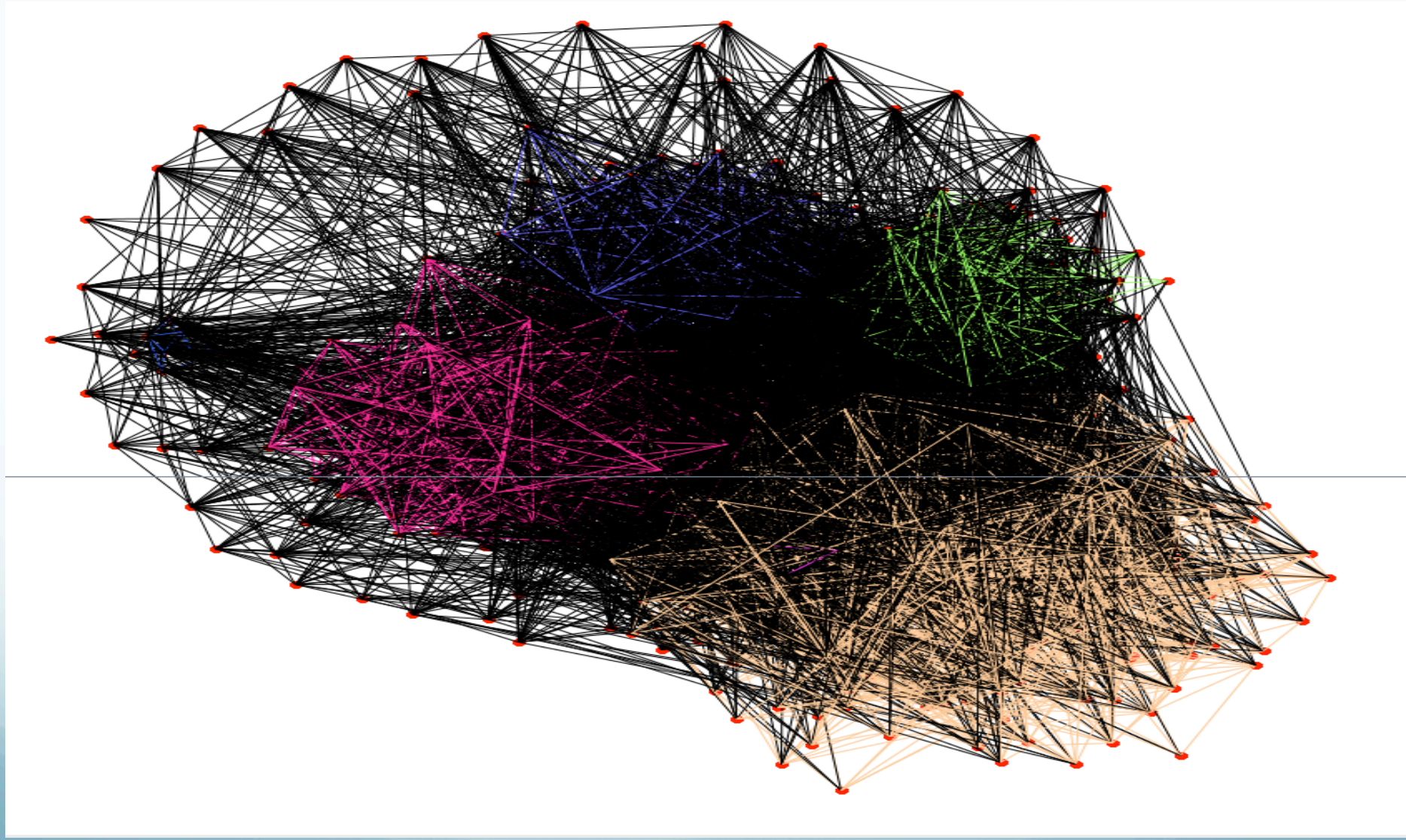
Variable-incidence Graph of Non-random Formula



SOURCE: mrpp example from SAT 2013 competition viewed using our SATGraf tool

Community Structure in Graphs

Variable-incidence Graph of Randomly-generated Formula



SOURCE: unif-k3-r4.267-v421-c1796-S4839562527790587617 **randomly-generated** example from SAT 2013 competition

Modularity (Q-factor) and Communities in Graphs
Community Structure in Graphs

- How to compute community structure?
- The decision version of the Q maximization problem is NP-complete [Brandes et al., 2006]
- Many efficient *approximate* algorithms proposed, e.g., [CNM04] and [0L13]
- We use the above two algorithms for our experiments
- We got similar results with both algorithms, increasing confidence in our results
- There are other methods to compute community structure based on graph centrality

Literal Block Distance (LBD) and Communities
Experiment #1: Hypothesis and Definitions

Hypothesis tested

- The number of communities in a conflict clause correlates strongly with its LBD measure

What is LBD? (Introduced first in Glucose solver [AS09])

- LBD measure M of a learnt clause C is a rank based on the number N of distinct decision levels the vars in C belong to
- The lower the value of N, the higher the rank M
- LBD is a powerful measure of the *utility of a conflict clause*

Literal Block Distance (LBD) and Communities
Experiment #1: Hypothesis and Definitions

Clause deletion

- Clause deletion is integral to efficiency of modern solvers

- *Sans* clause deletion, rate of conflict clause production may cause solvers to quickly run out of memory

Which clauses to delete? LBD to the rescue

- Periodically delete conflict clauses with bad LBD rank

- As we will see, clauses with bad LBD rank are shared by many communities

Literal Block Distance (LBD) and Communities
Experiment #1: Intuition

The number of communities in a conflict clause

- The number of communities N in a conflict clause C is the number of distinct communities the variable in C belong to

Intuition behind the hypothesis

- High-quality conflict clauses tend to span very few communities, i.e., the number N of different communities their variables belong to is small
- High-quality conflict clauses are likely to cause more propagation per decision variable, and hence likely to have low LBD
- LBD picks out high-quality conflict clauses

Literal Block Distance (LBD) and Communities
Experiment #1 Setup

- Instances considered
  - 189 SAT 2013 industrial category instances out of 300
  - We were able to compute communities only for these 189
  - The rest caused memory-out

- Step 1 of the experiments

  - For each of the 189 instances in our benchmark compute
    - Community structure
    - The number of communities a learnt clause belongs to
    - LBD of every learnt clause (considered only the first 20,000 learnt clauses due to resource constraints)

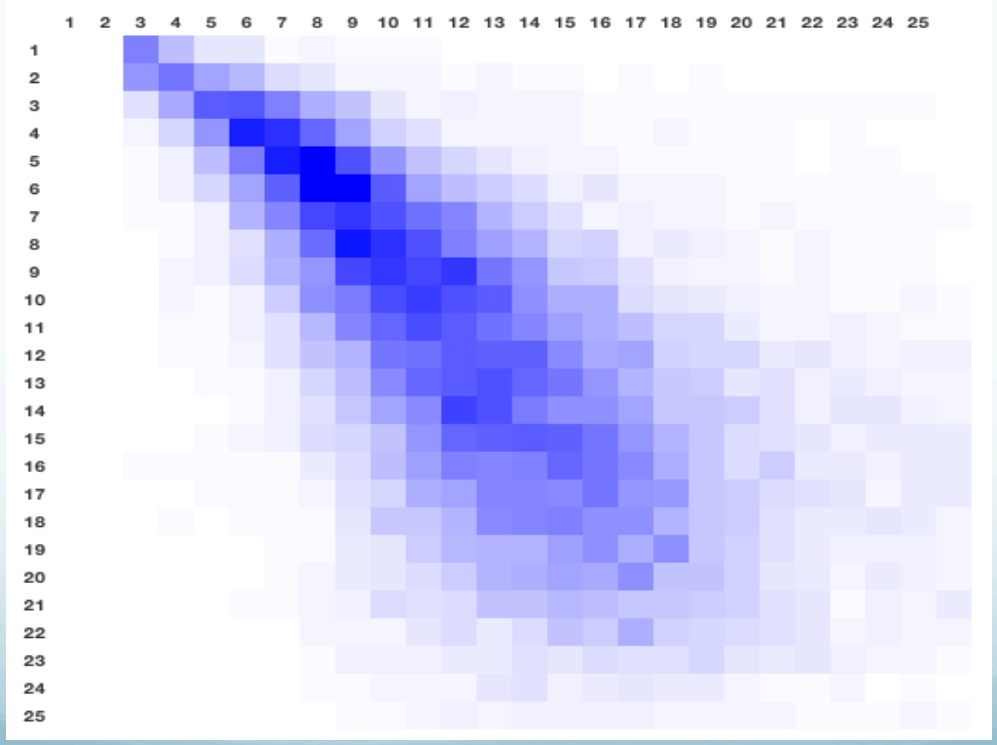
Literal Block Distance (LBD) and Communities Experiments Performed (#1)

- Step 2 of the experiments
  - LBD of every learnt clause L considered was correlated with the number of communities L belongs to
  - Thousands of data points over 189 instances
  - Correlate LBD and num-of-communities using heatmaps
    - Heatmap of LBD and communities of learnt clauses
    - Otherwise difficult to correlate thousands of data points over hundreds of instances
    - One heatmap per SAT instance

Literal Block Distance (LBD) and Communities
Experiment #1 Results and Interpretation

Result #1

Most industrial instances have a very strong (diagonal) relationship between LBD and communities



Community Structure and Random Instances
Experiment #2: Hypothesis and Definitions

Hypothesis tested

- Is there a range of Q-factor values for randomly-generated instances that are hard for CDCL SAT solvers, irrespective of number of variables/ clauses

- Are randomly-generated instances outside this range uniformly easy

Community Structure and Solver Running Time
Experiment #2 Setup

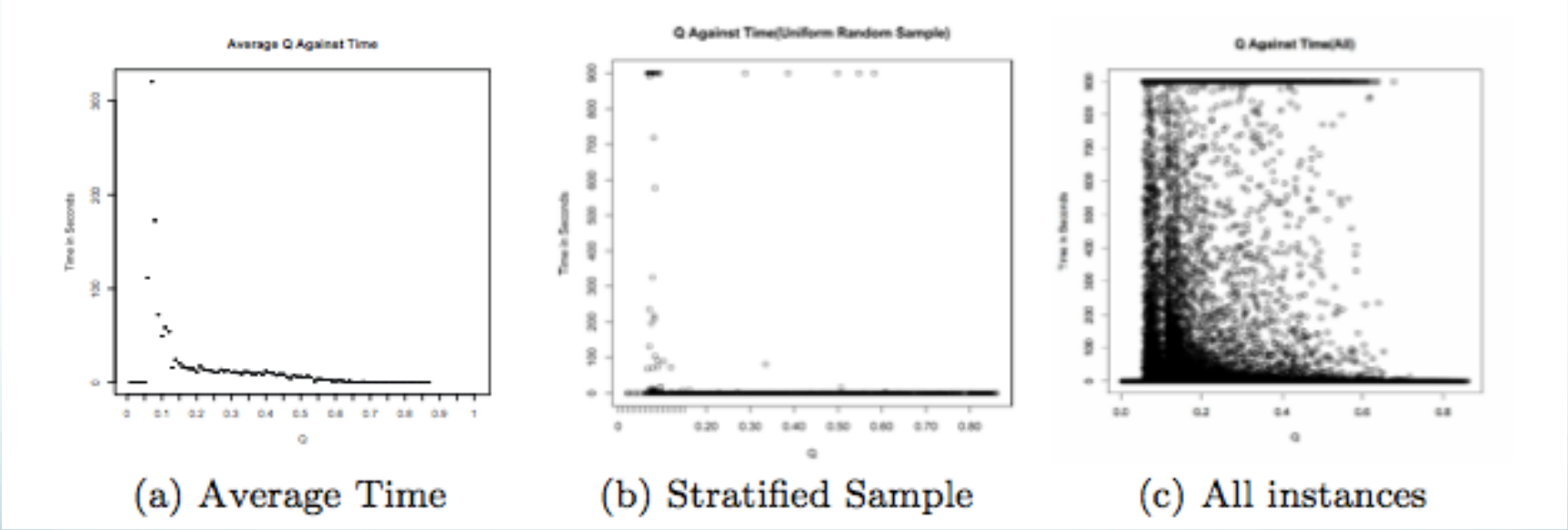
- Randomly generated 550,000 SAT instances for the experiment
  - Varied N_v between 500 to 2000 in increments of 100
  - Varied N_{cl} between 2000 and 10000 in increments of 1000
  - Varied target Q between 0 and 1 in increments of 0.01
  - Varied “Num of communities” between 20 and 400 in increments of 20
- Experiments using MiniSAT
  - Timeout of 900 seconds per run
  - Run solver on inputs in a random order
  - Average the running time over several runs

Community Structure and Solver Running Time Experiments Performed (#2)

- Plotted Q against time
- Noticed significant increase in execution time when $0.05 \leq Q \leq 0.13$
- Also recomputed the results using a stratified sample
  - Used due to high number of instances in target range
  - Randomly sample the data taking 250 results from each 0.1 range of Q between 0 and 0.9
  - Almost same result $0.05 \leq Q \leq 0.12$

Community Structure and Solver Running Time Results and Interpretation (#2)

- Huge increase in running time for randomly-generated instances with $0.05 \leq Q \leq 0.13$



Community Structure and Solver Running Time
Experiment #3: Hypothesis and Definitions

Hypothesis tested

- Are the community structure (Q factor) and number of communities better correlated with running time of a CDCL SAT solver than traditional metrics

- Is the correlation better for industrial instances than randomly-generated or hand-crafted ones

Community Structure and Solver Running Time
Experiment #3 Setup

Instances for the experiment

- Approx. 800 instances from SAT 2013 competition. For the remaining we couldn't compute community structure due to resource constraints

Used OL algorithm to compute community structure for the 800 instances

- Much faster and scales better
- Approximates the community structure

All experimental results are for MiniSAT

- Obtained running time of solver from SAT 2013 competition website

Used statistical tool R to perform standard linear regression

Community Structure and Solver Running Time Experiments Performed (#3)

- Performed linear regression on the solver running time data using statistical tool R twice
  - Once with community structure metrics
  - And once without community structure metrics
- Compared the adjusted R^2 (variability) from both experiments
  - Variability measures how "far off" are the model's prediction from given data
- R tells which of these two models has lower variability, i.e., is a better predictor

Community Structure and Solver Running Time
Experiment #3 Results and Interpretation

- R tells that the model with community structure metrics is a better predictor of running time than traditional metrics like # of clauses and variables
- The model is even better if we only consider industrial instances
- Q was included in all but one significant factors with 99.9% confidence

Factor	Estimate	Std. Error	t value	$Pr(>\|t\|)$	Sig
$\|CO\|$	-1.237e+00	3.202e-01	-3.864	0.000121	***
$\|CL\| \odot Q \odot QCOR$	-4.226e+02	1.207e+02	-3.500	0.000492	***
$\|CL\| \odot Q$	-2.137e+02	6.136e+01	-3.483	0.000523	***
$\|CL\| \odot Q \odot \|CO\| \odot QCOR \odot VCLR$	-1.177e+03	3.461e+02	-3.402	0.000702	***
$\|CL\| \odot Q \odot \|CO\|$	-6.024e+02	1.774e+02	-3.396	0.000719	***
$Q \odot QCOR$	3.415e+02	1.023e+02	3.339	0.000881	***
Q	1.726e+02	5.200e+01	3.318	0.000947	***
$Q \odot \|CO\| \odot QCOR$	9.451e+02	2.927e+02	3.229	0.001292	**

Impact of Community Structure and Solver Running Time
Scope for Improvement

- Consider different graph representations for community detection
- Try experiments on more solvers
- Can we construct a highly predictive model
- Compare community structure-based model against graph-width based models
- Community structure of SAT vs. UNSAT cases
- Try different random generation strategy and larger instances
- Hierarchical communities?
- Other solver measures like memory usage, number/rate of conflicts generated,…
- Build visualization tools to confirm/refute hypotheses
- Dynamically track how community structure of learnt clauses evolve

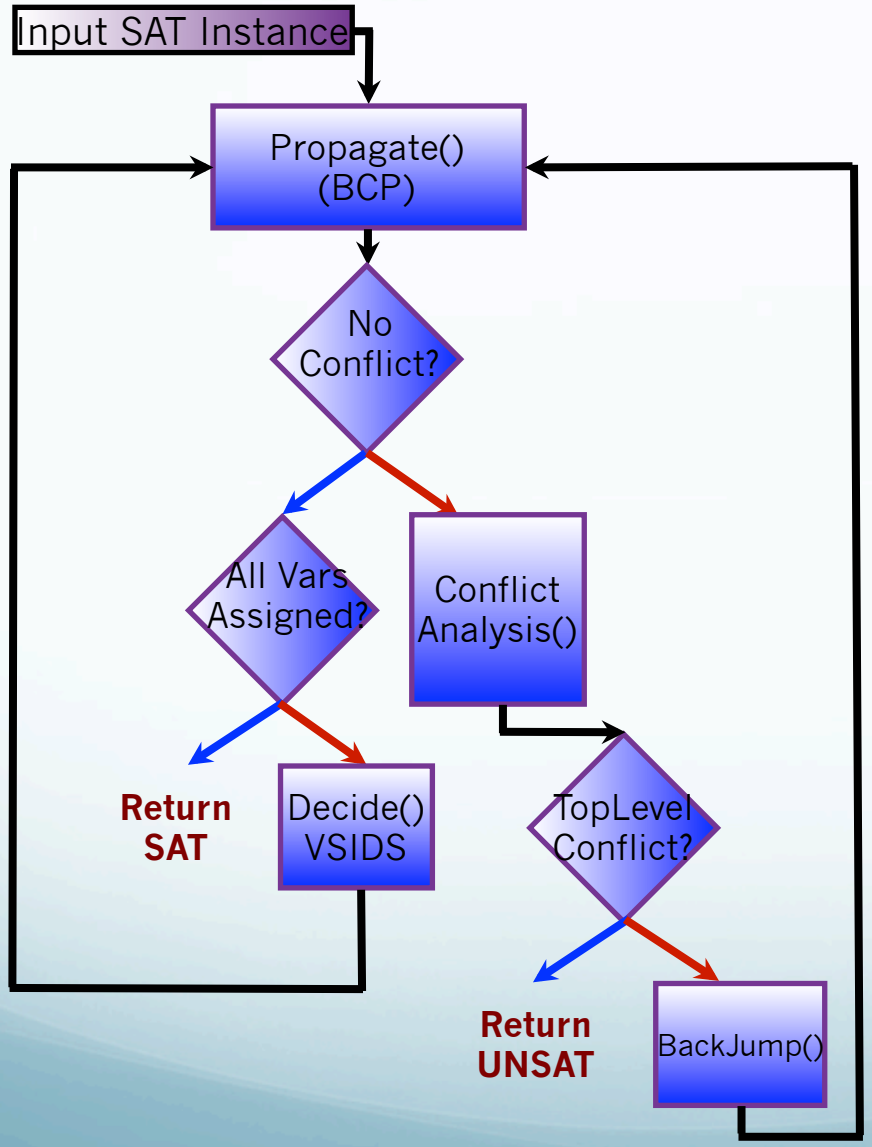
Community Structure and SAT Solver Performance

Conclusions and Future Work

- Take-home Message
  - Community structure of SAT instances strongly affect solver performance
- Result #1
  - Strong correlation between community structure and LBD (Literal Block Distance) in Glucose
- Result #2
  - Hard random instances have low Q ($0.05 \leq Q \leq 0.13$)
- Result #3
  - Number of communities and Q of SAT instances are more predictive of MiniSAT performance than other measures
- Result #4 (in progress)
  - VSIDS correlates strongly with graph centrality with exponential smoothing average. VSIDS picks out variables central to communities. New VSIDS based on this observation with promising results

Modern CDCL SAT Solver Architecture

Key Steps and Data-structures



Key steps

- Decide()
- Propagate()
- Conflict analysis and learning()
- Backjump()
- Forget()
- Restart()

CDCL: Conflict-Driven Clause-Learning

- Conflict analysis is a key step
- Learn a conflict clause (CC)
- Prunes the search space

Decide: VSIDS branching heuristic

- Crucial to performance
- Maintain a score for each var
- Additively bump recent CC vars
- Multiplicative decay all

Key data-structures (State)

- Stack of partial assignments
- Input clause database
- Conflict clause database
- Conflict analysis graph
- Decision level (DL) of a variable

A Model for CDCL Solvers

Why are CDCL Solvers efficient for Industrial Instances

1. A mathematical model (algorithm) with an inductive and deductive engine with feedback

    - First component is the VSIDS-like branching heuristic (induction)
    - Second component is the conflict-clause analysis method (deduction)
    - CDCL solver is a proof system with feedback between these two components

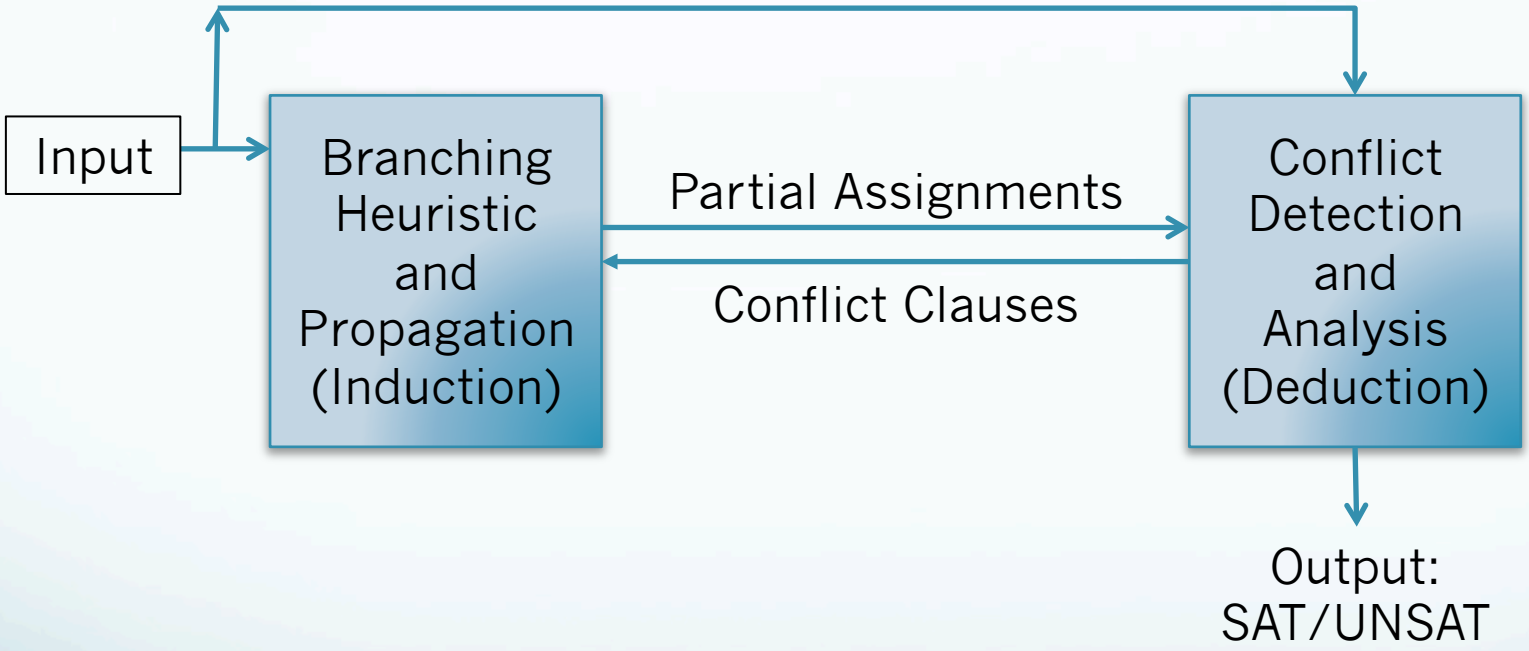
2. A precise mathematical description of the class of input instances on which the CDCL algorithm performs well (the focus of this talk)

3. Possibly a theorem connecting 1) and 2):

"if input instances have certain structure described precisely in graph-theoretic terms, then the above described algorithm will terminate in time polynomial in the size of the input (and possibly exponential in the parameters of the graph of the input) in a parameterized complexity-theoretic setting"

A Model for CDCL Solvers

CDCL Solvers: Induction and Deduction with Feedback



Research Results and Future Directions

Selected Results to-date

- Practice
  - STP solver (CAV 2007)
    - Symbolic-execution based testing and analysis (CCS 2006)
    - Dominant solver design for theories over bit-vectors (1st in 2006/2010, 2nd in 2011/2014 SMTCOMP)
  - String solvers, HAMPI (TOSEM 2012, CAV 2011, ISSTA 2009) and Z3-str (FSE 2013), and their applications to computer security
  - Automated taint-based white-box fuzz testing (ICSE 2009)
  - Automated input rectification (ICSE 2012)
- Theory
  - Boundary between decidable and undecidable fragment of word equations (HVC 2012)

Future directions

- Explaining the power of Boolean SAT solvers for industrial applications (e.g., VSIDS branching heuristic and timed Google PageRank)
- Theory and practice of solving the satisfiability problem of theories of strings
- Provers based on integration of induction and deduction with feedback
- Attack-resistance programs

Impact of Community Structure on CDCL SAT Solver Performance

Questions