The entropy compression technique

William Lochet, UiB

Plan of the lecture

1 Lovász Local Lemma and Moser's Algorithm

Plan of the lecture

1 Lovász Local Lemma and Moser's Algorithm

2 Examples of application

Plan of the lecture

1 Lovász Local Lemma and Moser's Algorithm

2 Examples of application

- Square-free word
- Acyclic edge-colouring

Lovász Local Lemma and Moser's Algorithm

The Lovász Local Lemma (LLL) setting

- Probability space Ω + Set of bad events $\mathcal{B} = \{B_1, \ldots, B_m\}$.
- If $\{B_i\}$ are independent, $Pr[\cap \bar{B}_i] = \prod_{i=1}^{m}(1 - Pr[B_i])$.

The Lovász Local Lemma (LLL) setting

- Probability space Ω + Set of bad events $\mathcal{B} = \{B_1, \ldots, B_m\}$.
- If $\{B_i\}$ are independent, $Pr[\cap \bar{B}_i] = \prod_{i=1}^{m}(1 - Pr[B_i])$.
- What happens when the $\{B_i\}$ are not independent?

The Lovász Local Lemma (LLL) setting

- Probability space Ω + Set of bad events $\mathcal{B} = \{B_1, \ldots, B_m\}$.
- If $\{B_i\}$ are independent, $Pr[\cap \bar{B}_i] = \prod_{i=1}^{m}(1 - Pr[B_i])$.
- What happens when the $\{B_i\}$ are not independent?

Lemma (Lovász 1975)

- *If each B_i is independent from all but d events;*
- $Pr[B_i] \leq p$*; and*
- $e \cdot p \cdot d \leq 1$.

Then $Pr[\cap \bar{B}_i] > 0$

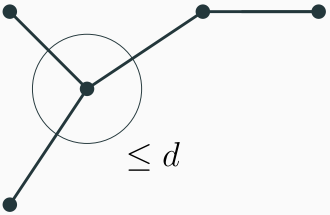
The Lovász Local Lemma (LLL) setting

- Probability space Ω + Set of bad events $\mathcal{B} = \{B_1, \ldots, B_m\}$.
- If $\{B_i\}$ are independent, $Pr[\cap \bar{B}_i] = \prod_{i=1}^{m}(1 - Pr[B_i])$.
- What happens when the $\{B_i\}$ are not independent?

Lemma (Lovász 1975)

- *If each B_i is independent from all but d events;*
- *$Pr[B_i] \leq p$; and*
- *$e \cdot p \cdot d \leq 1$.*

Then $Pr[\cap \bar{B}_i] > 0$



The canonical example

Definition (k-CNF)
A k-CNF formula is a conjunction of m clauses $(C_1, \ldots, C_m)$, where a clause is a disjunction of k literals.

$F = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_3 \vee \bar{x_4} \vee \bar{x_5}) \wedge (x_1 \vee \bar{x_2} \vee x_7)$

The canonical example

Definition (k-CNF)
A k-CNF formula is a conjunction of m clauses $(C_1, \ldots, C_m)$, where a clause is a disjunction of k literals.

$F = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_3 \vee \bar{x_4} \vee \bar{x_5}) \wedge (x_1 \vee \bar{x_2} \vee x_7)$

Question (k-sat)
Given a k-CNF formula F, is F satisfiable?

The canonical example

Definition (k-CNF)
A k-CNF formula is a conjunction of m clauses $(C_1, \ldots, C_m)$, where a clause is a disjunction of k literals.

$$F = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_3 \vee \bar{x_4} \vee \bar{x_5}) \wedge (x_1 \vee \bar{x_2} \vee x_7)$$

Question (k-sat)
Given a k-CNF formula F, is F satisfiable?

Pick the x_i **uniformly at random**, B_i :="C_i is not satisfied".

The canonical example

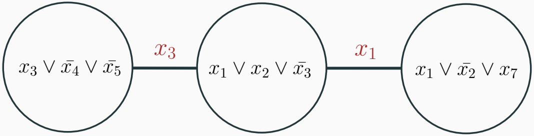
Definition (k-CNF)
A k-CNF formula is a conjunction of m clauses $(C_1, \ldots, C_m)$, where a clause is a disjunction of k literals.

$$F = (x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_3 \vee \bar{x_4} \vee \bar{x_5}) \wedge (x_1 \vee \bar{x_2} \vee x_7)$$

Question (k-sat)
Given a k-CNF formula F, is F satisfiable?

Pick the x_i **uniformly at random**, B_i :="C_i is not satisfied".

Observation
If C_i and C_j do not share a variable, then B_i and B_j are independent. Moreover, $Pr[B_i] = (\frac{1}{2})^k$



Tight bound and breakthrough

Theorem
Every k-CNF formula where each clause shares variables with at most $d \leq 2^k/e$ other clauses is satisfiable.

By applying LLL since $e \cdot Pr[B_i] \cdot d \leq e \cdot (\frac{1}{2})^k \cdot \frac{2^k}{e} \leq 1$.

Tight bound and breakthrough

Theorem
Every k-CNF formula where each clause shares variables with at most $d \leq 2^k/e$ other clauses is satisfiable.

By applying LLL since $e \cdot Pr[B_i] \cdot d \leq e \cdot (\frac{1}{2})^k \cdot \frac{2^k}{e} \leq 1$.

Question
Can we find such an assignment efficiently?

- $Pr[\cap \bar{B}_i]$ is exponentially small in the **number of clauses**.

Tight bound and breakthrough

Theorem
Every k-CNF formula where each clause shares variables with at most $d \leq 2^k/e$ other clauses is satisfiable.

By applying LLL since $e \cdot Pr[B_i] \cdot d \leq e \cdot (\frac{1}{2})^k \cdot \frac{2^k}{e} \leq 1$.

Question
Can we find such an assignment efficiently?

- $Pr[\cap \bar{B}_i]$ is exponentially small in the **number of clauses**.
- Beck, 1991 → existence of an algorithm when $d \leq 2^{k/48}$.

Tight bound and breakthrough

Theorem
Every k-CNF formula where each clause shares variables with at most $d \leq 2^k/e$ other clauses is satisfiable.

By applying LLL since $e \cdot Pr[B_i] \cdot d \leq e \cdot (\frac{1}{2})^k \cdot \frac{2^k}{e} \leq 1$.

Question
Can we find such an assignment efficiently?

- $Pr[\cap \bar{B}_i]$ is exponentially small in the **number of clauses**.
- Beck, 1991 → existence of an algorithm when $d \leq 2^{k/48}$.

Theorem (Moser 2009, Moser and Tardos 2010)
If $d \leq 2^k/e$, then a solution can be found in $O(|V| + |C| log|C|)$.

Tight bound and breakthrough

Theorem
Every k-CNF formula where each clause shares variables with at most $d \leq 2^k/e$ other clauses is satisfiable.

By applying LLL since $e \cdot Pr[B_i] \cdot d \leq e \cdot (\frac{1}{2})^k \cdot \frac{2^k}{e} \leq 1$.

Question
Can we find such an assignment efficiently?

- $Pr[\cap \bar{B}_i]$ is exponentially small in the **number of clauses**.
- Beck, 1991 → existence of an algorithm when $d \leq 2^{k/48}$.

Theorem (Moser 2009, Moser and Tardos 2010)
If $d \leq 2^k/e$, then a solution can be found in $O(|V| + |C| log |C|)$.

- Best paper award STOC 2009.

Tight bound and breakthrough

Theorem
Every k-CNF formula where each clause shares variables with at most $d \leq 2^k/e$ other clauses is satisfiable.

By applying LLL since $e \cdot Pr[B_i] \cdot d \leq e \cdot (\frac{1}{2})^k \cdot \frac{2^k}{e} \leq 1$.

Question
Can we find such an assignment efficiently?

- $Pr[\cap \bar{B}_i]$ is exponentially small in the **number of clauses**.
- Beck, 1991 → existence of an algorithm when $d \leq 2^{k/48}$.

Theorem (Moser 2009, Moser and Tardos 2010)
If $d \leq 2^k/e$, then a solution can be found in $O(|V| + |C|log|C|)$.

- Best paper award STOC 2009.
- Gödel prize in 2020.

The algorithm

Suppose $F = C_1 \wedge \cdots \wedge C_m$ is a k-CNF and every clause C_i depends of variables $x_{i_1}, \ldots, x_{i_k}$.

Algorithm 1 Moser's Algorithm

```
Pick random values for x_1, ..., x_n
while There exists a clause C_i not satisfied do
    pick new values for all variables x_{i_1}, ..., x_{i_k} in C_i
end while
Return: Value of the variables x_1, ..., x_n
```

The algorithm

Suppose $F = C_1 \wedge \cdots \wedge C_m$ is a k-CNF and every clause C_i depends of variables $x_{i_1}, \ldots, x_{i_k}$.

Algorithm 1 Moser's Algorithm

1: Pick random values for $x_1, \ldots, x_n$
2: **while** There exists a clause C_i **not satisfied** **do**
3: pick new values for **all variables** $x_{i_1}, \ldots, x_{i_k}$ in C_i
4: **end while**
5: **Return:** Value of the variables $x_1, \ldots, x_n$

- Can we use the number of unsatisfied closes as **loop invariant**?

The algorithm

Suppose $F = C_1 \wedge \cdots \wedge C_m$ is a k-CNF and every clause C_i depends of variables $x_{i_1}, \ldots, x_{i_k}$.

Algorithm 1 Moser's Algorithm

```
Pick random values for x_1, ..., x_n
while There exists a clause C_i not satisfied do
    pick new values for all variables x_{i_1}, ..., x_{i_k} in C_i
end while
Return: Value of the variables x_1, ..., x_n
```

- Can we use the number of unsatisfied closes as **loop invariant**?
- No, changing the value x_{i_1} might change the status of some clause C_j **neighbour** of C_i.

Entropy compression

- We focus on the first t steps of the algorithm.
- All the random choices can be described with $n + tk$ bits.

Entropy compression

- We focus on the first t steps of the algorithm.
- All the random choices can be described with $n + tk$ bits.

Theorem (Moser and Tardos 2010)
If $t = \Omega(m \log(m))$, then there is a way to describe the running of t steps of the algorithm using $o(n + tk)$ bits.

Entropy compression

- We focus on the first t steps of the algorithm.
- All the random choices can be described with $n + tk$ bits.

Theorem (Moser and Tardos 2010)
If $t = \Omega(m \log(m))$, then there is a way to describe the running of t steps of the algorithm using $o(n + tk)$ bits.

The algorithm can then be seen as:

- Take as input the $n + tk$ random choices
- **Assuming the algorithm runs for t steps**, outputs an encoding of these random choices using this description

Description of a run

Definition (Log)
A **log** is a description of:

- The sequence $u = (u_1, \ldots, u_t)$ of clauses treated at each step
- The values $X_t = (x_1^t, \ldots, x_n^t)$ of the variables after t steps

Description of a run

Definition (Log)
A **log** is a description of:

- The sequence $u = (u_1, \ldots, u_t)$ of clauses treated at each step
- The values $X_t = (x_1^t, \ldots, x_n^t)$ of the variables after t steps

Lemma
Given u and X_t, we can recover the values $X_i = (x_1^i, \ldots, x_n^i)$ of the variables after i steps for any $i \in [t]$

Description of a run

Definition (Log)
A **log** is a description of:

- The sequence $u = (u_1, \dots, u_t)$ of clauses treated at each step
- The values $X_t = (x_1^t, \dots, x_n^t)$ of the variables after t steps

Lemma
Given u and X_t, we can recover the values $X_i = (x_1^i, \dots, x_n^i)$ of the variables after i steps for any $i \in [t]$

- Between X_t and X_{t-1} only the variables in C_{u_t} change

Description of a run

Definition (Log)
A **log** is a description of:

- The sequence $u = (u_1, \ldots, u_t)$ of clauses treated at each step
- The values $X_t = (x_1^t, \ldots, x_n^t)$ of the variables after t steps

Lemma
Given u and X_t, we can recover the values $X_i = (x_1^i, \ldots, x_n^i)$ of the variables after i steps for any $i \in [t]$

- Between X_t and X_{t-1} only the variables in C_{u_t} change
- Because C_{u_t} was not satisfied, we know the value of those variables.

Number of logs

Lemma

If R_1 and R_2 are two sets of $n + tk$ bits for which the algorithm ***does not terminate****, then the logs associated to R_1 and R_2 are different.*

Number of logs

Lemma
If R_1 and R_2 are two sets of $n + tk$ bits for which the algorithm ***does not terminate****, then the logs associated to R_1 and R_2 are different.*

It means that:

$$\#\text{random choices that do no terminate} \leq \#\text{of possible logs}$$

Number of logs

Lemma
If R_1 and R_2 are two sets of $n + tk$ bits for which the algorithm ***does not terminate****, then the logs associated to R_1 and R_2 are different.*

It means that:

$$\#\text{random choices that do no terminate} \leq \#\text{of possible logs}$$

This implies that the probability that the algorithm does not terminate after t steps is at most:

$$\frac{\#\text{of possible logs}}{\#\text{of possible random choices}}$$

Efficient encoding

Question
How to encode $u = (u_1, \ldots, u_t)$ *and* X_t *efficiently?*
(compared to $n + tk$ *bits)*

Efficient encoding

Question
How to encode $u = (u_1, \ldots, u_t)$ *and* X_t *efficiently?*
(compared to $n + tk$ *bits)*

- $X_t = (x_1^t, \ldots, x_n^t)$.

Efficient encoding

Question
How to encode $u = (u_1, \ldots, u_t)$ *and* X_t *efficiently?*
(compared to $n + tk$ *bits)*

- $X_t = (x_1^t, \ldots, x_n^t)$.
- Naively, u_i can be encoded using $\log(m)$ bits, Not good!

Efficient encoding

Question

How to encode $u = (u_1, \ldots, u_t)$ *and* X_t *efficiently?*

(compared to $n + tk$ *bits)*

- $X_t = (x_1^t, \ldots, x_n^t)$.
- Naively, u_i can be encoded using $\log(m)$ bits, Not good!

Observation

If $C_{u_{i+1}}$ *is a* ***neighbour*** *of* C_{u_i}*, it costs* $\log(2^k/e) < k$.

Efficient encoding

Question
How to encode $u = (u_1, \ldots, u_t)$ *and* X_t *efficiently?*
(compared to $n + tk$ *bits)*

- $X_t = (x_1^t, \ldots, x_n^t)$.
- Naively, u_i can be encoded using $\log(m)$ bits, Not good!

Observation
If $C_{u_{i+1}}$ *is a* ***neighbour*** *of* C_{u_i}, *it costs* $\log(2^k/e) < k$.

If the algorithm runs long enough, it will be the case for most u_i.

Square-free words

Square-free words

Definition
A word w over some alphabet Σ is said to be **square-free** if it does not contain a word of type uu as a subword.

Square-free words

Definition
A word w over some alphabet Σ is said to be **square-free** if it does not contain a word of type uu as a subword.

- $u = abcbca$ is not square-free.
- $v = abcba$ is.

Square-free words

Definition
A word w over some alphabet Σ is said to be **square-free** if it does not contain a word of type uu as a subword.

- $u = abcbca$ is not square-free.
- $v = abcba$ is.

Theorem (Thue 1906)
There exists an infinite word without square when $|\Sigma| \geq 3$.

Square-free words

Definition
A word w over some alphabet Σ is said to be **square-free** if it does not contain a word of type uu as a subword.

- $u = abcbca$ is not square-free.
- $v = abcba$ is.

Theorem (Thue 1906)
There exists an infinite word without square when $|\Sigma| \geq 3$.

Question
Suppose $L_1, \ldots, L_n$ are n list of 3 elements of Σ, does there exists a ***square-free*** *word $u = u_1 u_2 \ldots u_n$ such that $u_i \in L_i$?*

Algorithm for $|L_i| = 5$

Theorem (Grytczuk, Kozik and Micek 2013)
Entropy compression works for $|L_i| \geq 4$.

Algorithm for $|L_i| = 5$

Theorem (Grytczuk, Kozik and Micek 2013)
Entropy compression works for $|L_i| \geq 4$.

Algorithm 2 Finding square-free words

$u \leftarrow$ empty word
while $|u| < n$ **do**
  $a \leftarrow$ random letter in $L_{|u|+1}$
  $u \leftarrow ua$
  if $u = wbb$ for some word b **then**
    $u \leftarrow wb$
  end if
end while

Algorithm for $|L_i| = 5$

Theorem (Grytczuk, Kozik and Micek 2013)
Entropy compression works for $|L_i| \geq 4$.

Algorithm 2 Finding square-free words

$u \leftarrow$ empty word
while $|u| < n$ **do**
  $a \leftarrow$ random letter in $L_{|u|+1}$
  $u \leftarrow ua$
  if $u = wbb$ for some word b **then**
    $u \leftarrow wb$
  end if
end while

Lemma
This algorithm terminates in $O(n)$ steps.

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0,1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := \emptyset$$
$$l := \emptyset$$

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0, 1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := a$$
$$l := 1$$

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0,1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := ab$$
$$l := 11$$

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0,1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := aba$$
$$l := 111$$

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0,1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := abab$$
$$l := 1111$$

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0, 1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := abab$$
$$l := 1111$$

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0,1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := ab$$
$$l := 111100$$

Encoding

The **log** of a run consists of the value of u at the end and a word $l \in \{0,1\}^*$ obtained by:

- Adding 1 each time the algorithm adds a letter.
- Adding 0 each time the algorithm removes a letter.

$$u := abc$$
$$l := 1111001$$

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

$$u := abc$$
$$l := 1111001$$

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

$u := abc$

$l := 1111001$

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

$$u := ab$$
$$l := 111100$$

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

$$u := ab$$
$$l := 111100$$

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

$$u := abab$$
$$l := 1111$$

Counting

Lemma
Given a log : (u, l) *it is possible to deduce the set of random choices.*

- Two sets of random choices that **do not terminate** produce different logs.

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

- Two sets of random choices that **do not terminate** produce different logs.
- The number of possible logs of t steps is $5^n \cdot 2^{2t}$

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

- Two sets of random choices that **do not terminate** produce different logs.
- The number of possible logs of t steps is $5^n \cdot 2^{2t}$
- The number of possible random choices is 5^t

Counting

Lemma
Given a log : (u, l) it is possible to deduce the set of random choices.

- Two sets of random choices that **do not terminate** produce different logs.
- The number of possible logs of t steps is $5^n \cdot 2^{2t}$
- The number of possible random choices is 5^t

Theorem
The probability that the algorithm does not terminate after t steps is at most $\frac{5^n 4^t}{5^t} = \frac{4^t}{5^{t-n}}$.

for $t = 11n$, we have $\frac{4^t}{5^{t-n}} \leq 1/2$.

List of size 4, 3?

With a better counting, we can prove:

Theorem (Grytczuk, Kozik and Micek 2013)
Entropy compression works for $|L_i| \geq 4$.

List of size 4, 3?

With a better counting, we can prove:

Theorem (Grytczuk, Kozik and Micek 2013)
Entropy compression works for $|L_i| \geq 4$.

Conjecture
It works when $|L_i| \geq 3$.

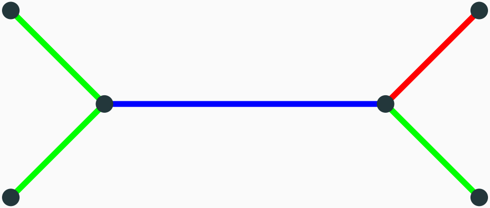
- If all the list are the same, then this is the result of Thue.
- Experimentally, the algorithm seems to work, but much slower

Acyclic colouring

Proper Edge Colouring

Definition
An edge-colouring of a graph G is said to be **proper** if:

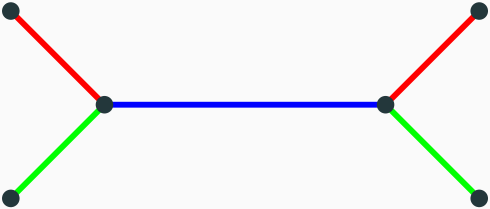
- No two adjacent **edges** have the same colour



Proper Edge Colouring

Definition
An edge-colouring of a graph G is said to be **proper** if:

- No two adjacent **edges** have the same colour



Theorem (Vizing 1964)
For any graph G, there exists a ***proper edge colouring*** *using $\Delta(G) + 1$ colours.*

Where $\Delta(G)$ is the maximal degree.

Acyclic Edge Colouring

Definition
An edge-colouring of a graph G is said to be **acyclic** if:

- It is proper
- There is no **bicoloured** cycle.

Definition
An edge-colouring of a graph G is said to be **acyclic** if:

- It is proper
- There is no **bicoloured** cycle.

Theorem (Alon, McDiarmid and Reed 1991)
*For any graph G, there exists an **acyclic edge colouring** using at most $64\Delta(G)$ colours.*

- After a series of improvements the best bound is now 3.74Δ
- It has been conjectured that $\Delta + 2$ should be enough.

Using entropy compression

Theorem (Esperet and Parreau 2013)
*For any graph G, there exists an **acyclic edge colouring** using at most $4\Delta(G)$ colours.*

We will do the proof with $7\Delta(G)$ colours.

Using entropy compression

Theorem (Esperet and Parreau 2013)
*For any graph G, there exists an **acyclic edge colouring** using at most $4\Delta(G)$ colours.*

We will do the proof with $7\Delta(G)$ colours.
The algorithm will colour the edges one by one, ensuring:

- The colouring is proper
- The colouring is acyclic

Sampling a proper colouring

Suppose G is partially coloured and we are trying to colour (uv)

Sampling a proper colouring

Suppose G is partially coloured and we are trying to colour (uv)

- v and u are both adjacent to at most Δ colours
- There is $(7-2)\Delta = 5\Delta$ colours available

Sampling a proper colouring

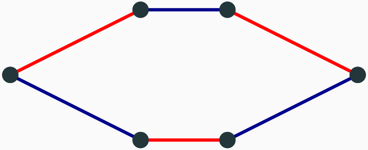
Suppose G is partially coloured and we are trying to colour (uv)

- v and u are both adjacent to at most Δ colours
- There is $(7-2)\Delta = 5\Delta$ colours available

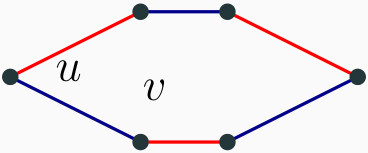
The algorithm will pick uniformly at random a color among the 5Δ available. The (partial) colouring throughout this process is always **proper**.

Removing bi-coloured cycles

Lemma
If G has a proper edge colouring, then any bi-coloured cycle C is ***even*** *and with* ***alternating colors****.*



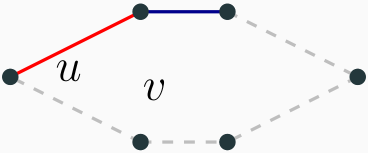
Lemma
If G has a proper edge colouring, then any bi-coloured cycle C is ***even*** *and with* ***alternating colors****.*



If after colouring the edge (uv), there is a bi-coloured cycle C of size $2k$ containing uv:

Removing bi-coloured cycles

Lemma
If G has a proper edge colouring, then any bi-coloured cycle C is ***even*** *and with* ***alternating colors****.*

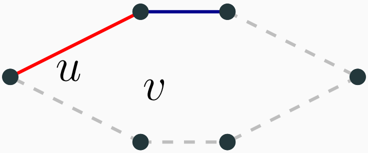


If after colouring the edge (uv), there is a bi-coloured cycle C of size $2k$ containing uv:

- Remove the colours all the edges of the cycle except 2

Removing bi-coloured cycles

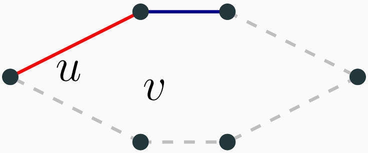
Lemma
If G has a proper edge colouring, then any bi-coloured cycle C is ***even*** *and with* ***alternating colors****.*



If after colouring the edge (uv), there is a bi-coloured cycle C of size $2k$ containing uv:

- Remove the colours all the edges of the cycle except 2
- Knowing uv, we only need to know the cycle C in order to recover the colouring. There are Δ^{2k-2} possible choices.

Lemma
If G has a proper edge colouring, then any bi-coloured cycle C is ***even*** *and with* ***alternating colors****.*



If after colouring the edge (uv), there is a bi-coloured cycle C of size $2k$ containing uv:

- Remove the colours all the edges of the cycle except 2
- Knowing uv, we only need to know the cycle C in order to recover the colouring. There are Δ^{2k-2} possible choices.
- To compare with the $(5\Delta)^{2k-2}$ possible choices of colour.

The algorithm

We will keep two logs: (L, R) and assume there is an arbitrary order on the edges $e_1, \ldots, e_m$.

The algorithm

We will keep two logs: (L, R) and assume there is an arbitrary order on the edges $e_1, \ldots, e_m$.

Algorithm 3 Finding an acyclic colouring

$c \leftarrow$ empty colouring.
while there is an non coloured edge e_i **do**
  $c(e_i) \leftarrow$ random available colour.
  $L \leftarrow L \cdot 1$
  if $\exists$ bi-coloured cycle C of size $2k$ containing e_i **then**
    un-colour all edges of C except 2
    Add $(2k-2)$ 0's at the end of L
    Add to R the $2k-2$ integers to recover C from e_i
  end if
end while

Finishing the proof

Suppose the algorithm runs for t steps (while loop). We need to show the following two things:

1. (L, R) and the value of the colouring c at the end of the algorithm is enough to recover the set of random choices.

Finishing the proof

Suppose the algorithm runs for t steps (while loop). We need to show the following two things:

1. (L, R) and the value of the colouring c at the end of the algorithm is enough to recover the set of random choices.
2. If t is big enough, the number of possible (L, R) and c is much smaller $(5\Delta)^t$.

Recovering the random choices.

Lemma
Given (L, R), we can recover the set of coloured edges after i steps for any $i \in [t]$.

Recovering the random choices.

Lemma
Given (L, R), we can recover the set of coloured edges after i steps for any $i \in [t]$.

Proof.
By induction on i (at $i = 0$, no edge is coloured). Suppose we know the set of coloured edges after $i - 1$ steps.

- The algorithm starts the loop by colouring the non-coloured edge with smallest index, e_j.

Recovering the random choices.

Lemma
Given (L, R), we can recover the set of coloured edges after i steps for any $i \in [t]$.

Proof.
By induction on i (at $i = 0$, no edge is coloured). Suppose we know the set of coloured edges after $i - 1$ steps.

- The algorithm starts the loop by colouring the non-coloured edge with smallest index, e_j.
- If after the i-th 1 in L there is another 1: No bad event.

Recovering the random choices.

Lemma
Given (L, R), we can recover the set of coloured edges after i steps for any $i \in [t]$.

Proof.
By induction on i (at $i = 0$, no edge is coloured). Suppose we know the set of coloured edges after $i - 1$ steps.

- The algorithm starts the loop by colouring the non-coloured edge with smallest index, e_j.
- If after the i-th 1 in L there is another 1: No bad event.
- If there is a 0, the number of consecutive 0's tells us the length of the bad cycle C

Recovering the random choices.

Lemma
Given (L, R), we can recover the set of coloured edges after i steps for any $i \in [t]$.

Proof.
By induction on i (at $i = 0$, no edge is coloured). Suppose we know the set of coloured edges after $i - 1$ steps.

- The algorithm starts the loop by colouring the non-coloured edge with smallest index, e_j.
- If after the i-th 1 in L there is another 1: No bad event.
- If there is a 0, the number of consecutive 0's tells us the length of the bad cycle C
- By looking at R we are able to recover C from e_j

□

Recovering the random choices.

Lemma
Given (L, R) and the value of c at the end, we can recover the value of c after i steps for any $i \in [t]$.

Recovering the random choices.

Lemma
Given (L, R) and the value of c at the end, we can recover the value of c after i steps for any $i \in [t]$.

Proof.
By induction on $t - i$ (at $i = 0$, we already know c). Suppose we know the set of coloured edges after $t - i + 1$ steps.

- By looking at L we know if there is a bad cycle at step $t - i$.

Recovering the random choices.

Lemma

Given (L, R) and the value of c at the end, we can recover the value of c after i steps for any $i \in [t]$.

Proof.

By induction on $t - i$ (at $i = 0$, we already know c). Suppose we know the set of coloured edges after $t - i + 1$ steps.

- By looking at L we know if there is a bad cycle at step $t - i$.
- If there was not, we know by the previous lemma which edge was coloured at that step.

Recovering the random choices.

Lemma
Given (L, R) and the value of c at the end, we can recover the value of c after i steps for any $i \in [t]$.

Proof.
By induction on $t - i$ (at $i = 0$, we already know c). Suppose we know the set of coloured edges after $t - i + 1$ steps.

- By looking at L we know if there is a bad cycle at step $t - i$.
- If there was not, we know by the previous lemma which edge was coloured at that step.
- If there was a bad cycle, by looking at R we can recover this cycle and thus the colouring.

□

Counting the number of logs

After t steps:

- The number of possible L is smaller than 4^t

Counting the number of logs

After t steps:

- The number of possible L is smaller than 4^t
- The number of possible R is Δ^t

Counting the number of logs

After t steps:

- The number of possible L is smaller than 4^t
- The number of possible R is Δ^t
- The number of possible c is $(7\Delta)^m$

Counting the number of logs

After t steps:

- The number of possible L is smaller than 4^t
- The number of possible R is Δ^t
- The number of possible c is $(7\Delta)^m$
- The number of possible random choices is $(5\Delta)^t$.

Counting the number of logs

After t steps:

- The number of possible L is smaller than 4^t
- The number of possible R is Δ^t
- The number of possible c is $(7\Delta)^m$
- The number of possible random choices is $(5\Delta)^t$.

Overall when t is large enough, we have $4^t \cdot \Delta^t \cdot (7\Delta)^m < (5\Delta)^t$.

Counting the number of logs

After t steps:

- The number of possible L is smaller than 4^t
- The number of possible R is Δ^t
- The number of possible c is $(7\Delta)^m$
- The number of possible random choices is $(5\Delta)^t$.

Overall when t is large enough, we have $4^t \cdot \Delta^t \cdot (7\Delta)^m < (5\Delta)^t$.

Theorem
The algorithm terminates in linear time with constant probability.

Concluding remarks

Conjecture (Alon et al. 2001)
$\Delta + 2$ *should be enough*

Concluding remarks

Conjecture (Alon et al. 2001)
$\Delta + 2$ *should be enough*

- Similar arguments can get the bound down to 3.74Δ

Concluding remarks

Conjecture (Alon et al. 2001)
$\Delta + 2$ *should be enough*

- Similar arguments can get the bound down to 3.74Δ
- Cai et al. proved $(1+\epsilon)\Delta$ when the girth is larger than some constant $g(\epsilon)$

Concluding remarks

Conjecture (Alon et al. 2001)
$\Delta + 2$ *should be enough*

- Similar arguments can get the bound down to 3.74Δ
- Cai et al. proved $(1+\epsilon)\Delta$ when the girth is larger than some constant $g(\epsilon)$
- It seems like the "limit" of EC for this is 2Δ

Thank you!