The Lovász Local Lemma and Satisfiability

Algorithmic Aspects

Robin Moser

ETH Zurich

China Theory Week - September 2010

1. Introduction
   - Problem overview
2. Proof
   - Algorithm
   - The proof
3. Further work

Setting: the k-SAT problem

a k-CNF formula:

$$F = \underbrace{(x_1 \vee \bar{x}_2 \vee \ldots \vee \bar{x}_{19})}_{k} \wedge \underbrace{(\bar{x}_3 \vee x_7 \vee \ldots \vee x_{19})}_{k} \wedge \ldots$$

A conjunction of disjunctions (*clauses*), each composed of k *literals*.

Setting: the k-SAT problem

a k-CNF formula:

$$F = \underbrace{(x_1 \vee \bar{x}_2 \vee \ldots \vee \bar{x}_{19})}_{k} \wedge \underbrace{(\bar{x}_3 \vee x_7 \vee \ldots \vee x_{19})}_{k} \wedge \ldots$$

A conjunction of disjunctions (*clauses*), each composed of k *literals*.

Is there a satisfying assignment of truth values? Which one?

Setting: the k-SAT problem

a k-CNF formula:

$$F = \underbrace{(x_1 \vee \bar{x}_2 \vee \ldots \vee \bar{x}_{19})}_{k} \wedge \underbrace{(\bar{x}_3 \vee x_7 \vee \ldots \vee x_{19})}_{k} \wedge \ldots$$

A conjunction of disjunctions (*clauses*), each composed of k *literals*.

Is there a satisfying assignment of truth values? Which one?

Notation:
n : number of variables

Setting: the k-SAT problem

a k-CNF formula:

$$F = \underbrace{(x_1 \vee \bar{x}_2 \vee \ldots \vee \bar{x}_{19})}_{k} \wedge \underbrace{(\bar{x}_3 \vee x_7 \vee \ldots \vee x_{19})}_{k} \wedge \ldots$$

A conjunction of disjunctions (*clauses*), each composed of k *literals*.

Is there a satisfying assignment of truth values? Which one?

Notation:
n : number of variables
m : number of clauses

Setting: the k-SAT problem

a k-CNF formula:

$$F = \underbrace{(x_1 \vee \bar{x}_2 \vee \ldots \vee \bar{x}_{19})}_{k} \wedge \underbrace{(\bar{x}_3 \vee x_7 \vee \ldots \vee x_{19})}_{k} \wedge \ldots$$

A conjunction of disjunctions (*clauses*), each composed of k *literals*.

Is there a satisfying assignment of truth values? Which one?

Notation:
n : number of variables
m : number of clauses
$\mathrm{vbl}(C)$: the set of variables occurring in clause C

A simple subclass

Neighbourhood of a clause $C \in F$:

$$\Gamma(C) := \{\ D \in F \mid D \neq C, \mathrm{vbl}(C) \cap \mathrm{vbl}(D) \neq \emptyset\ \}$$

A simple subclass

Neighbourhood of a clause $C \in F$:

$$\Gamma(C) := \{\ D \in F \mid D \neq C, \mathrm{vbl}(C) \cap \mathrm{vbl}(D) \neq \emptyset\ \}$$

Inclusive neighbourhood :

$$\Gamma^+(C) := \Gamma(C) \cup \{C\}$$

A simple subclass

Neighbourhood of a clause $C \in F$:

$$\Gamma(C) := \{\ D \in F \mid D \neq C, \mathrm{vbl}(C) \cap \mathrm{vbl}(D) \neq \emptyset\ \}$$

Inclusive neighbourhood :

$$\Gamma^+(C) := \Gamma(C) \cup \{C\}$$

Theorem (Erdös, Lovász '75)

Let F be any k-CNF formula. If each $C \in F$ has $|\Gamma^+(C)| \leq 2^k/e$, then F admits a satisfying assignment.

A simple subclass

Neighbourhood of a clause $C \in F$:

$$\Gamma(C) := \{\, D \in F \mid D \neq C, \mathrm{vbl}(C) \cap \mathrm{vbl}(D) \neq \emptyset \,\}$$

Inclusive neighbourhood :

$$\Gamma^{+}(C) := \Gamma(C) \cup \{C\}$$

Theorem (Erdös, Lovász '75)

Let F be any k-CNF formula. If each $C \in F$ has $|\Gamma^{+}(C)| \leq 2^{k}/e$, then F admits a satisfying assignment.

- classical proof is non-constructive

A simple subclass

Neighbourhood of a clause $C \in F$:

$$\Gamma(C) := \{ D \in F \mid D \neq C, \mathrm{vbl}(C) \cap \mathrm{vbl}(D) \neq \emptyset \}$$

Inclusive neighbourhood :

$$\Gamma^+(C) := \Gamma(C) \cup \{C\}$$

Theorem (Erdös, Lovász '75)

Let F be any k-CNF formula. If each $C \in F$ has $|\Gamma^+(C)| \leq 2^k/e$, then F admits a satisfying assignment.

- classical proof is non-constructive
- Q: can we *find* a satisfying assignment?

History

Previous approaches to the problem:

Beck, 1991: for neighbourhoods up to $2^{k/48}$

History

Previous approaches to the problem:

Beck, 1991: for neighbourhoods up to $2^{k/48}$
Alon, 1991: for neighbourhoods up to $2^{k/8}$

History

Previous approaches to the problem:

Beck, 1991: for neighbourhoods up to $2^{k/48}$
Alon, 1991: for neighbourhoods up to $2^{k/8}$
Srinivasan, 2008: for neighbourhoods up to $2^{k/4}$

History

Previous approaches to the problem:

Beck, 1991: for neighbourhoods up to $2^{k/48}$
Alon, 1991: for neighbourhoods up to $2^{k/8}$
Srinivasan, 2008: for neighbourhoods up to $2^{k/4}$
M, 2008: for neighbourhoods up to $2^{k/2}$

History

Previous approaches to the problem:

Beck, 1991: for neighbourhoods up to $2^{k/48}$
Alon, 1991: for neighbourhoods up to $2^{k/8}$
Srinivasan, 2008: for neighbourhoods up to $2^{k/4}$
M, 2008: for neighbourhoods up to $2^{k/2}$

Theorem

There exists a randomized algorithm which, given a k-CNF formula F with $\forall C \in F : |\Gamma^+(C)| \leq 2^{k-3}$, finds a satisfying assignment for F in expected polynomial time.

Algorithm

```
solve(F):
      start with a random assignment
      while(∃C ∈ F : C violated)
            pick lexicographically first such C
            locally_correct(C)
```

Algorithm

```
solve(F):
      start with a random assignment
      while(∃C ∈ F : C violated)
           pick lexicographically first such C
           locally_correct(C)

locally_correct(C):
      resample new values for vbl(C)
```

Algorithm

```
solve(F):
      start with a random assignment
      while(∃C ∈ F : C violated)
           pick lexicographically first such C
           locally_correct(C)

locally_correct(C):
      resample new values for vbl(C)
// post-condition:  strictly fewer violated clauses
```

Algorithm

```
solve(F):
      start with a random assignment
      while(∃C ∈ F : C violated)
            pick lexicographically first such C
            locally_correct(C)

locally_correct(C):
      resample new values for vbl(C)
      while(∃D ∈ Γ⁺(C) : D violated)
            pick lexicographically first such D
            locally_correct(D)
// post-condition:  strictly fewer violated clauses
```

Algorithm

```
solve(F):
      start with a random assignment
      while(∃C ∈ F : C violated) // repeats <= m times
            pick lexicographically first such C
            locally_correct(C)

locally_correct(C):
      resample new values for vbl(C)
      while(∃D ∈ Γ+(C) : D violated)
            pick lexicographically first such D
            locally_correct(D)
// post-condition:  strictly fewer violated clauses
```

Logging the Program Execution

```
solve(F):
   start with a random assignment
   while(∃C ∈ F : C violated)
     pick lexicographically first such C

     locally_correct(C)

locally_correct(C):
   resample new values for vbl(C)
   while(∃D ∈ Γ⁺(C) : D violated)
     pick lexicographically first such D

     locally_correct(D)
```

Logging the Program Execution

```
solve(F):
   start with a random assignment
   while(∃C ∈ F : C violated)
     pick lexicographically first such C
     log(‘‘new recursion for‘‘ + index(C))
     locally_correct(C)

locally_correct(C):
   resample new values for vbl(C)
   while(∃D ∈ Γ⁺(C) : D violated)
     pick lexicographically first such D

     locally_correct(D)
```

Logging the Program Execution

```
solve(F):
   start with a random assignment
   while(∃C ∈ F : C violated)
     pick lexicographically first such C
     log(“new recursion for“ + index(C))
     locally_correct(C)

locally_correct(C):
   resample new values for vbl(C)
   while(∃D ∈ Γ+(C) : D violated)
     pick lexicographically first such D
     log(->“next clause“ + relative_index(D,C))
     locally_correct(D)
```

Logging the Program Execution

```
solve(F):
   start with a random assignment
   while(∃C ∈ F : C violated)
     pick lexicographically first such C
     log(‘‘new recursion for‘‘ + index(C))
     locally_correct(C)

locally_correct(C):
   resample new values for vbl(C)
   while(∃D ∈ Γ+(C) : D violated)
     pick lexicographically first such D
     log(->‘‘next clause‘‘ + relative_index(D,C))
     locally_correct(D)
   log(<-)
```

Logging the Program Execution

A sample log looks like this [with storage space requirements]:

`new recursion for 6`	[$\log m$ bits]
`    next clause 1`	[$(k-3)+1+1$ bits]
`        next clause 2`	[$(k-3)+1+1$ bits]
`    next clause 2`	[$(k-3)+1+1$ bits]
`    ...`	

Lemma

Each line of the log allows to reconstruct k bits of the random input used by the algorihtm.

Information Theoretic Balance

- at most $\mathcal{O}(m \log m)$ bits (in total) output by top-level calls

Information Theoretic Balance

- at most $\mathcal{O}(m \log m)$ bits (in total) output by top-level calls
- every further recursive call: $k - 1$ bits

Information Theoretic Balance

- at most $\mathcal{O}(m \log m)$ bits (in total) output by top-level calls
- every further recursive call: $k - 1$ bits
- every line allows to reconstruct k bits of random input

Information Theoretic Balance

- at most $\mathcal{O}(m \log m)$ bits (in total) output by top-level calls
- every further recursive call: $k - 1$ bits
- every line allows to reconstruct k bits of random input
- after $\mathcal{O}(m \log m)$, process starts compressing fully random data

Information Theoretic Balance

- at most $\mathcal{O}(m \log m)$ bits (in total) output by top-level calls
- every further recursive call: $k - 1$ bits
- every line allows to reconstruct k bits of random input
- after $\mathcal{O}(m \log m)$, process starts compressing fully random data

The process has to stop in $\mathcal{O}(m \log m)$ time. □

Further work

References:

- Schweitzer '09: independently found almost same proof

Further work

References:

- Schweitzer '09: independently found almost same proof

- Final version in collaboration with Gábor Tardos:
  - $2^k/e$ neighbours (no gap anymore)

Further work

References:

- Schweitzer '09: independently found almost same proof
- Final version in collaboration with Gábor Tardos:
  - $2^k/e$ neighbours (no gap anymore)
  - rules on choice of constraints not necessary

Further work

References:

- Schweitzer '09: independently found almost same proof

- Final version in collaboration with Gábor Tardos:
  - $2^k/e$ neighbours (no gap anymore)
  - rules on choice of constraints not necessary
  - generalization to applications beyond SAT

Derandomization and Output Distribution

References:

- Chandrasekaran, Goyal, Haeupler '09:

Derandomization and Output Distribution

References:

- Chandrasekaran, Goyal, Haeupler '09:
  - deterministic variant if neighborhood of each clause is at most $2^{k/(1+\epsilon)}$ in size

Derandomization and Output Distribution

References:

- Chandrasekaran, Goyal, Haeupler '09:
  - deterministic variant if neighborhood of each clause is at most $2^{k/(1+\epsilon)}$ in size
  - works by considering partial witness trees and adding the ϵ-slack

Derandomization and Output Distribution

References:

- Chandrasekaran, Goyal, Haeupler '09:
  - deterministic variant if neighborhood of each clause is at most $2^{k/(1+\epsilon)}$ in size
  - works by considering partial witness trees and adding the ϵ-slack
- Haeupler, Saha, Srinivasan '10:

Derandomization and Output Distribution

References:

- Chandrasekaran, Goyal, Haeupler '09:
    - deterministic variant if neighborhood of each clause is at most $2^{k/(1+\epsilon)}$ in size
    - works by considering partial witness trees and adding the ϵ-slack
- Haeupler, Saha, Srinivasan '10:
    - analyse the output distribution of the algorithm

Derandomization and Output Distribution

References:

- Chandrasekaran, Goyal, Haeupler '09:
  - deterministic variant if neighborhood of each clause is at most $2^{k/(1+\epsilon)}$ in size
  - works by considering partial witness trees and adding the ϵ-slack
- Haeupler, Saha, Srinivasan '10:
  - analyse the output distribution of the algorithm
  - settings with super-polynomially many events

Derandomization and Output Distribution

References:

- Chandrasekaran, Goyal, Haeupler '09:
  - deterministic variant if neighborhood of each clause is at most $2^{k/(1+\epsilon)}$ in size
  - works by considering partial witness trees and adding the ϵ-slack
- Haeupler, Saha, Srinivasan '10:
  - analyse the output distribution of the algorithm
  - settings with super-polynomially many events
  - interesting applications (e.g. Santa Claus problem and coloring problems)

Simplified algorithm

The more sophisticated proof variant demonstrates the following simplified algorithm to terminte in $\mathcal{O}(mk)$ expected time:

solve(F):
  start with a random assignment
  while($\exists C \in F : C$ violated)
    pick *any* such C
    resample new values for vbl(C)

Simplified algorithm

The more sophisticated proof variant demonstrates the following simplified algorithm to terminte in $\mathcal{O}(mk)$ expected time:

solve(F):
  start with a random assignment
  while($\exists C \in F : C$ violated)
    pick *any* such C
    resample new values for vbl(C)

=> That's almost Schöning's algorithm for general k-SAT with success probability

$$\left(\frac{1}{2}\frac{k}{k-1}\right)^n$$

Simplified algorithm

The more sophisticated proof variant demonstrates the following simplified algorithm to terminte in $\mathcal{O}(mk)$ expected time:

```
solve(F):
  start with a random assignment
  while(∃C ∈ F : C violated)
    pick any such C
    Schöning: flip one u.a.r. from vbl(C)
```

=> That's almost Schöning's algorithm for general k-SAT with success probability

$$\left(\frac{1}{2}\frac{k}{k-1}\right)^n$$

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

- 'throw away' **all** previous information on the support of the violated clause

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

- 'throw away' **all** previous information on the support of the violated clause
- complexity linear

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

- 'throw away' **all** previous information on the support of the violated clause
- complexity linear

For $2^k/e$ or more neighbors:

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

- 'throw away' **all** previous information on the support of the violated clause
- complexity linear

For $2^k/e$ or more neighbors:

- be minimalistic: change **as little as possible** information on the support of the violated clause

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

- 'throw away' **all** previous information on the support of the violated clause
- complexity linear

For $2^k/e$ or more neighbors:

- be minimalistic: change **as little as possible** information on the support of the violated clause
- complexity exponential

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

- 'throw away' **all** previous information on the support of the violated clause
- complexity linear

For $2^k/e$ or more neighbors:

- be minimalistic: change **as little as possible** information on the support of the violated clause
- complexity exponential

Open questions:

- does Schöning work for the LLL case?

Juxtaposition

For up to $2^k/e - 1$ neighbors per clause:

- 'throw away' **all** previous information on the support of the violated clause
- complexity linear

For $2^k/e$ or more neighbors:

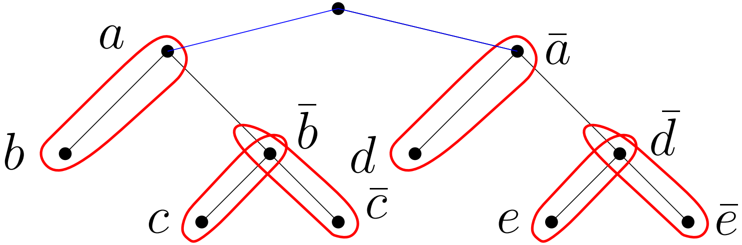
- be minimalistic: change **as little as possible** information on the support of the violated clause
- complexity exponential

Open questions:

- does Schöning work for the LLL case?
- is there such a jump in complexity or can it be smoothened?

Structural results on the 'jump'

By Gebauer, Szabó and Tardos: there are unsatisfiable formulas where each clause has $2^k(e^{-1} + o(1))$ neighbors, so the LLL is asymptotically tight for SAT.



The formulas can be constructed in such a way that no variables is featured in more than $(2/e + \epsilon) \cdot 2^k/k$ clauses.

Thanks

THANK YOU