On the Modularity of Industrial SAT Instances

Carlos Ansótegui
DIEI, UdL, Lleida, Spain

Jordi Levy
IIIA, CSIC, Barcelona, Spain

CCIA'11, Lleida, Spain

SAT

- SAT is a central problem in computer science and AI with theoretical and **practical** applications
- The problem is NP-complete
- State-of-the-art solvers (heuristics, backjumping, learning, restarts, ...) are of practical use with real-world SAT instances
- SAT competitions:
  good solvers for random SAT inst. are bad for industrial inst. and vice versa
- **Objective:** Design solvers that perform well on **real-world** SAT instances

- SAT is a central problem in computer science and AI with theoretical and **practical** applications
- The problem is NP-complete
- State-of-the-art solvers (heuristics, backjumping, learning, restarts, . . .) are of practical use with real-world SAT instances
- SAT competitions:
  good solvers for random SAT inst. are bad for industrial inst. and vice versa
- **Objective:** Design solvers that perform well on **real-world** SAT instances

What is a "real-world" SAT instance?

SAT Formulas as Graphs

Clause-Variable Incidence Graph:

Nodes: are variables **v** and clauses **c**

Edges: **v**—**c** if clause **c** contains variable **v** with weight $w = 1$

SAT Formulas as Graphs

Clause-Variable Incidence Graph:

Nodes: are variables **v** and clauses **c**

Edges: **v**—**c** if clause **c** contains variable **v** with weight $w = 1$

Arity of nodes: **v** number of occurrences of v

c size of c

SAT Formulas as Graphs

Clause-Variable Incidence Graph:

Nodes: are variables **v** and clauses **c**

Edges: **v**—**c** if clause **c** contains variable **v**

with weight $w = 1$

Variable Incidence Graph:

Nodes: are variables **v**

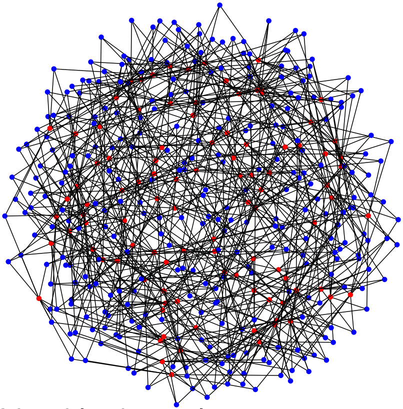
Edges: $\mathbf{v_1}$—$\mathbf{v_2}$ if some clause **c** contains variables $\mathbf{v_1}$ and $\mathbf{v_2}$

with weight $w = \frac{1}{\binom{|c|}{2}}$

the sum of the weights of the edges generated by a clause is one

What is the Structure of Industrial Instances?

Random 3-CNF
(in the fase transition point)

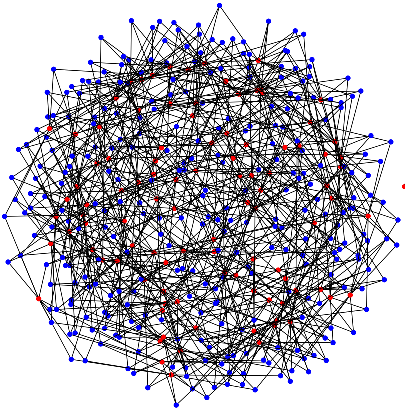


Variables have close to
4.25*3=12.75 occurrences

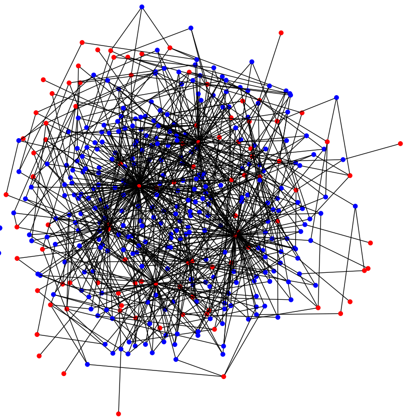
Real-World Instance

What is the Structure of Industrial Instances?

Random 3-CNF
(in the fase transition point)



Real-World Instance



Variables have close to 4.25*3=12.75 occurrences

Real-world instances contain **hubs**, are **scale-free** graphs!

Number of Occurrences of Variables

- Analyzed 100 instances of the SAT Race 2008
- $n = 25.693.792$ variables
- $\sum_{i=1}^{n} N(i) = 349.760.681$ occurrences
- $E\left[N(i)\right] = \sum_{i=1}^{n} N(i)/n = 13.6$ average number of occurrences

Number of Occurrences of Variables

- Analyzed 100 instances of the SAT Race 2008
- $n = 25.693.792$ variables
- $\sum_{i=1}^{n} N(i) = 349.760.681$ occurrences
- $E[N(i)] = \sum_{i=1}^{n} N(i)/n = 13.6$ average number of occurrences
- 90% of variables have less than this number of occurrences
  60% have 6 or less occurrences

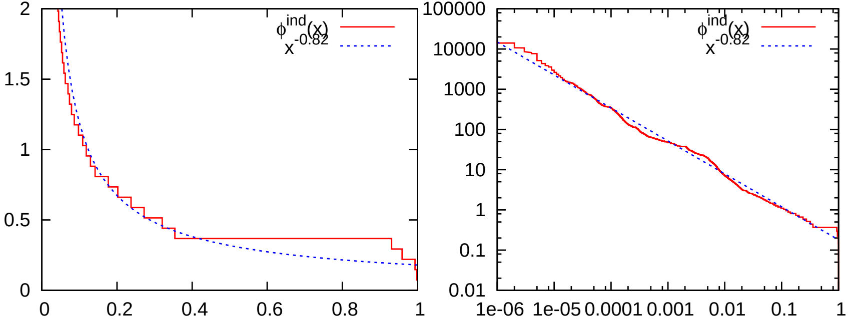
Number of Occurrences of Variables

- Analyzed 100 instances of the SAT Race 2008
- $n = 25.693.792$ variables
- $\sum_{i=1}^{n} N(i) = 349.760.681$ occurrences
- $E[N(i)] = \sum_{i=1}^{n} N(i)/n = 13.6$ average number of occurrences
- 90% of variables have less than this number of occurrences
  60% have 6 or less occurrences
- Let $N(i)$ = number of occurrences of the ith most frequent variable $(N(i) \geq N(i+1))$
- Estimate

$$\phi^{ind}(i/n) = \frac{n}{\sum_{j=1}^{n} N(j)} N(i)$$

- Estimate

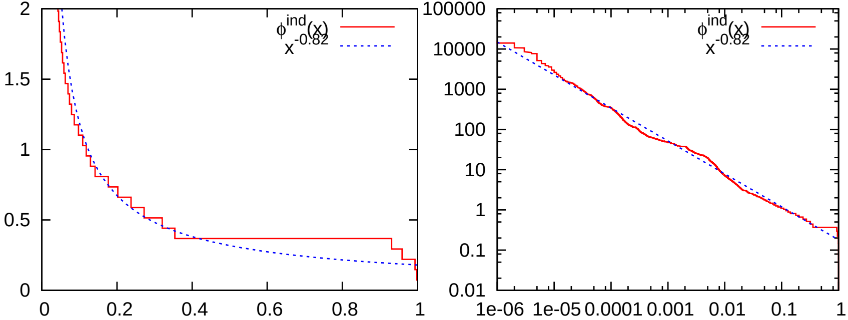
$$\phi^{ind}(i/n) = \frac{n}{\sum_{j=1}^{n} N(j)} N(i)$$



Number of Occurrences of Variables

- Estimate

$$\phi^{ind}(i/n) = \frac{n}{\sum_{j=1}^{n} N(j)} N(i)$$



We have $\phi^{ind}(x) \sim x^{-\beta}$ for $\beta \approx 0.82$

$P(ocurrences = k) \sim k^{-\alpha}$, **where** $\alpha = 1/\beta + 1 = 2.22$

Conclusions

"Everything should be made as simple as possible, but no simpler"

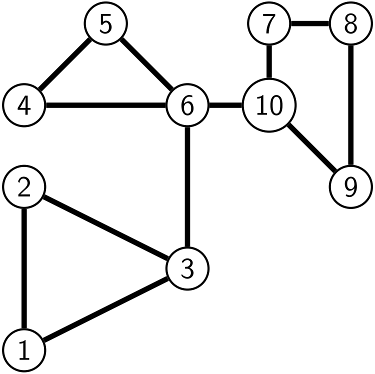
Aha! (over-simplification)

- We may think that variable hubs are backdoors, solvers instantiate them and solve the problem...

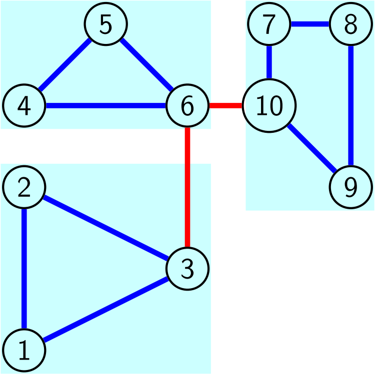
But, it is not so easy...

- Variable selection heuristics are not so simple, the consider how many times variables participate in conflicts (activity), and even some randomness...
- Given a random formula, added learned clauses make it scalefree!!!
- Scalefree structure is not lost by instantiations!!!

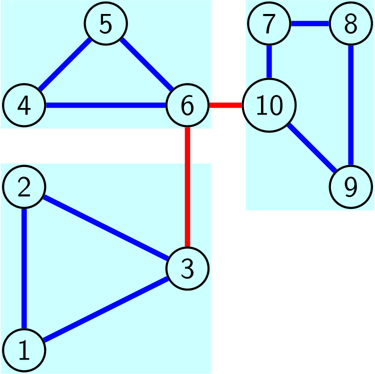
Clusters Detection in Graphs



Clusters Detection in Graphs

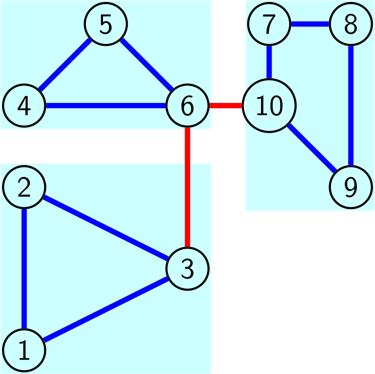


Clusters Detection in Graphs



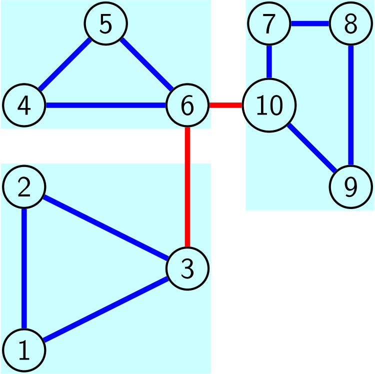
$$Q = \frac{inner\ \ edges}{total\ \ edges}$$

Clusters Detection in Graphs



$$Q = \frac{inner\ \ edges}{total\ \ edges} - \frac{expected\ \ inner\ \ edges}{total\ \ edges}$$

Clusters Detection in Graphs



$$Q = \frac{inner\ \ edges}{total\ \ edges} - \frac{expected\ \ inner\ \ edges}{total\ \ edges}$$

$$Q = \frac{10}{12} - \frac{7 \cdot \frac{7}{24} + 8 \cdot \frac{8}{24} + 9 \cdot \frac{9}{24}}{12} \approx 0.8333 - 0.3368 = 0.4965$$

Modularity

Modularity $Q \in [-1\ 1]$ measures **how good** is partition $P = \{P_i\}$

$$Q = \frac{inner\ edges}{total\ edges} - \frac{expected\ inner\ edges}{total\ edges}$$

For w-weighted graphs:

$$Q = \sum_{P_i \in P} \frac{\sum_{x,y \in P_i} w(x,y)}{\sum_{x,y \in V} w(x,y)} - \sum_{P_i \in P} \left(\frac{\sum_{x \in P_i} \deg(x)}{\sum_{x \in V} \deg(x)} \right)^2$$

where $\deg(x) = \sum_{y \in V} w(x,y)$

Modularity

Modularity $Q \in [-1\ 1]$ measures **how good** is partition $P = \{P_i\}$
For w-weighted graphs:

$$Q = \sum_{P_i \in P} \frac{\sum\limits_{x,y \in P_i} w(x,y)}{\sum\limits_{x,y \in V} w(x,y)} - \sum_{P_i \in P} \left(\frac{\sum\limits_{x \in P_i} \deg(x)}{\sum\limits_{x \in V} \deg(x)} \right)^2$$

where $\deg(x) = \sum_{y \in V} w(x,y)$
For (V_1, V_2)-bi-partite graphs:

$$Q = \sum_{P_i \in P} \frac{\sum\limits_{\substack{x \in P_i \cap V_1 \\ y \in P_i \cap V_2}} w(x,y)}{\sum\limits_{\substack{x \in V_1 \\ y \in V_2}} w(x,y)} - \sum_{P_i \in P} \frac{\sum\limits_{x \in P_i \cap V_1} \deg(x)}{\sum\limits_{x \in V_1} \deg(x)} \cdot \frac{\sum\limits_{y \in P_i \cap V_2} \deg(y)}{\sum\limits_{y \in V_2} \deg(y)}$$

Modularity

Modularity $Q \in [-1\ 1]$ measures **how good** is partition $P = \{P_i\}$
For w-weighted graphs:

$$Q = \sum_{P_i \in P} \frac{\sum\limits_{x,y \in P_i} w(x,y)}{\sum\limits_{x,y \in V} w(x,y)} - \sum_{P_i \in P} \left(\frac{\sum\limits_{x \in P_i} \deg(x)}{\sum\limits_{x \in V} \deg(x)} \right)^2$$

where $\deg(x) = \sum_{y \in V} w(x,y)$
For (V_1, V_2)-bi-partite graphs:

$$Q = \sum_{P_i \in P} \frac{\sum\limits_{\substack{x \in P_i \cap V_1 \\ y \in P_i \cap V_2}} w(x,y)}{\sum\limits_{\substack{x \in V_1 \\ y \in V_2}} w(x,y)} - \sum_{P_i \in P} \frac{\sum\limits_{x \in P_i \cap V_1} \deg(x)}{\sum\limits_{x \in V_1} \deg(x)} \cdot \frac{\sum\limits_{y \in P_i \cap V_2} \deg(y)}{\sum\limits_{y \in V_2} \deg(y)}$$

Maximaxing modularity is NP-complete

A Clustering Algorithm

Input: $G = (X, w)$
Output: a labelling for X

```
for x ∈ X do label[x] := x endfor
do
    changes := false
    for i ∈ |X| in random order do
        l := most_freq_label(i,neighbors(i))
        if l ≠ label[i] then
            changes := true
            label[i] := l
    endfor
while changes
return label
```

Modularity of SAT Race 2010

	Family (#inst.)	Variable Incid. Graph (VIG)							Clause-Variable Incid. Graph (CVIG)						
		time	IEF	IEF^e	Q	$\|P\|$	larg.	iter	time	IEF	IEF^e	Q	$\|P\|$	larg.	iter
cripto.	desgen(4)	7	0.89	0.01	**0.88**	517	0.01	34	15	0.77	0.00	**0.77**	2752	0.01	28
	md5gen(3)	6	0.61	0.00	**0.61**	7151	0.00	15	43	0.78	0.00	**0.78**	6934	0.00	32
	mizh(8)	2	1.00	1.00	0.00	21	1.00	4	52	0.78	0.10	**0.69**	4505	0.30	33
hard. ver.	ibm(4)	13	0.80	0.00	**0.80**	2681	0.01	10	51	0.79	0.00	**0.79**	8839	0.00	18
	manolios(16)	10	0.97	0.71	0.26	44	0.82	9	100	0.76	0.00	**0.76**	5115	0.01	35
	velev (10)	(1)	0.90	0.52	**0.38**	70	0.65	10							
mixed	anbulagan(8)	31	0.56	0.00	**0.56**	36745	0.00	11	88	0.87	0.00	**0.87**	13875	0.00	18
	bioinf(6)	8	0.80	0.18	**0.62**	172	0.38	4	37	0.82	0.37	**0.46**	2147	0.43	30
	diagnosis(4)	62	0.63	0.00	**0.63**	16372	0.02	14	198	0.74	0.00	**0.74**	31999	0.00	25
	grieu(3)	0	1.00	1.00	0.00	1.0	1.00	2	9	0.97	0.92	0.05	1.7	0.96	13
	jarvisalo(1)	0	0.59	0.01	**0.57**	260	0.05	8	0	0.73	0.00	**0.72**	294	0.01	12
	palacios(3)	134	0.96	0.62	**0.34**	2117	0.66	66	269	0.81	0.10	**0.72**	2853	0.17	49
soft. ver.	babic(2)	61	0.70	0.02	**0.68**	34033	0.08	54	379	0.72	0.01	**0.71**	61577	0.05	86
	bitverif(5)	78	0.97	0.58	**0.39**	83	0.64	96	363	0.82	0.02	**0.80**	9145	0.05	199
	fuhs(4)	8	0.93	0.76	0.17	379	0.79	25	5	0.71	0.06	**0.64**	5747	0.13	13
	nec(10)	207	0.99	0.87	0.12	372	0.93	22	882	0.80	0.02	**0.78**	31914	0.02	114

Modularity of Learned Clauses

Family	Variable Incid. Graph							Clause-Variable Incid. Graph						
	orig.	first 100 learned			all learned			orig	first 100 learned			all learned		
	Q	IEF	IEF^e	Q	IEF	IEF^e	Q	Q	IEF	IEF^e	Q	IEF	IEF^e	Q
desgen(1)	**0.89**	0.77	0.03	**0.74**	0.12	0.04	0.08	**0.77**	0.33	0.05	0.28	0.14	0.04	0.09
md5gen(1)	**0.61**	0.76	0.02	**0.74**	0.03	0.01	0.02	**0.78**	0.99	0.03	**0.96**	0.03	0.01	0.02
ibm(2)	**0.84**	0.70	0.11	**0.60**	0.48	0.01	**0.47**	**0.81**	0.70	0.11	**0.58**	0.29	0.00	0.29
manolios(10)	0.21	0.87	0.84	0.04	0.80	0.71	0.10	**0.76**	0.13	0.02	0.11	0.10	0.01	0.09
anbulagan(2)	**0.56**	0.18	0.02	0.16	0.02	0.01	0.01	**0.87**	0.10	0.01	0.10	0.06	0.02	0.04
bioinf(4)	**0.62**	0.57	0.12	**0.46**	0.42	0.36	0.06	**0.68**	0.77	0.08	**0.69**	0.24	0.09	0.15
grieu(1)	0.00	1.00	1.00	0.00	1.00	1.00	0.00	0.00	1.00	1.00	0.00	1.00	1.00	0.00
babic(2)	**0.68**	0.84	0.48	**0.36**	0.84	0.48	**0.36**	**0.71**	0.55	0.22	**0.33**	0.55	0.22	**0.33**
fuhs(1)	**0.66**	0.67	0.08	**0.59**	0.24	0.10	0.14	**0.71**	0.80	0.02	**0.78**	0.09	0.01	0.07
nec(10)	0.12	0.89	0.88	0.01	0.96	0.84	0.12	**0.78**	0.73	0.24	**0.49**	0.70	0.46	0.24

Other Clustering Algorithms

- methods based on simulated annealing
- based on spectral analysis of graphs
- Gready algorithms:
  - [Newman'04] [Clauset,Newman, Moore'04]
    Start with every node in a singleton partition
    Join two partitions maximizing ΔQ
    Stop when all joining decrease Q
  - [Raghavan, Albert, Kumara'07]
    Start with every node in a singleton partition
    Move a node to the partition where it has more neighbors
    Stop when every node is with most of its neighbors
  - [Blondel et al.'08]
    Start with every node in a singleton partition
    Move a node to the partition where ΔQ is maximized
    Stop when no movement improves Q