The Community Structure of SAT Formulas

Carlos Ansótegui[1], **Jesús Giráldez-Cru**[2], and Jordi Levy[2]

(1) DIEI, Univ. of Lleida
(2) **Artificial Intelligence Research Institute (IIIA-CSIC)**

SAT Conference 2012
June 20, 2012

Introduction

- SAT is a central problem in computer science and AI with theoretical and **practical** applications.
- SAT competitions: *good* solvers for random SAT instances are *bad* for industrial instances, and vice versa.
- SAT solvers efficiency solving industrial instances has undergone a great advance, mainly motivated by the introduction of lazy data-structures, learning mechanisms and activity-based heuristics.
- In parallel, there have been significant advances in our understanding of complex networks, e.g., the notion of small world as first model of complex networks alternatively to the classical random graph models.

Introduction

- SAT is a central problem in computer science and AI with theoretical and **practical** applications.
- SAT competitions: *good* solvers for random SAT instances are *bad* for industrial instances, and vice versa.
- SAT solvers efficiency solving industrial instances has undergone a great advance, mainly motivated by the introduction of lazy data-structures, learning mechanisms and activity-based heuristics.
- In parallel, there have been significant advances in our understanding of complex networks, e.g., the notion of small world as first model of complex networks alternatively to the classical random graph models.

Introduction

- We propose the use of **modularity** (of graphs) for detecting the **community structure** of SAT instances.
  - The notion of *community* is more general than the notion of *connected components*.
  - It allows the existence of (a few) connections between communities.

Introduction

- We propose the use of **modularity** (of graphs) for detecting the **community structure** of SAT instances.
  - The notion of *community* is more general than the notion of *connected components*.
  - It allows the existence of (a few) connections between communities.

Introduction

- We propose the use of **modularity** (of graphs) for detecting the **community structure** of SAT instances.
  - The notion of *community* is more general than the notion of *connected components*.
  - It allows the existence of (a few) connections between communities.

Do industrial SAT instances have community structure?

Introduction

- We propose the use of **modularity** (of graphs) for detecting the **community structure** of SAT instances.
  - The notion of *community* is more general than the notion of *connected components*.
  - It allows the existence of (a few) connections between communities.

Do industrial SAT instances have community structure?

What about random instances?

Introduction

- We propose the use of **modularity** (of graphs) for detecting the **community structure** of SAT instances.
  - The notion of *community* is more general than the notion of *connected components*.
  - It allows the existence of (a few) connections between communities.

Do industrial SAT instances have community structure?

What about random instances?

How does learning modify it?

SAT Formulas as Graphs

Clause-Variable Incidence Graph (CVIG):

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

Edges: **v**—**c** if clause **c** contains variable **v** with weight $w = \frac{1}{|c|}$

Clause-Variable Incidence Graph (CVIG):

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

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

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

Variable Incidence Graph (VIG):

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

SAT Formulas as Graphs

Clause-Variable Incidence Graph (CVIG):

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

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

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

Variable Incidence Graph (VIG):

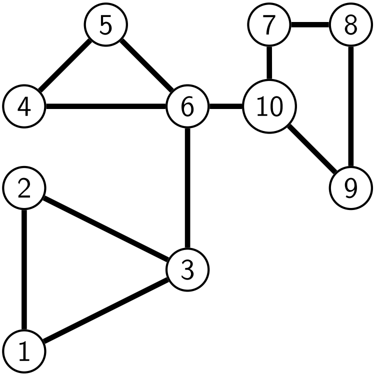
Nodes: are variables **v**

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

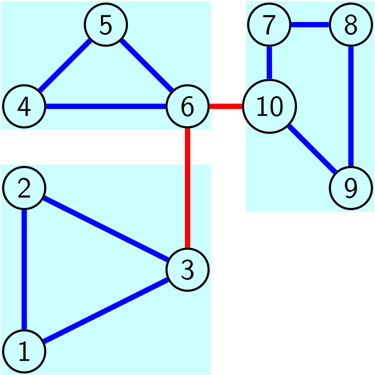
Partitions in Graphs



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

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

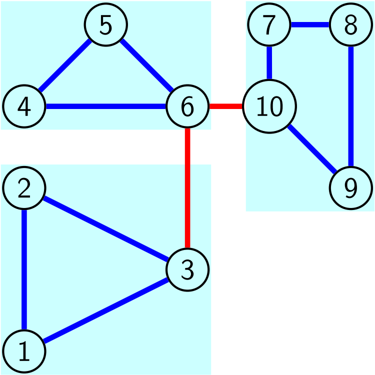
Partitions in Graphs



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

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

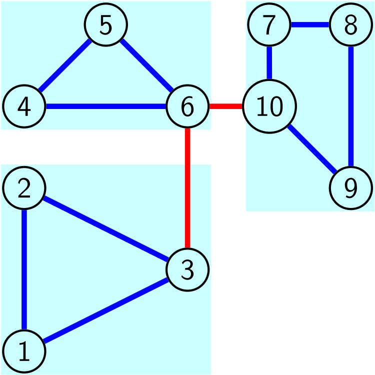
Partitions in Graphs



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

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

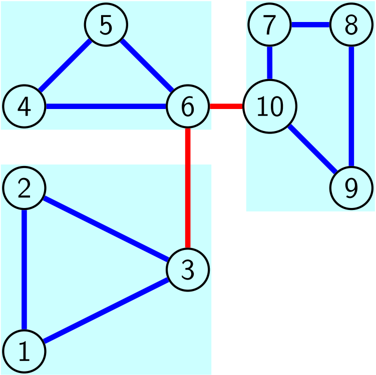
Partitions in Graphs



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

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

Partitions in Graphs



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

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

Definition of Modularity

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

For **weighted graphs** (Newman et al. (2004)):

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

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

For **bipartite graphs** (Barber (2007)):

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

Maximaxing modularity is NP-hard

Definition of Modularity

Modularity $Q \in [-1, \ 1]$ measures **how good** is partition $P = \{P_i\}$
For **weighted graphs** (Newman et al. (2004)):

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

where $\deg(x) = \sum_{y \in V} w(x,y)$
For **bipartite graphs** (Barber (2007)):

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

Maximaxing modularity is NP-hard

Definition of Modularity

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

For **weighted graphs** (Newman et al. (2004)):

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

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

For **bipartite graphs** (Barber (2007)):

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

Maximaxing modularity is NP-hard

Definition of Modularity

Modularity $Q \in [-1,\ 1]$ measures **how good** is partition $P = \{P_i\}$
For **weighted graphs** (Newman et al. (2004)):

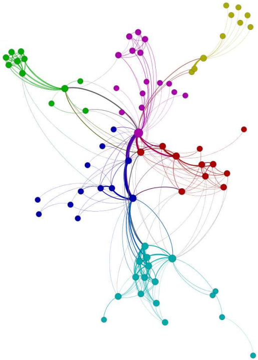
$$Q = \sum_{P_i \in P} \frac{\displaystyle\sum_{x,y \in P_i} w(x,y)}{\displaystyle\sum_{x,y \in V} w(x,y)} - \sum_{P_i \in P} \left(\frac{\displaystyle\sum_{x \in P_i} \deg(x)}{\displaystyle\sum_{x \in V} \deg(x)} \right)^2$$

where $\deg(x) = \sum_{y \in V} w(x,y)$
For **bipartite graphs** (Barber (2007)):

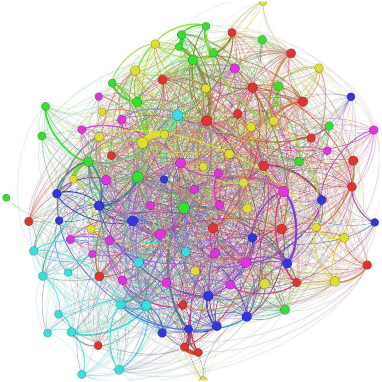
$$Q = \sum_{P_i \in P} \frac{\displaystyle\sum_{\substack{x \in P_i \cap V_1 \\ y \in P_i \cap V_2}} w(x,y)}{\displaystyle\sum_{\substack{x \in V_1 \\ y \in V_2}} w(x,y)} - \sum_{P_i \in P} \frac{\displaystyle\sum_{x \in P_i \cap V_1} \deg(x)}{\displaystyle\sum_{x \in V_1} \deg(x)} \cdot \frac{\displaystyle\sum_{y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

Maximaxing modularity is NP-hard

Example



$Q = 0.560$

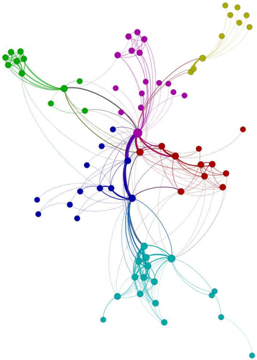


$Q = 0.097$

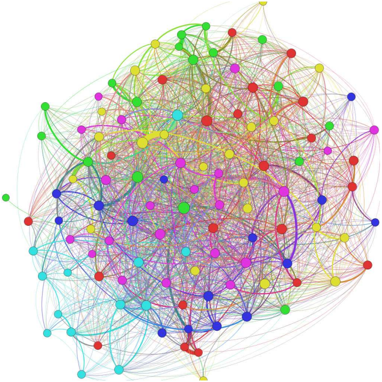
"In practice, Q values for networks showing a strong community structure range from 0.3 to 0.7, higher values are rare."

Newman et al. (2004).

Example



$Q = 0.560$



$Q = 0.097$

"In practice, Q values for networks showing a strong community structure range from 0.3 to 0.7, higher values are rare."

Newman et al. (2004).

Label Propagation Algorithm (LPA)

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

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

Raghavan et al. (2007).

Graph Folding Algorithm (GFA)

Input: Graph $G = (X, w)$
Output: a labelling L_1 for X
for $x \in X$ **do** $L_1[x] := x$; **endfor**
$L_2 :=$ **OneLevel**(G);
while $Mod(G, L_1) < Mod(G, L_2)$ **do**
  $L_1 := L_1 \circ L_2$;
  $G :=$ **Fold**(G, L_2);
  $L_2 :=$ **OneLevel**(G);
return L_1

function OneLevel(Graph $G = (X, w)$): Label L
  do
    $changes := \texttt{FALSE}$;
    foreach $i \in X$ **do**
      $bestInc := 0$;
      foreach $c \in \{c \mid \exists j . w(i,j) \neq 0 \wedge L[j] = 0\}$ **do**
        $inc := \sum_{L[j]=c} w(i,j) - arity(i) \cdot \sum_{L[j=c]} arity(j) / \sum_{j \in X} arity(j)$;
        if $inc > bestInc$ **then**
          $L[i] := c$;
          $bestInc := inc$;
          $changes := \texttt{TRUE}$;
  until $\neg changes$
  return L;

function Fold(Graph G_1, Label L): Graph G_2
  $X_2 = \{c \mid \forall i, j \in c . L[i] = L[j]\}$;
  $w_2(c_1, c_2) = \sum_{i \in c_1, j \in c_2} w_1(i,j)$;
  return $G_2 = (X_2, w_2)$;

Blondel et al. (2008).

Modularity of industrial SAT instances

- Modularity of the SAT instances used in the **2010 SAT Race Finals**.
    - 100 instances.
    - Grouped into 16 families.
    - Families classified as *cryptography*, *hardware verification*, *software verification*, and *mixed*, according to their application area.
    - All instances are *industrial* in the sense that their solubility has an industrial or practical application.
    - However, they are expected to show a **distinct nature** from random instances.

Modularity of industrial SAT instances

- Modularity of the SAT instances used in the **2010 SAT Race Finals**.
  - 100 instances.
  - Grouped into 16 families.
  - Families classified as *cryptography*, *hardware verification*, *software verification*, and *mixed*, according to their application area.
  - All instances are *industrial* in the sense that their solubility has an industrial or practical application.
  - However, they are expected to show a **distinct nature** from random instances.

Modularity of industrial SAT instances

	Family (#instanc.)	Variable IG LPA Q	LPA $\|P\|$	LPA larg.	LPA iter.	GFA Q	GFA $\|P\|$	GFA larg.	GFA iter.	Clause-Variable IG LPA Q	LPA $\|P\|$	LPA larg.	LPA iter.	Connect. Comp. $\|P\|$	larg.
cripto.	desgen(4)	**0.88**	532	0.8	36	**0.95**	97	2	37	**0.75**	3639	1	25	1	100
	md5gen(3)	**0.61**	7176	0.1	16	**0.88**	38	7	40	**0.78**	7904	0.1	42	1	100
	mizh(8)	0.00	33	99	6	**0.74**	30	9	33	**0.67**	5189	31	43	1	100
hard. ver.	ibm(4)	**0.81**	3017	0.6	9	**0.95**	723	4	32	**0.77**	19743	0.2	140	70	99
	manolios(16)	0.30	66	81	9	**0.89**	37	9	81	**0.76**	6080	1	26	1	100
	velev (10)	**0.47**	8	68	9	**0.69**	12	30	24	0.30	1476	77	31	1	100
mixed	anbulagan(8)	**0.55**	30902	0.1	11	**0.91**	90	2	43	**0.72**	46689	0.6	26	1	100
	bioinf(6)	**0.61**	87	44	3	**0.67**	60	17	22	**0.64**	94027	15	10	1	100
	diagnosis(4)	**0.61**	20279	0.7	15	**0.95**	68	3	43	**0.65**	85928	0.1	42	1	100
	grieu(3)	0	1	100	2	0.23	9	14	11	0	1	100	14	1	100
	jarvisalo(1)	**0.57**	260	5	8	**0.76**	19	9	26	**0.71**	336	1	11	1	100
	palacios(3)	0.14	1864	96	58	**0.93**	1802	6	13	**0.76**	2899	0.4	35	1	100
soft. ver.	babic(2)	**0.68**	34033	8	54	**0.90**	6944	10	141	**0.73**	59743	4	53	41	99
	bitverif(5)	**0.48**	3	57	4	**0.87**	24	6	29	**0.76**	33276	0.4	8	1	100
	fuhs(4)	0.02	18	99	43	**0.81**	43	7	30	**0.67**	12617	0.8	28	1	100
	nec(17)	0.07	107	96	31	**0.93**	65	14	124	**0.79**	23826	0.8	114	1	100
	post(2)	**0.36**	$3{\cdot}10^6$	53	54	**0.81**	$3{\cdot}10^6$	9	262	**0.72**	$3{\cdot}10^6$	6	49	224	99

Modularity of 2010 SAT Race instances.

Modularity after preprocessing formulas

Family		Orig. Form.	Preprocessed Formula				
			Modularity			Connect. Comp.	
		Q	Q	$\|P\|$	larg.	$\|P\|$	larg.
cripto.	desgen (4)	**0.951**	**0.929**	81	3.1	1	100
	md5gen (3)	**0.884**	**0.884**	18	8.5	1	100
	mizh (8)	**0.741**	**0.741**	18	9.5	1	100
hard. ver.	ibm (4)	**0.950**	**0.905**	26	6.1	1	100
	manolios (16)	**0.890**	**0.800**	16	14.9	1	100
	velev (10)	**0.689**	**0.687**	6	30.3	1	100
mixed	anbulagan (8)	**0.909**	**0.913**	47	5.1	1	100
	bioinf (6)	**0.673**	**0.657**	25	11.1	2	99.9
	diagnosis (4)	**0.952**	**0.950**	65	3.6	1	100
	grieu (3)	0.235	0.235	9	14.3	1	100
	jarvisalo (1)	**0.758**	**0.722**	11	13.1	1	100
	palacios (3)	**0.928**	**0.848**	17	10.76	1	100
soft. ver.	babic (2)	**0.901**	**0.875**	23	9.7	1	100
	bitverif (5)	**0.875**	**0.833**	19	7.3	1	100
	fuhs (4)	**0.805**	**0.743**	32	9.6	1	100
	nec (17)	**0.929**	**0.879**	37	10.3	1	100

Modularity after preprocessing the formula with *Satelite* preprocessor.

Modularity of Learnt Clauses

- Formulas are transformed during the proof or the satisfying assignment search.
  - Even if the original formula shows a community structure, could it be the case that this structure is quickly destroyed during the search process?
  - How new learnt clauses affect the community structure of the formula?
  - Even if the value of the modularity is not altered, could it be the case that communities are changed?

- Formulas are transformed during the proof or the satisfying assignment search.
  - Even if the original formula shows a community structure, could it be the case that this structure is quickly destroyed during the search process?
  - How new learnt clauses affect the community structure of the formula?
  - Even if the value of the modularity is not altered, could it be the case that communities are changed?

Modularity of Learnt Clauses

- Formulas are transformed during the proof or the satisfying assignment search.
  - Even if the original formula shows a community structure, could it be the case that this structure is quickly destroyed during the search process?
  - How new learnt clauses affect the community structure of the formula?
  - Even if the value of the modularity is not altered, could it be the case that communities are changed?

Modularity of Learnt Clauses

- Formulas are transformed during the proof or the satisfying assignment search.
  - Even if the original formula shows a community structure, could it be the case that this structure is quickly destroyed during the search process?
  - How new learnt clauses affect the community structure of the formula?
  - Even if the value of the modularity is not altered, could it be the case that communities are changed?

Modularity of resulting Formulas adding Learnt Clauses

Family		Orig. Form. Q	Orig. + Learnt Formula Modularity Q	$\|P\|$	larg.
cripto.	desgen (2)	**0.951**	**0.561**	53	13.0
	md5gen (3)	**0.884**	**0.838**	19	8.0
	mizh (1)	**0.741**	**0.705**	28	11.4
hard. ver.	ibm (3)	**0.950**	**0.912**	752	6.7
	manolios (14)	**0.890**	**0.776**	31	11.2
	velev (1)	**0.689**	**0.558**	6	30.1
mixed	anbulagan (4)	**0.909**	**0.876**	84	2.6
	bioinf (5)	**0.673**	0.287	32	58.7
	grieu (3)	0.235	0.085	6	35.0
	palacios (2)	**0.928**	**0.851**	2289	7.4
soft. ver.	babic (2)	**0.901**	**0.904**	6942	10.7
	fuhs (1)	**0.805**	**0.670**	24	7.6
	nec (15)	**0.929**	**0.936**	62	7.3

Modularity of the formula with learnt clauses included (using *PicoSAT*).

Modularity of Learnt Clauses using Original Partitions

Family	VIG orig.	VIG first 100	VIG all	CVIG orig	CVIG first 100	CVIG all
desgen (1)	**0.89**	**0.74**	0.08	**0.77**	0.28	0.09
md5gen (1)	**0.61**	**0.74**	0.02	**0.78**	**0.96**	0.02
ibm (2)	**0.84**	**0.60**	**0.47**	**0.81**	**0.58**	0.29
manolios (10)	0.21	0.04	0.10	**0.76**	0.11	0.09
anbulagan (2)	**0.56**	0.16	0.01	**0.87**	0.10	0.04
bioinf (4)	**0.62**	**0.46**	0.06	**0.68**	**0.69**	0.15
grieu (1)	0.00	0.00	0.00	0.00	0.00	0.00
babic (2)	**0.68**	**0.36**	**0.36**	**0.71**	**0.33**	**0.33**
fuhs (1)	**0.66**	**0.59**	0.14	**0.71**	**0.78**	0.07
nec (10)	0.12	0.01	0.12	**0.78**	**0.49**	0.24

Family	VIG orig.	VIG first 100	VIG all	CVIG orig.	CVIG first 100	CVIG all
md5gen (1)	**0.61**	**0.74**	0.02	**0.78**	**0.96**	0.02
ibm (2)	**0.84**	**0.50**	**0.43**	**0.81**	**0.58**	**0.42**
anbulagan (1)	**0.55**	0.03	0.00	**0.87**	0.13	0.05
manolios (9)	0.20	0.07	0.10	**0.76**	0.27	0.15
nec (10)	0.12	0.05	0.05	**0.78**	**0.30**	0.22

Modularity of (only) learnt clauses, computed using original partition, for all the instances that were solved (top), and only for unsatisfiable ones (bottom).

Modularity of Random Formulas

- **Random formulas** have a clear community structure?
  Obviously NO!!
- **Modularity** can be used as a reference value to compare how *modular* other kind of instances are:
  - Is it as high in random formulas as in industrial ones?
  - Is it dependent on the clause-variable ratio α?
  - Is it dependent on the number of variables?
  - How modularity is affected by the effects of:
    - Preprocessing the formula?
    - Learning new clauses?

Modularity of Random Formulas

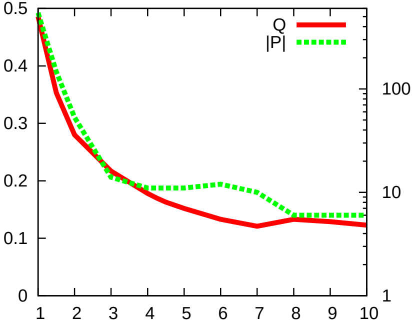
- **Random formulas** have a clear community structure?
  Obviously NO!!
- **Modularity** can be used as a reference value to compare how *modular* other kind of instances are:
  - Is it as high in random formulas as in industrial ones?
  - Is it dependent on the clause-variable ratio α?
  - Is it dependent on the number of variables?
  - How modularity is affected by the effects of:
    - Preprocessing the formula?
    - Learning new clauses?

Modularity of Random Formulas

- **Random formulas** have a clear community structure?
  Obviously NO!!
- **Modularity** can be used as a reference value to compare how *modular* other kind of instances are:
  - Is it as high in random formulas as in industrial ones?
  - Is it dependent on the clause-variable ratio α?
  - Is it dependent on the number of variables?
  - How modularity is affected by the effects of:
    - Preprocessing the formula?
    - Learning new clauses?

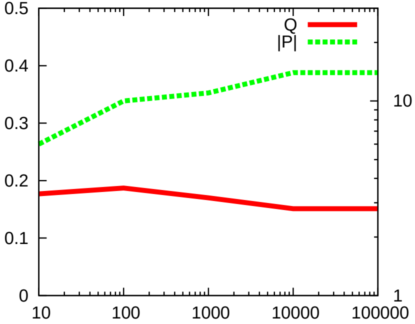
Modularity of Random Formulas

n	α	Q	$\|P\|$	larg.	iter
10^4	1	0.486	545	3.8	54
10^4	1.5	0.353	146	5.1	52
10^4	2	0.280	53	6.8	51
10^4	3	0.217	14	15.5	64
10^4	4	0.178	11	14.8	54
10^4	4.25	0.170	11	14.6	53
10^4	4.5	0.163	11	14.7	53
10^4	5	0.152	11	14.3	51
10^4	6	0.133	12	13.9	53
10^4	7	0.120	10	15.0	56
10^4	8	0.138	6	25.0	50
10^4	9	0.130	6	24.3	49
10^4	10	0.123	6	24.4	47



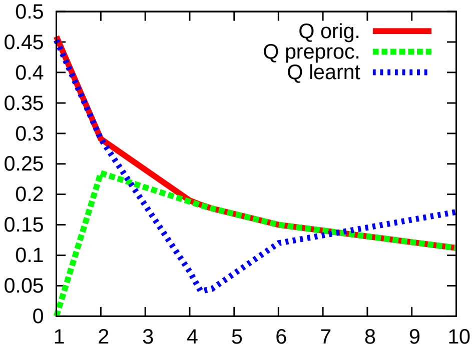
Modularity of random formulas varying the clause variable ratio α.

n	α	Q	$\|P\|$	larg.	iter
10^2	4.25	0.177	6	14.5	11
10^3	4.25	0.187	10.5	11.4	35
10^4	4.25	0.170	11	12.2	53
10^5	4.25	0.151	14	6.8	102
10^6	4.25	0.151	14	5.7	167



Modularity of random formulas at the peak transition region (clause-var. ratio $\alpha = 4.25$) varying the number of variables (n).

n	α	Orig.	Prep.	Learnt	C.C.
300	1	0.459	0	0.453	1
300	2	0.291	0.235	0.291	1
300	4	0.190	0.188	0.073	1
300	4.25	0.183	0.182	0.041	1
300	4.5	0.177	0.177	0.045	1
300	6	0.150	0.150	0.120	1
300	10	0.112	0.112	0.171	1



Modularity of random formulas (varying the clause variable ratio) after preprocessing, and including all learnt clauses.

Conclusions

- Industrial SAT instances have a **clear community structure**.
- Preprocessing these instances has almost no impact in this structure. However, it eliminates almost all the small unconnected components.
- Modularity weakly decreases with the learnt clauses. Therefore, learning does **not completely destroy the organization of the formula**. Nevertheless, it does **change the partitions structure**.
- Modularity of random instances is **only significant** for very low clause-variable ratios (α). Even for them, the modularity is **not as high as for industrial instances**. No abrupt change in the phase transition point is observed. This modularity does not depend on the number of variables (it seems to tend to a particular value).
- Preprocessing has almost no impact on the modularity of the formula. However, learning process **totally changes** this modularity. So, it totally changes the structure of the formula.

Conclusions

- Industrial SAT instances have a **clear community structure**.
- Preprocessing these instances has almost no impact in this structure. However, it eliminates almost all the small unconnected components.
- Modularity weakly decreases with the learnt clauses. Therefore, learning does **not completely destroy the organization of the formula**. Nevertheless, it does **change the partitions structure**.
- Modularity of random instances is **only significant** for very low clause-variable ratios (α). Even for them, the modularity is **not as high as for industrial instances**. No abrupt change in the phase transition point is observed. This modularity does not depend on the number of variables (it seems to tend to a particular value).
- Preprocessing has almost no impact on the modularity of the formula. However, learning process **totally changes** this modularity. So, it totally changes the structure of the formula.

Conclusions

- Industrial SAT instances have a **clear community structure**.
- Preprocessing these instances has almost no impact in this structure. However, it eliminates almost all the small unconnected components.
- Modularity weakly decreases with the learnt clauses. Therefore, learning does **not completely destroy the organization of the formula**. Nevertheless, it does **change the partitions structure**.
- Modularity of random instances is **only significant** for very low clause-variable ratios (α). Even for them, the modularity is **not as high as for industrial instances**. No abrupt change in the phase transition point is observed. This modularity does not depend on the number of variables (it seems to tend to a particular value).
- Preprocessing has almost no impact on the modularity of the formula. However, learning process **totally changes** this modularity. So, it totally changes the structure of the formula.

Conclusions

- Industrial SAT instances have a **clear community structure**.
- Preprocessing these instances has almost no impact in this structure. However, it eliminates almost all the small unconnected components.
- Modularity weakly decreases with the learnt clauses. Therefore, learning does **not completely destroy the organization of the formula**. Nevertheless, it does **change the partitions structure**.
- Modularity of random instances is **only significant** for very low clause-variable ratios (α). Even for them, the modularity is **not as high as for industrial instances**. No abrupt change in the phase transition point is observed. This modularity does not depend on the number of variables (it seems to tend to a particular value).
- Preprocessing has almost no impact on the modularity of the formula. However, learning process **totally changes** this modularity. So, it totally changes the structure of the formula.

Future Work

- To understand better how to **exploit** the community structure in SAT solvers (heuristics, learning, restarts, ...).
- **Generate** industrial-like random SAT instances with similar modularity to industrial instances.
- Study **other measures** of complex networks (such as the fractal dimension).

Thank you for your attention!!