Outline

Shuffling and Strong Stationary Times

Covertime

$s - t$ Connectivity

2-Sat
Card Shuffling

A *Permutation* $\sigma$ of $[n] = \{1, \ldots, n\}$ is a bijection $\sigma : [n] \rightarrow [n]$. 

Let $\Sigma_n$ be the set of all $n!$ permutations of $[n]$. 

Given an ordered set $[n]$ we wish to sample a permutation of $[n]$ uniformly. 

Sampling from uniform. 

Given a deck of $n$ cards take the top card and place it at random position in the deck. 

Markov chain on $\Sigma_n$ with $\pi$ uniform. 

**Top-to-Random (T-to-R) Shuffling**
A *Permutation* $\sigma$ of $[n] = \{1, \ldots, n\}$ is a bijection $\sigma : [n] \to [n]$.

Let $\Sigma_n$ be the set of all $n!$ permutations of $[n]$. 
A **Permutation** $\sigma$ of $[n] = \{1, \ldots, n\}$ is a bijection $\sigma : [n] \rightarrow [n]$.

Let $\Sigma_n$ be the set of all $n!$ permutations of $[n]$.

---

**Sampling from uniform.**

Given an ordered set $[n]$ we wish to sample a permutation of $[n]$ uniformly.
Card Shuffling

A *Permutation* $\sigma$ of $[n] = \{1, \ldots, n\}$ is a bijection $\sigma : [n] \rightarrow [n]$.

Let $\Sigma_n$ be the set of all $n!$ permutations of $[n]$.

**Sampling from uniform.**

Given an ordered set $[n]$ we wish to sample a permutation of $[n]$ uniformly.

**Top-to-Random (T-to-R) Shuffling**

Given a deck of $n$ cards take the top card and place it at random position in the deck.

- Markov chain on $\Sigma_n$ with $\pi$ uniform.
Card Shuffling

A *Permutation* $\sigma$ of $[n] = \{1, \ldots, n\}$ is a bijection $\sigma : [n] \rightarrow [n]$.

Let $\Sigma_n$ be the set of all $n!$ permutations of $[n]$.

**Sampling from uniform.**

Given an ordered set $[n]$ we wish to sample a permutation of $[n]$ uniformly.

**Top-to-Random (T-to-R) Shuffling**

Given a deck of $n$ cards take the top card and place it at random position in the deck.

- Markov chain on $\Sigma_n$ with $\pi$ uniform.
Strong Stationary Time

A *Strong Stationary Time* for a Markov Chain \((X_t)\) with stationary distribution \(\pi\) is a stopping time \(\tau\), possibly depending on the starting state \(x\), such that

\[
P_x[t = \tau, X_\tau = y] = P_x[t = \tau] \pi_y.
\]
A *Strong Stationary Time* for a Markov Chain \((X_t)\) with stationary distribution \(\pi\) is a stopping time \(\tau\), possibly depending on the stating state \(x\), such that

\[
P_x[t = \tau, X_\tau = y] = P_x[t = \tau] \pi_y.
\]

- Thus \(X_\tau\) has distribution \(\pi\) and is independent of \(\tau\).
A Strong Stationary Time for a Markov Chain \((X_t)\) with stationary distribution \(\pi\) is a stopping time \(\tau\), possibly depending on the stating state \(x\), such that
\[
P_x[t = \tau, X_\tau = y] = P_x[t = \tau] \pi_y.
\]

- Thus \(X_\tau\) has distribution \(\pi\) and is independent of \(\tau\).

### Mixing from Strong Stationary Times

If \(\tau\) is a strong stationary time then for any \(x \in \mathcal{I}\),
\[
\left\| P^t_x - \pi \right\|_{tv} \leq P[\tau > t \mid X_0 = x].
\]
A **Strong Stationary Time** for a Markov Chain \((X_t)\) with stationary distribution \(\pi\) is a stopping time \(\tau\), possibly depending on the stating state \(x\), such that

\[
P_x[t = \tau, X_\tau = y] = P_x[t = \tau] \pi_y.
\]

- Thus \(X_\tau\) has distribution \(\pi\) and is independent of \(\tau\).

**Mixing from Strong Stationary Times**

If \(\tau\) is a strong stationary time then for any \(x \in I\),

\[
\left\| P^t_x - \pi \right\|_{tv} \leq P[\tau > t \mid X_0 = x].
\]

**Proof:** For any \(A \subseteq I\) the difference \(P_x[X_t \in A] - \pi(A)\) is equal to

\[
P_x[X_t \in A \mid \tau > t] P_x[\tau > t] + P_x[X_t \in A \mid \tau \leq t] (1 - P_x[\tau > t]) - \pi(A)
\]
A **Strong Stationary Time** for a Markov Chain \((X_t)\) with stationary distribution \(\pi\) is a stopping time \(\tau\), possibly depending on the starting state \(x\), such that

\[
P_x[t = \tau, X_\tau = y] = P_x[t = \tau] \pi_y.
\]

- Thus \(X_\tau\) has distribution \(\pi\) and is independent of \(\tau\).

---

**Mixing from Strong Stationary Times**

If \(\tau\) is a strong stationary time then for any \(x \in \mathcal{I}\),

\[
\left\| P^t_x - \pi \right\|_{tv} \leq P[\tau > t \mid X_0 = x].
\]

**Proof:** For any \(A \subseteq \mathcal{I}\) the difference \(P_x[X_t \in A] - \pi(A)\) is equal to

\[
P_x[X_t \in A \mid \tau > t] P_x[\tau > t] + P_x[X_t \in A \mid \tau \leq t] (1 - P_x[\tau > t]) - \pi(A)
\]

\[
= (P_x[X_t \in A \mid \tau > t] - \pi(A)) P_x[\tau > t].
\]
Strong Stationary Time

A **Strong Stationary Time** for a Markov Chain \((X_t)\) with stationary distribution \(\pi\) is a stopping time \(\tau\), possibly depending on the stating state \(x\), such that

\[
P_x[t = \tau, X_\tau = y] = P_x[t = \tau] \pi_y.
\]

- Thus \(X_\tau\) has distribution \(\pi\) and is independent of \(\tau\).

**Mixing from Strong Stationary Times**

If \(\tau\) is a strong stationary time then for any \(x \in \mathcal{I}\),

\[
\left\| P^t_x - \pi \right\|_{tv} \leq P[\tau > t \mid X_0 = x].
\]

**Proof:** For any \(A \subseteq \mathcal{I}\) the difference \(P_x[X_t \in A] - \pi(A)\) is equal to

\[
P_x[X_t \in A \mid \tau > t] P_x[\tau > t] + P_x[X_t \in A \mid \tau \leq t] (1 - P_x[\tau > t]) - \pi(A)
\]

\[
= (P_x[X_t \in A \mid \tau > t] - \pi(A)) P_x[\tau > t].
\]

Then since \(-1 \leq P_x[X_t \in A \mid \tau > t] - \pi(A) \leq 1\) we have

\[
|P_x[X_t \in A] - \pi(A)| = |P_x[X_t \in A \mid \tau > t] - \pi(A)| P_x[\tau > t] \leq P_x[\tau > t],
\]

for any \(A \subseteq \mathcal{I}\).
Strong Stationary Time

A *Strong Stationary Time* for a Markov Chain \((X_t)\) with stationary distribution \(\pi\) is a stopping time \(\tau\), possibly depending on the stating state \(x\), such that

\[ P_x[t = \tau, X_\tau = y] = P_x[t = \tau] \pi_y. \]

- Thus \(X_\tau\) has distribution \(\pi\) and is independent of \(\tau\).

Mixing from Strong Stationary Times

If \(\tau\) is a strong stationary time then for any \(x \in \mathcal{I}\),

\[ \left\| P_x^t - \pi \right\|_{tv} \leq P[\tau > t | X_0 = x]. \]

**Proof:** For any \(A \subseteq \mathcal{I}\) the difference \(P_x[X_t \in A] - \pi(A)\) is equal to

\[
P_x[X_t \in A | \tau > t] P_x[\tau > t] + P_x[X_t \in A | \tau \leq t] (1 - P_x[\tau > t]) - \pi(A)
= (P_x[X_t \in A | \tau > t] - \pi(A)) P_x[\tau > t].
\]

Then since \(-1 \leq P_x[X_t \in A | \tau > t] - \pi(A) \leq 1\) we have

\[
|P_x[X_t \in A] - \pi(A)| = |P_x[X_t \in A | \tau > t] - \pi(A)| P_x[\tau > t] \leq P_x[\tau > t],
\]

for any \(A \subset \mathcal{I}\). We can take \(\sup_{A \subset \mathcal{I}}\) to complete the result. \(\square\)
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{top}$ be one step after the first time when $B$ is on top of the deck.
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{top}$ be one step after the first time when $B$ is on top of the deck.

$\tau_{top}$ is a Strong Stationary time for the T-to-R chain.
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{top}$ be one step after the first time when $B$ is on top of the deck.

**Strong Stationary time for T-to-R**

$\tau_{top}$ is a Strong Stationary time for the T-to-R chain.

**Proof:** At any $t \geq 0$ all arrangements of the cards under $B$ are equally likely.
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{\text{top}}$ be one step after the first time when $B$ is on top of the deck.

$\tau_{\text{top}}$ is a Strong Stationary time for the T-to-R chain.

Proof: At any $t \geq 0$ all arrangements of the cards under $B$ are equally likely.

Induction: When $t = 0$, there are no cards under $B$. Suppose that the claim holds at time $t \geq 0$ with $k \geq 0$ cards under $B$. 

Lecture 4: Mixing and shuffling
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{top}$ be one step after the first time when $B$ is on top of the deck.

$\tau_{top}$ is a Strong Stationary time for the T-to-R chain.

Proof: At any $t \geq 0$ all arrangements of the cards under $B$ are equally likely.

Induction: When $t = 0$, there are no cards under $B$. Suppose that the claim holds at time $t \geq 0$ with $k \geq 0$ cards under $B$. Two cases for time $t + 1$: either the top card is placed under $B$, or it is placed above $B$. 

\[ P[X | \tau_{top} = t] = \frac{1}{n!} \]
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{top}$ be one step after the first time when $B$ is on top of the deck.

$\tau_{top}$ is a Strong Stationary time for the T-to-R chain.

Proof: At any $t \geq 0$ all arrangements of the cards under $B$ are equally likely.

Induction: When $t = 0$, there are no cards under $B$. Suppose that the claim holds at time $t \geq 0$ with $k \geq 0$ cards under $B$. Two cases for time $t + 1$: either the top card is placed under $B$, or it is placed above $B$.

Case 1 Hypothesis: all orderings of the $k$ cards already under $B$ are equiprobable. Top card is equally likely to be added to any of the $k + 1$ possible locations under $B$, so each of the $(k + 1)!$ arrangements is equiprobable.
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{\text{top}}$ be one step after the first time when $B$ is on top of the deck.

$\tau_{\text{top}}$ is a Strong Stationary time for the T-to-R chain.

**Proof:** At any $t \geq 0$ all arrangements of the cards under $B$ are equally likely.

**Induction:** When $t = 0$, there are no cards under $B$. Suppose that the claim holds at time $t \geq 0$ with $k \geq 0$ cards under $B$. Two cases for time $t + 1$: either the top card is placed under $B$, or it is placed above $B$.

**Case 1** Hypothesis: all orderings of the $k$ cards already under $B$ are equiprobable. Top card is equally likely to be added to any of the $k + 1$ possible locations under $B$, so each of the $(k + 1)!$ arrangements is equiprobable.

**Case 2** New card goes above $B$, so cards under $B$ remain in random order. □
Strong Stationary time for Top-to-Random Shuffling

- Let $B$ be the card at the bottom of the deck at $t = 0$.
- Let $\tau_{top}$ be one step after the first time when $B$ is on top of the deck.

$\tau_{top}$ is a Strong Stationary time for the T-to-R chain.

Proof: At any $t \geq 0$ all arrangements of the cards under $B$ are equally likely.

Induction: When $t = 0$, there are no cards under $B$. Suppose that the claim holds at time $t \geq 0$ with $k \geq 0$ cards under $B$. Two cases for time $t + 1$: either the top card is placed under $B$, or it is placed above $B$.

Case 1 Hypothesis: all orderings of the $k$ cards already under $B$ are equiprobable. Top card is equally likely to be added to any of the $k + 1$ possible locations under $B$, so each of the $(k + 1)!$ arrangements is equiprobable.

Case 2 New card goes above $B$, so cards under $B$ remain in random order.

Thus at time $\tau_{top} - 1$ $B$ sits on the top of a uniform permutation of $[n]\setminus\{B\}$, then we place $B$ in at random so $\Pr[X_{\tau_{top}} \mid \tau_{top} = t] = 1/n!$. 

Lecture 4: Mixing and shuffling
Top-to-Random Shuffle

Mixing of Top-to-Random Shuffle

Let $\epsilon > 0$ then for the top to random shuffle, $\tau(\epsilon) \leq n \ln n + O(n)$. 

Since the state space $\Sigma_n$ has size $n!$, we have $t_{\text{mix}} \approx \ln (|\Sigma_n|)$. 

Lecture 4: Mixing and shuffling
Top-to-Random Shuffle

<table>
<thead>
<tr>
<th>Mixing of Top-to-Random Shuffle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $\epsilon &gt; 0$ then for the top to random shuffle, $\tau(\epsilon) \leq n \ln n + O(n)$.</td>
</tr>
</tbody>
</table>

**Proof:** For $1 \leq k \leq n - 1$ the time between the $(k - 1)^{th}$ and $k^{th}$ cards going under $B$ is distributed $\text{Geo}(k/n)$. 

---

Lecture 4: Mixing and shuffling
Top-to-Random Shuffle

Mixing of Top-to-Random Shuffle

Let $\epsilon > 0$ then for the top to random shuffle, $\tau(\epsilon) \leq n \ln n + O(n)$.

Proof: For $1 \leq k \leq n - 1$ the time between the $(k - 1)^{th}$ and $k^{th}$ cards going under $B$ is distributed $\text{Geo}(k/n)$. This means that $\tau_{\text{top}}$ is distributed the same as the number of balls thrown until no bin is empty in “Balls and Bins”.
Top-to-Random Shuffle

Mixing of Top-to-Random Shuffle

Let $\epsilon > 0$ then for the top to random shuffle, $\tau(\epsilon) \leq n \ln n + O(n)$.

Proof: For $1 \leq k \leq n - 1$ the time between the $(k - 1)^{th}$ and $k^{th}$ cards going under $B$ is distributed $\text{Geo}(k/n)$. This means that $\tau_{\text{top}}$ is distributed the same as the number of balls thrown until no bin is empty in “Balls and Bins”. Thus

$$P[\tau > n \ln n + Cn] \leq P[\exists \text{ empty bin after } n \ln n + Cn \text{ balls}] \leq e^{-C}.$$
Top-to-Random Shuffle

Let $\epsilon > 0$ then for the top to random shuffle, $\tau(\epsilon) \leq n \ln n + O(n)$.

Proof: For $1 \leq k \leq n - 1$ the time between the $(k - 1)^{th}$ and $k^{th}$ cards going under $B$ is distributed $\text{Geo}(k/n)$. This means that $\tau_{\text{top}}$ is distributed the same as the number of balls thrown until no bin is empty in “Balls and Bins”. Thus

$$\Pr[\tau > n \ln n + Cn] \leq \Pr[\exists \text{ empty bin after } n \ln n + Cn \text{ balls}] \leq e^{-C}.$$

Taking $C$ large enough such that $e^{-C} \leq \epsilon$ yields the result. \qed
Top-to-Random Shuffle

Let $\epsilon > 0$ then for the top to random shuffle, $\tau(\epsilon) \leq n \ln n + O(n)$.

Proof: For $1 \leq k \leq n - 1$ the time between the $(k - 1)^{th}$ and $k^{th}$ cards going under $B$ is distributed $\text{Geo}(k/n)$. This means that $\tau_{\text{top}}$ is distributed the same as the number of balls thrown until no bin is empty in "Balls and Bins". Thus

$$P[\tau > n \ln n + Cn] \leq P[\exists \text{ empty bin after } n \ln n + Cn \text{ balls}] \leq e^{-C}.$$ 

Taking $C$ large enough such that $e^{-C} \leq \epsilon$ yields the result. □

- Since the state space $\Sigma_n$ has size $n!$, we have

$$t_{\text{mix}} \approx \ln (|\Sigma_n|).$$
Realistic Shuffling - Riffle Shuffle

Split the deck into two piles \(L, R\) where \(L\) is the first \(\frac{n}{2}\) cards and \(R\) is the rest. Form a new pile iteratively by adding a card from \(L\) with probability \(\frac{\ell}{\ell + r}\), where \(\ell, r\) sizes of \(L, R\) at that time, or otherwise from \(R\) with probability \(\frac{r}{\ell + r}\).

For the Riffle shuffle \(t\) mix \(\leq 2 \log_2(\frac{4n}{3})\).

Riffle is fast, same state space \(\Sigma_n\) as T-to-R however this time \(t\) mix \(\approx \ln \ln (|\Sigma_n|)\).

May have heard “7 riffle shuffles is enough”. \(t \leq 4, 5, 6, 7, 8, 9\).
Realistic Shuffling - Riffle Shuffle

- Split the deck into two piles $L, R$ where $L$ is the first $\text{Bin}(n, 1/2)$ cards and $R$ is the rest.
Realistic Shuffling - Riffle Shuffle

**Riffle Shuffle**

- Split the deck into two piles $L, R$ where $L$ is the first $\text{Bin}(n, 1/2)$ cards and $R$ is the rest.
- Form a new pile iteratively by adding a card from $L$ with probability $\ell/(\ell + r)$, where $\ell, r$ sizes of $L, R$ at that time, or otherwise from $R$ with probability $r/(\ell + r)$. 

Riffle is fast

Same state space $\Sigma_n$ as T-to-R however this time $t_{\text{mix}} \approx \ln \ln (|\Sigma_n|)$. May have heard “7 riffle shuffles is enough”. $t \leq 4, 5, 6, 7, 8, 9, \Delta(t) = 1.00$. 92. 61. 33. 17. 09.
Realistic Shuffling - Riffle Shuffle

- **Riffle Shuffle**
  - Split the deck into two piles $L, R$ where $L$ is the first $\text{Bin}(n, 1/2)$ cards and $R$ is the rest.
  - Form a new pile iteratively by adding a card from $L$ with probability $\frac{\ell}{\ell + r}$, where $\ell, r$ sizes of $L, R$ at that time, or otherwise from $R$ with probability $\frac{r}{\ell + r}$.

**Riffle is fast**
For the Riffle shuffle $t_{\text{mix}} \leq 2 \log_2(4n/3)$.
Realistic Shuffling - Riffle Shuffle

Riffle Shuffle

- Split the deck into two piles $L, R$ where $L$ is the first $\text{Bin}(n, 1/2)$ cards and $R$ is the rest.
- Form a new pile iteratively by adding a card from $L$ with probability $\ell/(r + \ell)$, where $\ell, r$ sizes of $L, R$ at that time, or otherwise from $R$ with probability $r/(\ell + r)$.

Riffle is fast

For the Riffle shuffle $t_{\text{mix}} \leq 2 \log_2(4n/3)$.

- Same state space $\Sigma_n$ as T-to-R however this time
  \[ t_{\text{mix}} \approx \ln \ln (|\Sigma_n|). \]
Realistic Shuffling - Riffle Shuffle

Riffle Shuffle

- Split the deck into two piles $L, R$ where $L$ is the first $\text{Bin}(n, 1/2)$ cards and $R$ is the rest.
- Form a new pile iteratively by adding a card from $L$ with probability $\frac{\ell}{\ell + r}$, where $\ell, r$ sizes of $L, R$ at that time, or otherwise from $R$ with probability $\frac{r}{\ell + r}$.

Riffle is fast

For the Riffle shuffle $t_{\text{mix}} \leq 2 \log_2(4n/3)$.

- Same state space $\Sigma_n$ as T-to-R however this time $t_{\text{mix}} \approx \ln \ln (|\Sigma_n|)$.
- May have heard “7 riffle shuffles is enough”.
Realistic Shuffling - Riffle Shuffle

**Riffle Shuffle**

- Split the deck into two piles $L, R$ where $L$ is the first $\text{Bin}(n, 1/2)$ cards and $R$ is the rest.
- Form a new pile iteratively by adding a card from $L$ with probability $\ell/(r + \ell)$, where $\ell, r$ sizes of $L, R$ at that time, or otherwise from $R$ with probability $r/(\ell + r)$.

**Riffle is fast**

For the Riffle shuffle $t_{\text{mix}} \leq 2 \log_2(4n/3)$.

- Same state space $\Sigma_n$ as T-to-R however this time
  
  \[ t_{\text{mix}} \approx \ln \ln (|\Sigma_n|). \]

- May have heard “7 riffle shuffles is enough”.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\leq 4$</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta(t)$</td>
<td>1.00</td>
<td>.92</td>
<td>.61</td>
<td>.33</td>
<td>.17</td>
<td>.09</td>
</tr>
</tbody>
</table>
Outline

Shuffling and Strong Stationary Times

Covertime

$s - t$ Connectivity

2-Sat
Covertime

The Cover time $t_{\text{cov}}(G)$ of a graph $G = (V, E)$ is given by

$$t_{\text{cov}}(G) = \max_{v \in V} E_v[\tau_{\text{cov}}]$$

where

$$\tau_{\text{cov}} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.
The *Cover time* $t_{\text{cov}}(G)$ of a graph $G = (V, E)$ is given by

$$t_{\text{cov}}(G) = \max_{v \in V} \mathbb{E}_v[\tau_{\text{cov}}] \quad \text{where} \quad \tau_{\text{cov}} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$ 

- Expected time for a walk to visit the whole graph from worst case start.

Example:

![Graph Image]

$|V| = 6$
The Cover time $t_{cov}(G)$ of a graph $G = (V, E)$ is given by

$$t_{cov}(G) = \max_{v \in V} E_v[\tau_{cov}] \quad \text{where} \quad \tau_{cov} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$  

- Expected time for a walk to visit the whole graph from worst case start.

### Example:

$$|V| = 6$$
The *Cover time* $t_{\text{cov}}(G)$ of a graph $G = (V, E)$ is given by

$$t_{\text{cov}}(G) = \max_{v \in V} E_v[\tau_{\text{cov}}] \quad \text{where} \quad \tau_{\text{cov}} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.

Example:

![Graph Diagram]

$|V| = 6$
The **Cover time** \( t_{\text{cov}} (G) \) of a graph \( G = (V, E) \) is given by

\[
t_{\text{cov}} (G) = \max_{v \in V} \mathbb{E}_v [\tau_{\text{cov}}] \quad \text{where} \quad \tau_{\text{cov}} := \inf \left\{ t : \bigcup_{i=0}^{t} X_t = V \right\}.
\]

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

\[
|V| = 6
\]
Covertime

The **Cover time** $t_{cov}(G)$ of a graph $G = (V, E)$ is given by

$$t_{cov}(G) = \max_{v \in V} \mathbb{E}_v[\tau_{cov}]$$

where

$$\tau_{cov} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

![Graph Diagram]

$|V| = 6$
The Cover time $t_{cov}(G)$ of a graph $G = (V, E)$ is given by

$$t_{cov}(G) = \max_{v \in V} E_v[\tau_{cov}]$$

where

$$\tau_{cov} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_i\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.

Example:
The **Cover time** $t_{cov}(G)$ of a graph $G = (V, E)$ is given by

$$t_{cov}(G) = \max_{v \in V} \mathbb{E}_v[\tau_{cov}]$$

where

$$\tau_{cov} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

![Graph](image)

$|V| = 6$
The **Cover time** $t_{\text{cov}}(G)$ of a graph $G = (V, E)$ is given by

$$t_{\text{cov}}(G) = \max_{v \in V} \mathbb{E}_v[\tau_{\text{cov}}]$$

where $\tau_{\text{cov}} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}$.

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

```
\begin{center}
\begin{tikzpicture}
  \node[vertex] (a) at (0,0) {a};
  \node[vertex] (b) at (1,1) {b};
  \node[vertex] (c) at (1,-1) {c};
  \node[vertex] (d) at (-1,-1) {d};
  \node[vertex] (e) at (-1,0) {e};
  \node[vertex] (f) at (1,-2) {f};
  \draw (a) -- (b) -- (c) -- (d) -- (e) -- (f) -- (a);
\end{tikzpicture}
\end{center}
```

$|V| = 6$
The **Cover time** $t_{cov}(G)$ of a graph $G = (V, E)$ is given by

$$t_{cov}(G) = \max_{v \in V} E_v[\tau_{cov}]$$

where

$$\tau_{cov} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

![Graph diagram]

| $V$ | 6 |
The *Cover time* $t_{\text{cov}}(G)$ of a graph $G = (V, E)$ is given by

$$t_{\text{cov}}(G) = \max_{v \in V} \mathbf{E}_v[\tau_{\text{cov}}]$$

where

$$\tau_{\text{cov}} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_i\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

\[ |V| = 6 \]
Covertime

The **Cover time** $t_{cov}(G)$ of a graph $G = (V, E)$ is given by

$$t_{cov}(G) = \max_{v \in V} E_v[\tau_{cov}] \quad \text{where} \quad \tau_{cov} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}.$$

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

![Graph example](image)

$|V| = 6$
The Cover time $t_{cov}(G)$ of a graph $G = (V, E)$ is given by

$$t_{cov}(G) = \max_{v \in V} E_v[\tau_{cov}]$$

where $\tau_{cov} := \inf \left\{ t : \bigcup_{i=0}^{t} \{X_t\} = V \right\}$. 

- Expected time for a walk to visit the whole graph from worst case start.

**Example:**

$|V| = 6$

$\tau_{cov}(G) = 9$. 
Let $P$ be the SRW on a connected graph $G$, then $\pi_x = d(x)/2|E|$. 
Let \( P \) be the SRW on a connected graph \( G \), then \( \pi_x = \frac{d(x)}{2|E|} \).

**Proof:** Note that \( \sum_{x \in V} \pi = 1 \) and that for any \( x \in V \)

\[
(\pi P)_x = \sum_{y \in V} \pi_y P_{y,x} = \sum_{y \in d(x)} \frac{d(y)}{2|E|} \frac{1}{d(y)} = \frac{d(x)}{2|E|}.
\]
Let $P$ be the SRW on a connected graph $G$, then $\pi_x = \frac{d(x)}{2|E|}$.

Proof: Note that $\sum_{x \in V} \pi = 1$ and that for any $x \in V$

$$(\pi P)_x = \sum_{y \in V} \pi_y P_{y,x} = \sum_{y \in d(x)} \frac{d(y)}{2|E|} \frac{1}{d(y)} = \frac{d(x)}{2|E|}.$$  

Crossing time of an edge

Let $xy \in E(G)$ where $G$ is any finite connected graph then $h_{x,y} \leq 2|E|$. 

Lecture 4: Mixing and shuffling
Let $P$ be the SRW on a connected graph $G$, then $\pi_x = d(x) / 2|E|$.

**Proof:** Note that $\sum_{x \in V} \pi = 1$ and that for any $x \in V$

$$(\pi P)_x = \sum_{y \in V} \pi_y P_{y,x} = \sum_{y \in d(x)} \frac{d(y)}{2|E|} \frac{1}{d(y)} = \frac{d(x)}{2|E|}.$$ 

---

**Crossing time of an edge**

Let $xy \in E(G)$ where $G$ is any finite connected graph then $h_{x,y} \leq 2|E|$.

**Proof:** Since the SRW on any connected finite graph is irreducible we know

$$E_y[\tau^+_y] = \frac{1}{\pi_y} = \frac{2|E|}{d(y)}.$$ 

---

Lecture 4: Mixing and shuffling
Let \( P \) be the SRW on a connected graph \( G \), then \( \pi_x = d(x) / 2|E| \).

**Proof:** Note that \( \sum_{x \in V} \pi_x = 1 \) and that for any \( x \in V \)

\[
(\pi P)_x = \sum_{y \in V} \pi_y P_{y,x} = \sum_{y \in d(x)} \frac{d(y)}{2|E|} \frac{1}{d(y)} = \frac{d(x)}{2|E|}.
\]

**Crossing time of an edge**

Let \( xy \in E(G) \) where \( G \) is any finite connected graph then \( h_{x,y} \leq 2|E| \).

**Proof:** Since the SRW on any connected finite graph is irreducible we know

\[
E_y[\tau_y^+] = \frac{1}{\pi_y} = \frac{2|E|}{d(y)}.
\]

By the Markov property we have

\[
\frac{2|E|}{d(y)} = E_y[\tau_y^+] = 1 + \sum_{z \sim y} \frac{h_{z,y}}{d(y)}.
\]
Let $P$ be the SRW on a connected graph $G$, then $\pi_x = d(x) / 2|E|$.

Proof: Note that $\sum_{x \in V} \pi_x = 1$ and that for any $x \in V$

$$\left(\pi P\right)_x = \sum_{y \in V} \pi_y P_{y,x} = \sum_{y \in d(x)} \frac{d(y)}{2|E|} \frac{1}{d(y)} = \frac{d(x)}{2|E|}. \quad \square$$

Crossing time of an edge

Let $xy \in E(G)$ where $G$ is any finite connected graph then $h_{x,y} \leq 2|E|$.

Proof: Since the SRW on any connected finite graph is irreducible we know

$$E_y[\tau_y^+] = \frac{1}{\pi_y} = \frac{2|E|}{d(y)}.$$ 

By the Markov property we have

$$\frac{2|E|}{d(y)} = E_y[\tau_y^+] = 1 + \sum_{z \sim y} \frac{h_{z,y}}{d(y)}.$$ 

It follows that $\sum_{z \sim y} h_{z,y} \leq d(y) \left(E_y[\tau_y^+] - 1\right)$.
Let $P$ be the SRW on a connected graph $G$, then $\pi_x = d(x)/2|E|$.

**Proof:** Note that $\sum_{x \in V} \pi = 1$ and that for any $x \in V$

$$(\pi P)_x = \sum_{y \in V} \pi_y P_{y,x} = \sum_{y \in d(x)} \frac{d(y)}{2|E|} \frac{1}{d(y)} = \frac{d(x)}{2|E|}.$$ 

Crossing time of an edge

Let $xy \in E(G)$ where $G$ is any finite connected graph then $h_{x,y} \leq 2|E|$.

**Proof:** Since the SRW on any connected finite graph is irreducible we know

$$\mathbb{E}_y [\tau^+_y] = \frac{1}{\pi_y} = \frac{2|E|}{d(y)}.$$ 

By the Markov property we have

$$\frac{2|E|}{d(y)} = \mathbb{E}_y [\tau^+_y] = 1 + \sum_{z \sim y} \frac{h_{z,y}}{d(y)}.$$ 

It follows that $\sum_{z \sim y} h_{z,y} \leq d(y) \left( \mathbb{E}_y [\tau^+_y] - 1 \right)$ and thus

$$h_{x,y} \leq \sum_{z \sim y} h_{z,y} \leq d(y) \cdot \left( \frac{2|E|}{d(y)} - 1 \right) \leq 2|E|.$$ 

For any connected graph $t_{cov}(G) \leq 4n|E| \leq 2n^3$. 

Proof: Any connected graph has a spanning tree $T$ with $n-1$ edges. Choose any root $v_0$ for $T$ and fix a tour $v_0, \ldots, v_{2n-2}$ on $T$ which visits every vertex and returns to the root. The Covertime of $G$ is at most the expected length of this tour (from worst case start vertex). Thus $t_{cov}(G) \leq 2n-3 \sum_{i=0}^{n-2} h_{v_i, v_{i+1}} \leq \sum_{xy \in E(T)} (h_{xy} + h_{yx}) \leq 2 \sum_{xy \in E(T)} 2|E| \leq 4n|E|$, since for any $xy \in E$ we have $h_{x,y} \leq 2|E|$. □
For any connected graph $t_{cov}(G) \leq 4n|E| \leq 2n^3$.

**Proof:** Any connected graph has a spanning tree $T$ with $n - 1$ edges.
For any connected graph $t_{cov}(G) \leq 4n|E| \leq 2n^3$.

**Proof:** Any connected graph has a spanning tree $T$ with $n - 1$ edges. Choose any root $v_0$ for $T$ and fix a tour $v_0, \ldots, v_{2n-2}$ on $T$ which visits every vertex and returns to the root.
For any connected graph \( t_{\text{cov}}(G) \leq 4n|E| \leq 2n^3 \).

**Proof:** Any connected graph has a spanning tree \( T \) with \( n - 1 \) edges. Choose any root \( v_0 \) for \( T \) and fix a tour \( v_0, \ldots, v_{2n-2} \) on \( T \) which visits every vertex and returns to the root.

The Covertime of \( G \) is at most the expected length of this tour (from worst case start vertex).
For any connected graph \( t_{\text{cov}}(G) \leq 4n|E| \leq 2n^3 \).

**Proof:** Any connected graph has a spanning tree \( T \) with \( n - 1 \) edges. Choose any root \( v_0 \) for \( T \) and fix a tour \( v_0, \ldots, v_{2n-2} \) on \( T \) which visits every vertex and returns to the root.

The Covertime of \( G \) is at most the expected length of this tour (from worst case start vertex). Thus

\[
t_{\text{cov}}(G) \leq \sum_{i=0}^{2n-3} h_{v_i, v_{i+1}} = \sum_{xy \in E(T)} (h_{xy} + h_{yx}) \leq 2 \sum_{xy \in E(T)} 2|E| \leq 4n|E|,
\]

since for any \( xy \in E \) we have \( h_{x,y} \leq 2|E| \).

\( \square \)
For any connected graph $t_{\text{cov}}(G) \leq 4n|E| \leq 2n^3$.

**Proof:** Any connected graph has a spanning tree $T$ with $n - 1$ edges. Choose any root $v_0$ for $T$ and fix a tour $v_0, \ldots, v_{2n-2}$ on $T$ which visits every vertex and returns to the root.

The Covertime of $G$ is at most the expected length of this tour (from worst case start vertex). Thus

$$t_{\text{cov}}(G) \leq \sum_{i=0}^{2n-3} h_{v_i, v_{i+1}} = \sum_{xy \in E(T)} (h_{xy} + h_{yx}) \leq 2 \sum_{xy \in E(T)} 2|E| \leq 4n|E|,$$

since for any $xy \in E$ we have $h_{x,y} \leq 2|E|$.

---

Matthews bound

For any graph $G$ we have

$$t_{\text{cov}}(G) \leq \left( \sum_{m=1}^{n-1} \frac{1}{m} \right) \cdot \max_{x,y \in V} h_{x,y} \approx (\ln n) \cdot \max_{x,y \in V} h_{x,y}.$$
The $n$-path $P_n$ is the graph with $V(P_n) = [n]$ and $E(P_n) = \{ij : j = i + 1\}$.

**Proposition**

For the SRW on $P_n$ we have $h_{k,n} = n^2 - k^2$, for any $0 \leq k \leq n$. 

The cover time of the path on $n$ vertices is $n^2$. 

Lecture 4: Mixing and shuffling
Random Walk on a path

The $n$-path $P_n$ is the graph with $V(P_n) = [n]$ and $E(P_n) = \{ij : j = i + 1\}$.

**Proposition**

For the SRW on $P_n$ we have $h_{k,n} = n^2 - k^2$, for any $0 \leq k \leq n$.

**Proof:** Let $f_k = h_{k,n}$ and observe that $f_n = 0$. 
Random Walk on a path

The $n$-path $P_n$ is the graph with $V(P_n) = [n]$ and $E(P_n) = \{ij : j = i + 1\}$.

**Proposition**

For the SRW on $P_n$ we have $h_{k,n} = n^2 - k^2$, for any $0 \leq k \leq n$.

**Proof:** Let $f_k = h_{k,n}$ and observe that $f_n = 0$. By the Markov property

$$f_0 = 1 + f_1 \quad \text{and} \quad f_k = 1 + \frac{f_{k-1}}{2} + \frac{f_{k+1}}{2} \quad \text{for} \ 1 \leq k \leq n - 1.$$
Random Walk on a path

The $n$-path $P_n$ is the graph with $V(P_n) = [n]$ and $E(P_n) = \{ij : j = i + 1\}$.

**Proposition**

For the SRW on $P_n$ we have $h_{k,n} = n^2 - k^2$, for any $0 \leq k \leq n$.

**Proof:** Let $f_k = h_{k,n}$ and observe that $f_n = 0$. By the Markov property

$$f_0 = 1 + f_1 \quad \text{and} \quad f_k = 1 + \frac{f_{k-1}}{2} + \frac{f_{k+1}}{2} \quad \text{for } 1 \leq k \leq n - 1.$$ 

System of $n$ independent equations in $n$ unknowns so has a unique solution.
Random Walk on a path

The $n$-path $P_n$ is the graph with $V(P_n) = [n]$ and $E(P_n) = \{ij : j = i + 1\}$.

Proposition

For the SRW on $P_n$ we have $h_{k,n} = n^2 - k^2$, for any $0 \leq k \leq n$.

Proof: Let $f_k = h_{k,n}$ and observe that $f_n = 0$. By the Markov property

$$f_0 = 1 + f_1 \quad \text{and} \quad f_k = 1 + \frac{f_{k-1}}{2} + \frac{f_{k+1}}{2} \quad \text{for } 1 \leq k \leq n-1.$$ 

System of $n$ independent equations in $n$ unknowns so has a unique solution.

Thus it suffices to check that $f_k = n^2 - k^2$ satisfies the above.
Random Walk on a path

The $n$-path $P_n$ is the graph with $V(P_n) = [n]$ and $E(P_n) = \{ij : j = i + 1\}$.

**Proposition**

For the SRW on $P_n$ we have $h_{k,n} = n^2 - k^2$, for any $0 \leq k \leq n$.

**Proof:** Let $f_k = h_{k,n}$ and observe that $f_n = 0$. By the Markov property

$$f_0 = 1 + f_1 \quad \text{and} \quad f_k = 1 + \frac{f_{k-1}}{2} + \frac{f_{k+1}}{2} \quad \text{for } 1 \leq k \leq n - 1.$$ 

System of $n$ independent equations in $n$ unknowns so has a unique solution. Thus it suffices to check that $f_k = n^2 - k^2$ satisfies the above. Indeed

$$f_n = n^2 - n^2 = 0, \quad f_0 = 1 + f_1 = 1 + n^2 - 1^2 = n^2,$$

and for any $1 \leq k \leq n - 1$ we have,

$$f_k = 1 + \frac{n^2 - (k - 1)^2}{2} + \frac{n^2 - (k + 1)^2}{2} = n^2 - k^2.$$ 

\[ \square \]
Random Walk on a path

The $n$-path $P_n$ is the graph with $V(P_n) = [n]$ and $E(P_n) = \{ij : j = i + 1\}$.

---

**Proposition**

For the SRW on $P_n$ we have $h_{k,n} = n^2 - k^2$, for any $0 \leq k \leq n$.

**Proof:** Let $f_k = h_{k,n}$ and observe that $f_n = 0$. By the Markov property

$$f_0 = 1 + f_1 \quad \text{and} \quad f_k = 1 + \frac{f_{k-1}}{2} + \frac{f_{k+1}}{2} \quad \text{for} \ 1 \leq k \leq n-1.$$  

System of $n$ independent equations in $n$ unknowns so has a unique solution. Thus it suffices to check that $f_k = n^2 - k^2$ satisfies the above. Indeed

$$f_n = n^2 - n^2 = 0, \quad f_0 = 1 + f_1 = 1 + n^2 - 1^2 = n^2,$$

and for any $1 \leq k \leq n-1$ we have,

$$f_k = 1 + \frac{n^2 - (k - 1)^2}{2} + \frac{n^2 - (k + 1)^2}{2} = n^2 - k^2.$$

---

**Covertime of the Path**

The cover time of the path on $n$ vertices is $n^2$. 

---
Outline

Shuffling and Strong Stationary Times

Covertime

$s - t$ Connectivity

2-Sat
Given: Undirected graph $G = (V, E)$ and $s, t \in V$

Goal: Determine if $s$ is connected by a path to $t$.

$s - t$ Connectivity Problem

Start a random walk from $s$.
If the walk hits $t$ within $4n^3$ steps, return True. O/W return False.

$s - t$ Connectivity Algorithm
The $s - t$ Connectivity Algorithm runs in time $4n^3$ and returns the correct answer w.p. at least $1/2$ and never returns True incorrectly.

Proposition
Proof: By Markov inequality if there is a path to $t$ we will find it w.p. $\geq 1/2$.

Running this $T$ times gives the correct answer with probability $\geq 1 - 1/2^T$.

Only uses logspace.
s \rightarrow t \text{ Connectivity}

\textbf{s \rightarrow t Connectivity Problem}

- Given: Undirected graph \( G = (V, E) \) and \( s, t \in V \)
**s − t Connectivity**

**s − t Connectivity Problem**
- **Given:** Undirected graph \( G = (V, E) \) and \( s, t \in V \)
- **Goal:** Determine if \( s \) is connected by a path to \( t \).
**$s - t$ Connectivity**

---

**$s - t$ Connectivity Problem**

- **Given:** Undirected graph $G = (V, E)$ and $s, t \in V$
- **Goal:** Determine if $s$ is connected by a path to $t$.

---

**$s - t$ Connectivity Algorithm**

"Start a random walk from $s$.
If the walk hits $t$ within $4n^3$ steps, return True. O/W return False."
**s − t Connectivity**

---

**s − t Connectivity Problem**

- **Given**: Undirected graph $G = (V, E)$ and $s, t \in V$
- **Goal**: Determine if $s$ is connected by a path to $t$.

---

**s − t Connectivity Algorithm**

- Start a random walk from $s$. 

---
**$s - t$ Connectivity**

**$s - t$ Connectivity Problem**

- **Given:** Undirected graph $G = (V, E)$ and $s, t \in V$
- **Goal:** Determine if $s$ is connected by a path to $t$.

**$s - t$ Connectivity Algorithm**

- Start a random walk from $s$.
- If the walk hits $t$ within $4n^3$ steps, return True. O/W return False.
**s – t Connectivity**

---

**s – t Connectivity Problem**

- **Given:** Undirected graph \( G = (V, E) \) and \( s, t \in V \)
- **Goal:** Determine if \( s \) is connected by a path to \( t \).

---

**s – t Connectivity Algorithm**

- Start a random walk from \( s \).
- If the walk hits \( t \) within \( 4n^3 \) steps, return \text{True}\. O/W return \text{False}\. 

---

**Proposition**

The **s – t Connectivity Algorithm** runs in time \( 4n^3 \) and returns the correct answer w.p. at least 1/2 and never returns \text{True} incorrectly.
**s − t Connectivity**

### s − t Connectivity Problem

- **Given:** Undirected graph $G = (V, E)$ and $s, t \in V$
- **Goal:** Determine if $s$ is connected by a path to $t$.

### s − t Connectivity Algorithm

- Start a random walk from $s$.
- If the walk hits $t$ within $4n^3$ steps, return **True**. O/W return **False**.

### Proposition

The **s − t Connectivity Algorithm** runs in time $4n^3$ and returns the correct answer w.p. at least $1/2$ and never returns **True** incorrectly.

**Proof:** By Markov inequality if there is a path to $t$ we will find it w.p. $\geq 1/2$. □
**s − t Connectivity**

**s − t Connectivity Problem**
- **Given:** Undirected graph $G = (V, E)$ and $s, t \in V$
- **Goal:** Determine if $s$ is connected by a path to $t$.

**s − t Connectivity Algorithm**
- Start a random walk from $s$.
- If the walk hits $t$ within $4n^3$ steps, return **True**. O/W return **False**.

**Proposition**
The *s − t Connectivity Algorithm* runs in time $4n^3$ and returns the correct answer w.p. at least $1/2$ and never returns **True** incorrectly.

**Proof:** By Markov inequality if there is a path to $t$ we will find it w.p. $\geq 1/2$. □
- Running this $T$ times gives the correct answer with probability $\geq 1 − 1/2^T$. 

Lecture 4: Mixing and shuffling
**s – t Connectivity**

---

### s – t Connectivity Problem

- **Given:** Undirected graph \( G = (V, E) \) and \( s, t \in V \)
- **Goal:** Determine if \( s \) is connected by a path to \( t \).

---

### s – t Connectivity Algorithm

- Start a random walk from \( s \).
- If the walk hits \( t \) within \( 4n^3 \) steps, return **True**. O/W return **False**.

---

### Proposition

The **s – t Connectivity Algorithm** runs in time \( 4n^3 \) and returns the correct answer w.p. at least \( 1/2 \) and never returns **True** incorrectly.

**Proof:** By Markov inequality if there is a path to \( t \) we will find it w.p. \( \geq 1/2 \). ☐

- Running this \( T \) times gives the correct answer with probability \( \geq 1 – 1/2^T \).
- Only uses logspace.
Outline

Shuffling and Strong Stationary Times

Covertime

$s - t$ Connectivity

2-Sat
A *Satisfiability (SAT)* formula is a logical expression that’s the conjunction (AND) of a set of *Clauses*, where a clause is the disjunction (OR) of *Literals*. 

Example:

SAT: 

\[(x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor x_3) \land (x_4 \lor x_1)\]

Solution: 

\[x_1 = \text{True}, \quad x_2 = \text{False}, \quad x_3 = \text{False}, \quad \text{and} \quad x_4 = \text{True}.\]
A Satisfiability (SAT) formula is a logical expression that’s the conjunction (AND) of a set of Clauses, where a clause is the disjunction (OR) of Literals.

A Solution to a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied.
A *Satisfiability (SAT)* formula is a logical expression that’s the conjunction (AND) of a set of *Clauses*, where a clause is the disjunction (OR) of *Literals*.

A *Solution* to a SAT formula is an assignment of the variables to the values *True* and *False* so that all the clauses are satisfied.

Example:

\[
\text{SAT: } (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor \overline{x}_1)
\]
A **Satisfiability (SAT)** formula is a logical expression that’s the conjunction (AND) of a set of **Clauses**, where a clause is the disjunction (OR) of **Literals**.

A **Solution** to a SAT formula is an assignment of the variables to the values **True** and **False** so that all the clauses are satisfied.

**Example:**

\[
\text{SAT: } (x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})
\]

**Solution:** \( x_1 = \text{True}, \ x_2 = \text{False}, \ x_3 = \text{False} \) and \( x_4 = \text{True} \).
SAT Problems

A Satisfiability (SAT) formula is a logical expression that's the conjunction (AND) of a set of Clauses, where a clause is the disjunction (OR) of Literals.

A Solution to a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied.

Example:

\[
\text{SAT: } \left( x_1 \lor \overline{x_2} \lor \overline{x_3} \right) \land \left( \overline{x_1} \lor x_3 \right) \land \left( x_1 \lor x_2 \lor x_4 \right) \land \left( x_4 \lor \overline{x_3} \right) \land \left( x_4 \lor \overline{x_1} \right)
\]

\[
\text{Solution: } x_1 = \text{True, } x_2 = \text{False, } x_3 = \text{False } \text{and } x_4 = \text{True.}
\]

- If each clause has \( k \) literals we call the problem \( k\text{-SAT} \).
A **Satisfiability (SAT)** formula is a logical expression that’s the conjunction (AND) of a set of **Clauses**, where a clause is the disjunction (OR) of **Literals**.

A **Solution** to a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied.

**Example:**

\[
\text{SAT: } (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor \overline{x}_1)
\]

**Solution:** \(x_1 = \text{True}, \ x_2 = \text{False}, \ x_3 = \text{False} \) and \(x_4 = \text{True}\).

- If each clause has \(k\) literals we call the problem **\(k\)-SAT**.
- In general, determining if a SAT formula has a solution is NP-hard.
SAT Problems

A Satisfiability (SAT) formula is a logical expression that’s the conjunction (AND) of a set of Clauses, where a clause is the disjunction (OR) of Literals.

A Solution to a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied.

Example:

\[
\text{SAT: } (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor \overline{x}_1)
\]

\[
\text{Solution: } x_1 = \text{True}, \quad x_2 = \text{False}, \quad x_3 = \text{False} \quad \text{and} \quad x_4 = \text{True}.
\]

- If each clause has \( k \) literals we call the problem \( k\text{-SAT} \).
- In general, determining if a SAT formula has a solution is NP-hard
- In practice solvers are fast and used to great effect
SAT Problems

A **Satisfiability (SAT)** formula is a logical expression that’s the conjunction (AND) of a set of **Clauses**, where a clause is the disjunction (OR) of **Literals**.

A **Solution** to a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied.

Example:

\[
\text{SAT: } (x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})
\]

**Solution:** \(x_1 = \text{True, } \ x_2 = \text{False, } \ x_3 = \text{False} \) and \(x_4 = \text{True.}\)

- If each clause has \(k\) literals we call the problem **\(k\)-SAT**.
- In general, determining if a SAT formula has a solution is NP-hard
- In practice solvers are fast and used to great effect
- A huge amount of problems can be posed as a SAT:
A **Satisfiability (SAT)** formula is a logical expression that’s the conjunction (AND) of a set of **Clauses**, where a clause is the disjunction (OR) of **Literals**.

A **Solution** to a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied.

**Example:**

\[
\text{SAT: } (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor \overline{x}_1)
\]

**Solution:** \(x_1 = \text{True, } x_2 = \text{False, } x_3 = \text{False and } x_4 = \text{True.}\)

- If each clause has \(k\) literals we call the problem **\(k\)-SAT**.
- In general, determining if a SAT formula has a solution is NP-hard.
- In practice solvers are fast and used to great effect.
- A huge amount of problems can be posed as a SAT:
  - \(\rightarrow\) Model Checking and hardware/software verification
A *Satisfiability (SAT)* formula is a logical expression that’s the conjunction (AND) of a set of *Clauses*, where a clause is the disjunction (OR) of *Literals*.

A *Solution* to a SAT formula is an assignment of the variables to the values *True* and *False* so that all the clauses are satisfied.

**Example:**

SAT: \((x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})\)

Solution: \(x_1 = \text{True}, \ x_2 = \text{False}, \ x_3 = \text{False} \quad \text{and} \quad x_4 = \text{True}.

- If each clause has \(k\) literals we call the problem *\(k\)-SAT*.
- In general, determining if a SAT formula has a solution is NP-hard.
- In practice solvers are fast and used to great effect.
- A huge amount of problems can be posed as a SAT:
  - Model Checking and hardware/software verification
  - Design of experiments
SAT Problems

A Satisfiability (SAT) formula is a logical expression that’s the conjunction (AND) of a set of Clauses, where a clause is the disjunction (OR) of Literals.

A Solution to a SAT formula is an assignment of the variables to the values True and False so that all the clauses are satisfied.

Example:

\[
\text{SAT: } (x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor \overline{x_3}) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})
\]

Solution: \(x_1 = \text{True}, \ x_2 = \text{False}, \ x_3 = \text{False} \text{ and } x_4 = \text{True}.\)

- If each clause has \(k\) literals we call the problem \(k\)-SAT.
- In general, determining if a SAT formula has a solution is NP-hard
- In practice solvers are fast and used to great effect
- A huge amount of problems can be posed as a SAT:
  \(\rightarrow\) Model Checking and hardware/software verification
  \(\rightarrow\) Design of experiments
  \(\rightarrow\) Classical planning
A *Satisfiability (SAT)* formula is a logical expression that’s the conjunction (AND) of a set of *Clauses*, where a clause is the disjunction (OR) of *Literals*.

A *Solution* to a SAT formula is an assignment of the variables to the values *True* and *False* so that all the clauses are satisfied.

**Example:**

\[
\text{SAT: } (x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})
\]

**Solution:** \( x_1 = \text{True}, \ x_2 = \text{False}, \ x_3 = \text{False} \) and \( x_4 = \text{True} \).

- If each clause has \( k \) literals we call the problem *\( k \)-SAT*.
- In general, determining if a SAT formula has a solution is NP-hard.
- In practice solvers are fast and used to great effect.
- A huge amount of problems can be posed as a SAT:
  - Model Checking and hardware/software verification
  - Design of experiments
  - Classical planning
  - …
RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to $2^n$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals and switch the variable's value.
3. If a valid solution is found return it. Otherwise return unsatisfiable.

**Example 1:**
Solution Found

\[(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor x_1)\]

<table>
<thead>
<tr>
<th>F</th>
<th>T</th>
<th>T</th>
<th>F</th>
<th>F</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

$S = (T, T, F, T)$. 

$t_1, t_2, t_3, t_4$. 

Lecture 4: Mixing and shuffling
RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:

Example 1: Solution Found

$(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor x_1)$

F T T F F T F

$S = (T, T, F, T)$. 

---

Lecture 4: Mixing and shuffling
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of it’s literals UAR and switch the variables value.

Example 1:
Solution Found

$(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor x_1)$

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

$S = (T, T, F, T)$. 

Lecture 4: Mixing and shuffling
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of it’s literals UAR and switch the variables value.
(3) If a valid solution is found return it. O/W return unsatisfiable
**2-SAT**

**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   - Choose an arbitrary clause that is not satisfied
   - Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$. 

---

Example 1:

Solution Found

$(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor x_1)$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

$S = (T, T, F, T)$.  
$x_1$ $x_2$ $x_3$ $x_4$  
0   F   F   F  

---

Lecture 4: Mixing and shuffling
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.

(2) Repeat up to \(2n^2\) times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals \(UAR\) and switch the variables value.

(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let \(A_i\) be the variable assignment at step \(i\).
- Let \(S\) be any solution and \(X_i = |\text{variable values shared by } A_i \text{ and } S|\).
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals UAR and switch the variables value.
(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1:

\[(x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor \overline{x}_1)\]

F  T  T  T  F  F  F  T  T  T

$S = (T, T, F, T)$.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.

(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of it’s literals UAR and switch the variables value.

(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1 :

$(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})$

$S = (T, T, F, T)$.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals UAR and switch the variables value.
(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1:

$$\left(x_1 \lor \overline{x_2}\right) \land \left(\overline{x_1} \lor x_3\right) \land \left(x_1 \lor x_2\right) \land \left(x_4 \lor \overline{x_3}\right) \land \left(x_4 \lor \overline{x_1}\right)$$

$$\begin{array}{cccc}
F & T & T & T \\
F & F & F & F \\
F & T & F & T \\
\end{array}$$

$$S = (T, T, F, T).$$

$0$ $F$ $F$ $F$ $F$ $F$
RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to \(2n^2\) times, terminating if all clauses are satisfied:
   - (a) Choose an arbitrary clause that is not satisfied
   - (b) Choose one of its literals \(\overline{U}A\) and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let \(A_i\) be the variable assignment at step \(i\).
- Let \(S\) be any solution and \(X_i = |\text{variable values shared by } A_i \text{ and } S|\).

Example 1:

\[
(x_1 \lor \overline{x}_2) \land (x_1 \lor \overline{x}_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor x_1) \\
F \quad F \quad T \quad T \quad F \quad T \quad T \quad F \quad T \quad T
\]

\[S = (T, T, F, T).\]

<table>
<thead>
<tr>
<th>(t)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals UAR and switch the variables value.
(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|.$

Example 1:

\[
(x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor \overline{x}_1)
\]

\[
F \quad F \quad T \quad T \quad F \quad T \quad F \quad T \quad F \quad T
\]

\[S = (T, T, F, T)\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1:

$$(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})$$

$F \ F \ T \ T \ F \ T \ F \ T \ F \ T$

$S = (T, T, F, T)$.

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.

2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals and switch the variables value.

3. If a valid solution is found return it. O/W return unsatisfiable.

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1:

$$(\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor \overline{x}_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x}_3) \land (x_4 \lor \overline{x}_1)$$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

$S = (T, T, F, T)$. 

Lecture 4: Mixing and shuffling
RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1:

\[(x_1 \lor \overline{x_2}) \land (x_1 \lor \overline{x_3}) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

$S = (T, T, F, T)$. 

Lecture 4: Mixing and shuffling
2-SAT

**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   - (a) Choose an arbitrary clause that is not satisfied
   - (b) Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |$variable values shared by $A_i$ and $S|$.

**Example 1:**

$$((x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1}))$$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

$S = (T, T, F, T)$. 

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to \(2n^2\) times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals \(\overline{A}\) or \(\overline{\overline{A}}\) and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let \(A_i\) be the variable assignment at step \(i\).
- Let \(S\) be any solution and \(X_i = |\text{variable values shared by } A_i \text{ and } S|\).

Example 1:

\[
(x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x}_1)
\]

\[
\begin{array}{cccc}
T & F & F & T \\
T & T & T & T \\
T & T & T & T \\
\end{array}
\]

\[
S = (T, T, F, T).
\]

<table>
<thead>
<tr>
<th>(t)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1: Solution Found

\[(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor \overline{x_3}) \land (x_4 \lor \overline{x_1})\]

T F F T T T T T T

$S = (T, T, F, T)$. 

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to \(2n^2\) times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of it's literals UAR and switch the variables value.
(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let \(A_i\) be the variable assignment at step \(i\).
- Let \(S\) be any solution and \(X_i = |\text{variable values shared by } A_i \text{ and } S|\).

Example 2:

\[(x_1 \lor \overline{x_2}) \land (x_1 \lor \overline{x_3}) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x_1})\]

\[
\begin{array}{cccc}
F & T & T & T \\
F & F & F & F \\
F & F & F & T \\
\end{array}
\]

\[S = (T, F, F, T).\]
2-SAT

**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals $UAR$ and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |variable values shared by $A_i$ and $S$|.

Example 2:

$$(x_1 \lor \overline{x_2}) \land (x_1 \lor x_3) \land (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_4 \lor x_1)$$

$S = (T, F, F, T)$.
2-SAT

RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2:

\[
(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x_1})
\]

$F$ $T$ $T$ $T$ $T$ $F$ $F$ $F$ $F$ $F$ $T$

$S = (T, F, F, T)$.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**2-SAT**

### RAND 2-SAT Algorithm

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |variable\ values\ shared\ by\ A_i\ and\ S|$.

**Example 2:**

\[
(x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x}_1)
\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

$S = (T, F, F, T)$. 

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   - (a) Choose an arbitrary clause that is not satisfied
   - (b) Choose one of its literals $UAR$ and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable.

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |$variable values shared by $A_i$ and $S|$

**Example 2:**

\[
(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x_1})
\]

\[
\begin{array}{cccc}
F & T & T & T \\
T & F & F & T \\
F & F & T & F \\
T & F & T & T
\end{array}
\]

\[S = (T, F, F, T).\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
2-SAT

**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   a. Choose an arbitrary clause that is not satisfied
   b. Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

**Example 2**:

$$(x_1 \lor \overline{x}_2) \land (x_1 \lor \overline{x}_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x}_1)$$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

$S = (T, F, F, T)$. 

0 1 2 3 4
**2-SAT**

**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   - (a) Choose an arbitrary clause that is not satisfied
   - (b) Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

**Example 2:**

$$(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x_1})$$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

$S = (T, F, F, T)$. 

Lecture 4: Mixing and shuffling
**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a *Step*. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |$variable values shared by $A_i$ and $S|$.

**Example 2:**

\[
(x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x}_1)
\]

\[
\begin{array}{cccccc}
F & F & T & T & F & T \\
F & F & T & T & F & T \\
F & T & F & T & F & T \\
F & T & F & T & T & T \\
\end{array}
\]

\[
S = (T, F, F, T).
\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
**2-SAT**

---

**RAND 2-SAT Algorithm**

1. Start with an arbitrary truth assignment.
2. Repeat up to \(2n^2\) times, terminating if all clauses are satisfied:
   - (a) Choose an arbitrary clause that is not satisfied
   - (b) Choose one of its literals UAR and switch the variables value.
3. If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let \(A_i\) be the variable assignment at step \(i\).
- Let \(S\) be any solution and \(X_i = |\text{variable values shared by } A_i \text{ and } S|\).

**Example 2:**

\[(x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x}_1)\]

\[F\ F\ T\ T\ T\ F\ T\ T\ T\ T\]

\[S = (T, F, F, T).\]

<table>
<thead>
<tr>
<th>(t)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to \(2n^2\) times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of it's literals UAR and switch the variables value.
(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let \(A_i\) be the variable assignment at step \(i\).
- Let \(S\) be any solution and \(X_i = |\text{variable values shared by } A_i \text{ and } S|\).

Example 2:

\[
(x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x}_1)
\]

\[
F \quad F \quad T \quad T \quad F \quad T \quad T \quad T \quad T
\]

\[
S = (T, F, F, T).
\]

<table>
<thead>
<tr>
<th>(t)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.

(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals UAR and switch the variables value.

(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2:

$$(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x_1})$$

$S = (T, F, F, T)$.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
2-SAT

RAND 2-SAT Algorithm

(1) Start with an arbitrary truth assignment.
(2) Repeat up to $2n^2$ times, terminating if all clauses are satisfied:
   (a) Choose an arbitrary clause that is not satisfied
   (b) Choose one of its literals UAR and switch the variables value.
(3) If a valid solution is found return it. O/W return unsatisfiable

- Call each loop of (2) a Step. Let $A_i$ be the variable assignment at step $i$.
- Let $S$ be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2: Solution Found

$$(x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_3) \land (x_4 \lor \overline{x_1})$$

$S = (T, F, F, T)$.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

T F F T T T T F T F

Lecture 4: Mixing and shuffling
If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.
If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$, 
2-SAT and the SRW on the path

Expected iterations of (2) in RAND 2-SAT

If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$,

(i) $P[ X_{i+1} = 1 \mid X_i = 0 ] = 1$
2-SAT and the SRW on the path

Expected iterations of (2) in RAND 2-SAT

If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$,

(i) $P[X_{i+1} = 1 \mid X_i = 0] = 1$

(ii) $P[X_{i+1} = k + 1 \mid X_i = k] \geq 1/2$
2-SAT and the SRW on the path

Expected iterations of (2) in RAND 2-SAT

If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$,

(i) $\Pr[X_{i+1} = 1 \mid X_i = 0] = 1$

(ii) $\Pr[X_{i+1} = k + 1 \mid X_i = k] \geq 1/2$

(iii) $\Pr[X_{i+1} = k - 1 \mid X_i = k] \leq 1/2$. 

Notice that if $X_i = n$ then $A_i = S$ thus solution found (may find another first).

Assume (pessimistically) that $X_0 = 0$ (we get none of our initial guesses right).

The stochastic process $X_i$ is complicated to describe in full however by (i)−(iii) we can couple it with $Y_i$ - the SRW on the $n$-path from 0.

This gives $\mathbb{E}[\text{time to find } S] \leq \mathbb{E}_0[\inf \{t: X_t = n\}] \leq \mathbb{E}_0[\inf \{t: Y_t = n\}] = h_0(n) = n^2$.

Provided a solution exists the RAND 2-SAT Algorithm will return a valid solution in time $2n^2$ with probability at least $1/2$. 

Lecture 4: Mixing and shuffling
2-SAT and the SRW on the path

Expected iterations of (2) in RAND 2-SAT

If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$,

(i) $P[X_{i+1} = 1 | X_i = 0] = 1$

(ii) $P[X_{i+1} = k + 1 | X_i = k] \geq 1/2$

(iii) $P[X_{i+1} = k - 1 | X_i = k] \leq 1/2$.

Notice that if $X_i = n$ then $A_i = S$ thus solution found (may find another first).
If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$,

(i) $P[X_{i+1} = 1 \mid X_i = 0] = 1$
(ii) $P[X_{i+1} = k + 1 \mid X_i = k] \geq 1/2$
(iii) $P[X_{i+1} = k - 1 \mid X_i = k] \leq 1/2$.

Notice that if $X_i = n$ then $A_i = S$ thus solution found (may find another first). Assume (pessimistically) that $X_0 = 0$ (we get none of our initial guesses right).
2-SAT and the SRW on the path

Expected iterations of (2) in RAND 2-SAT

If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$,

(i) $P[X_{i+1} = 1 \mid X_i = 0] = 1$
(ii) $P[X_{i+1} = k + 1 \mid X_i = k] \geq 1/2$
(iii) $P[X_{i+1} = k - 1 \mid X_i = k] \leq 1/2$.

Notice that if $X_i = n$ then $A_i = S$ thus solution found (may find another first).

Assume (pessimistically) that $X_0 = 0$ (we get non of our initial guesses right).

The stochastic process $X_i$ is complicated to describe in full however by (i) – (iii) we can couple it with $Y_i$- the SRW on the $n$-path from 0.
2-SAT and the SRW on the path

Expected iterations of (2) in RAND 2-SAT

If a valid solution $S$ exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most $n^2$.

Proof: Fix any solution $S$, then for any $i \geq 0$ and $1 \leq k \leq n - 1$,

(i) $P[X_{i+1} = 1 \mid X_i = 0] = 1$

(ii) $P[X_{i+1} = k + 1 \mid X_i = k] \geq 1/2$

(iii) $P[X_{i+1} = k - 1 \mid X_i = k] \leq 1/2$.

Notice that if $X_i = n$ then $A_i = S$ thus solution found (may find another first).

Assume (pessimistically) that $X_0 = 0$ (we get non of our initial guesses right).

The stochastic process $X_i$ is complicated to describe in full however by (i) – (iii) we can couple it with $Y_t$- the SRW on the $n$-path from 0. This gives

$E[\text{time to find } S] \leq E_0[\inf\{t : X_t = n\}] \leq E_0[\inf\{t : Y_t = n\}] = h_{0,n} = n^2$. □
2-SAT and the SRW on the path

Expected iterations of (2) in RAND 2-SAT

If a valid solution \( S \) exists then the expected number of iterations of loop (2) before RAND 2-SAT outputs a valid solution is at most \( n^2 \).

Proof: Fix any solution \( S \), then for any \( i \geq 0 \) and \( 1 \leq k \leq n - 1 \),

(i) \( \mathbb{P}[ X_{i+1} = 1 \mid X_i = 0 ] = 1 \)

(ii) \( \mathbb{P}[ X_{i+1} = k + 1 \mid X_i = k ] \geq 1/2 \)

(iii) \( \mathbb{P}[ X_{i+1} = k - 1 \mid X_i = k ] \leq 1/2 \).

Notice that if \( X_i = n \) then \( A_i = S \) thus solution found (may find another first).

Assume (pessimistically) that \( X_0 = 0 \) (we get non of our initial guesses right).

The stochastic process \( X_i \) is complicated to describe in full however by (i) – (iii) we can couple it with \( Y_i \)- the SRW on the \( n \)-path from 0. This gives

\[
\mathbb{E}[ \text{time to find } S ] \leq \mathbb{E}_0[ \inf \{ t : X_t = n \} ] \leq \mathbb{E}_0[ \inf \{ t : Y_t = n \} ] = h_{0,n} = n^2. \quad \square
\]

Proposition

Provided a solution exists the RAND 2-SAT Algorithm will return a valid solution in time \( 2n^2 \) with probability at least \( 1/2 \).