Lecture 4: Card Shuffling and Covertime

John Sylvester   Nicolás Rivera   Luca Zanetti   Thomas Sauerwald

Lent 2019

UNIVERSITY OF CAMBRIDGE

Outline

Shuffling and Strong Stationary Times

Covertime

$s - t$ Connectivity

2-Sat

Card Shuffling

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

Card Shuffling

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

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

Card Shuffling

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

Let Σ_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* σ of $[n] = \{1, \ldots, n\}$ is a bijection $\sigma : [n] \to [n]$.

Let Σ_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 Σ_n with π uniform.

Card Shuffling

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

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

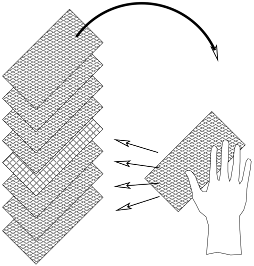
Sampling from uniform.

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 Σ_n with π uniform.



Strong Stationary Time

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

$$\mathbf{P}_x[t = \tau, X_\tau = y] = \mathbf{P}_x[t = \tau]\, \pi_y.$$

Strong Stationary Time

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

$$\mathbf{P}_x[t = \tau, X_\tau = y] = \mathbf{P}_x[t = \tau]\, \pi_y.$$

- Thus X_τ has distribution π and is independent of τ.

Strong Stationary Time

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

$$\mathbf{P}_x[t = \tau, X_\tau = y] = \mathbf{P}_x[t = \tau]\, \pi_y.$$

- Thus X_τ has distribution π and is independent of τ.

Mixing from Strong Stationary Times

If τ is a strong stationary time then for any $x \in \mathcal{I}$,

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

Strong Stationary Time

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

$$\mathbf{P}_x[t = \tau, X_\tau = y] = \mathbf{P}_x[t = \tau]\,\pi_y.$$

- Thus X_τ has distribution π and is independent of τ.

Mixing from Strong Stationary Times

If τ is a strong stationary time then for any $x \in \mathcal{I}$,

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

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

$$\mathbf{P}_x[X_t \in A \mid \tau > t]\,\mathbf{P}_x[\tau > t] + \mathbf{P}_x[X_t \in A \mid \tau \leq t]\,(1 - \mathbf{P}_x[\tau > t]) - \pi(A)$$

Strong Stationary Time

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

$$\mathbf{P}_x[t = \tau, X_\tau = y] = \mathbf{P}_x[t = \tau]\,\pi_y.$$

- Thus X_τ has distribution π and is independent of τ.

Mixing from Strong Stationary Times

If τ is a strong stationary time then for any $x \in \mathcal{I}$,

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

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

$$\begin{aligned}
&\mathbf{P}_x[X_t \in A \mid \tau > t]\,\mathbf{P}_x[\tau > t] + \mathbf{P}_x[X_t \in A \mid \tau \leq t]\,(1 - \mathbf{P}_x[\tau > t]) - \pi(A) \\
&\quad = (\mathbf{P}_x[X_t \in A \mid \tau > t] - \pi(A))\,\mathbf{P}_x[\tau > t]\,.
\end{aligned}$$

Strong Stationary Time

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

$$\mathbf{P}_x[t=\tau, X_\tau = y] = \mathbf{P}_x[t=\tau]\,\pi_y .$$

- Thus X_τ has distribution π and is independent of τ.

Mixing from Strong Stationary Times

If τ is a strong stationary time then for any $x \in \mathcal{I}$,

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

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

$$\begin{aligned} &\mathbf{P}_x[X_t \in A \mid \tau > t]\,\mathbf{P}_x[\tau > t] + \mathbf{P}_x[X_t \in A \mid \tau \leq t]\,(1 - \mathbf{P}_x[\tau > t]) - \pi(A) \\ &\qquad = (\mathbf{P}_x[X_t \in A \mid \tau > t] - \pi(A))\,\mathbf{P}_x[\tau > t] . \end{aligned}$$

Then since $-1 \leq \mathbf{P}_x[X_t \in A \mid \tau > t] - \pi(A) \leq 1$ we have

$$|\mathbf{P}_x[X_t \in A] - \pi(A)| = |\mathbf{P}_x[X_t \in A \mid \tau > t] - \pi(A)|\mathbf{P}_x[\tau > t] \leq \mathbf{P}_x[\tau > t] ,$$

for any $A \subset \mathcal{I}$.

Strong Stationary Time

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

$$\mathbf{P}_x[t = \tau, X_\tau = y] = \mathbf{P}_x[t = \tau]\,\pi_y.$$

- Thus X_τ has distribution π and is independent of τ.

Mixing from Strong Stationary Times

If τ is a strong stationary time then for any $x \in \mathcal{I}$,

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

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

$$\begin{aligned}\mathbf{P}_x[X_t \in A \mid \tau > t]\,\mathbf{P}_x[\tau > t] + \mathbf{P}_x[X_t \in A \mid \tau \leq t]\,(1 - \mathbf{P}_x[\tau > t]) - \pi(A)\\ = (\mathbf{P}_x[X_t \in A \mid \tau > t] - \pi(A))\,\mathbf{P}_x[\tau > t]\,.\end{aligned}$$

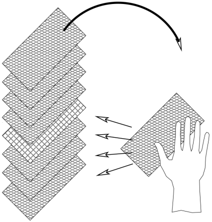
Then since $-1 \leq \mathbf{P}_x[X_t \in A \mid \tau > t] - \pi(A) \leq 1$ we have

$$|\mathbf{P}_x[X_t \in A] - \pi(A)| = |\mathbf{P}_x[X_t \in A \mid \tau > t] - \pi(A)|\mathbf{P}_x[\tau > t] \leq \mathbf{P}_x[\tau > t]\,,$$

for any $A \subset \mathcal{I}$. We can take $\sup_{A \subset \mathcal{I}}$ to complete the result. $\square$

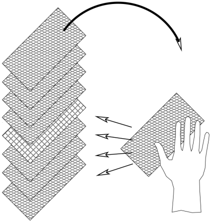
Strong Stationary time for Top-to-Random Shuffling

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

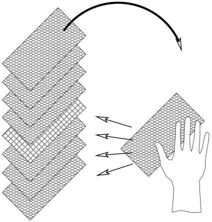


Strong Stationary time for T-to-R

τ_{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 τ_{top} be one step after the first time when B is on top of the deck.



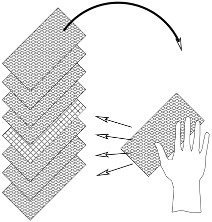
Strong Stationary time for T-to-R

τ_{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 τ_{top} be one step after the first time when B is on top of the deck.



Strong Stationary time for T-to-R

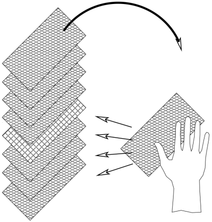
τ_{top} is a Strong Stationary time for the T-to-R chain.

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.

Strong Stationary time for Top-to-Random Shuffling

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



Strong Stationary time for T-to-R

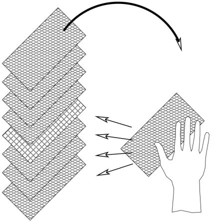
τ_{top} is a Strong Stationary time for the T-to-R chain.

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.

Strong Stationary time for Top-to-Random Shuffling

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



Strong Stationary time for T-to-R

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

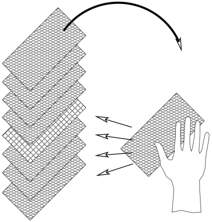
Proof: At any $t \geq 0$ all arrangements of the cards under B are equally likely.

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 τ_{top} be one step after the first time when B is on top of the deck.



Strong Stationary time for T-to-R

τ_{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.

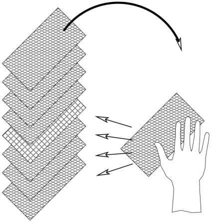
Induction: When $t = 0$, there are no cards under B. Suppose that the claim holds at time $t \geq 0$ with $k \geq 0$ cards under B. Two cases for time $t + 1$: either the top card is placed under B, or it is placed above B.

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 τ_{top} be one step after the first time when B is on top of the deck.



Strong Stationary time for T-to-R

τ_{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.

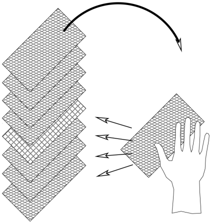
Case 2 New card goes above B, so cards under B remain in random order. □

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 $\mathbf{P}\left[X_{\tau_{top}} \mid \tau_{top} = t \right] = 1/n!$. □

Top-to-Random Shuffle

Mixing of Top-to-Random Shuffle

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

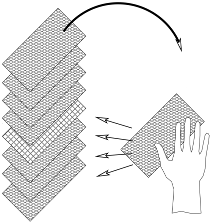


Top-to-Random Shuffle

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 $\text{Geo}(k/n)$.

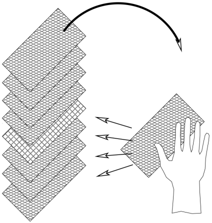


Top-to-Random Shuffle

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 τ_{top} is distributed the same as the number of balls thrown until no bin is empty in "*Balls and Bins*".



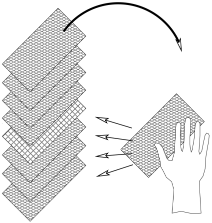
Top-to-Random Shuffle

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 $\mathrm{Geo}(k/n)$. This means that τ_{top} is distributed the same as the number of balls thrown until no bin is empty in "*Balls and Bins*". Thus

$$\mathbf{P}[\,\tau > n \ln n + Cn\,] \leq \mathbf{P}[\,\exists \text{ empty bin after } n \ln n + Cn \text{ balls}\,] \overset{\text{Lecture 1}}{\leq} e^{-C}.$$



Top-to-Random Shuffle

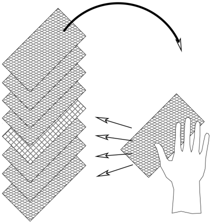
Mixing of Top-to-Random Shuffle

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 τ_{top} is distributed the same as the number of balls thrown until no bin is empty in "*Balls and Bins*". Thus

$$\mathbf{P}[\,\tau > n \ln n + Cn\,] \leq \mathbf{P}[\,\exists \text{ empty bin after } n \ln n + Cn \text{ balls}\,] \overset{\text{Lecture 1}}{\leq} e^{-C}.$$

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



Top-to-Random Shuffle

Mixing of Top-to-Random Shuffle

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

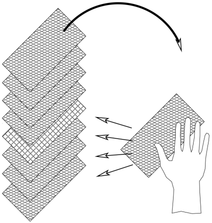
Proof: For $1 \leq k \leq n-1$ the time between the $(k-1)^{th}$ and k^{th} cards going under B is distributed $\text{Geo}(k/n)$. This means that τ_{top} is distributed the same as the number of balls thrown until no bin is empty in "*Balls and Bins*". Thus

$$\mathbf{P}[\,\tau > n \ln n + Cn\,] \leq \mathbf{P}[\,\exists \text{ empty bin after } n \ln n + Cn \text{ balls}\,] \overset{\text{Lecture 1}}{\leq} e^{-C}.$$

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

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

$$t_{mix} \approx \ln\left(|\Sigma_n|\right).$$



Realistic Shuffling - Riffle Shuffle

Realistic Shuffling - Riffle Shuffle

Riffle Shuffle

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

Realistic Shuffling - Riffle Shuffle

Riffle Shuffle

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

Riffle is fast

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

Realistic Shuffling - Riffle Shuffle

Riffle Shuffle

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

Riffle is fast

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

- Same state space Σ_n as T-to-R however this time

$$t_{mix} \approx \ln\ln\left(|\Sigma_n|\right).$$

Realistic Shuffling - Riffle Shuffle

Riffle Shuffle

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

Riffle is fast

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

- Same state space Σ_n as T-to-R however this time

$$t_{mix} \approx \ln\ln(|\Sigma_n|)\,.$$

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

Riffle is fast

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

- Same state space Σ_n as T-to-R however this time

$$t_{mix} \approx \ln\ln(|\Sigma_n|)\,.$$

- May have heard "7 riffle shuffles is enough" .
-

t	≤ 4	5	6	7	8	9
$\Delta(t)$	1.00	.92	.61	.33	.17.	.09

Outline

Shuffling and Strong Stationary Times

Covertime

$s - t$ Connectivity

2-Sat

Covertime

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

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

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

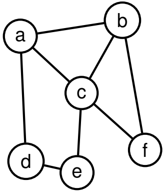
Covertime

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

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

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

Example:



$|V| = 6$

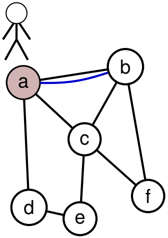
Covertime

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

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

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

Example:



$|V| = 6$

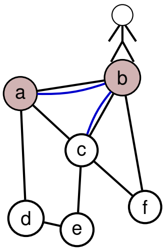
Covertime

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

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

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

Example:



$|V| = 6$

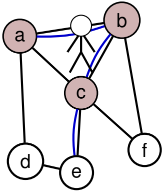
Covertime

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

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

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

Example:



$|V| = 6$

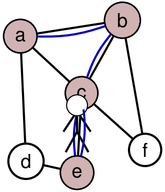
Covertime

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

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

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

Example:



$|V| = 6$

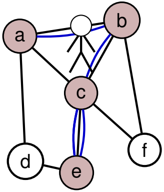
Covertime

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

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

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

Example:



$|V| = 6$

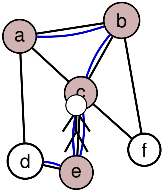
Covertime

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

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

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

Example:



$|V| = 6$

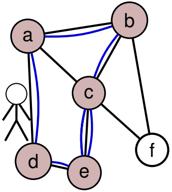
Covertime

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

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

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

Example:



$|V| = 6$

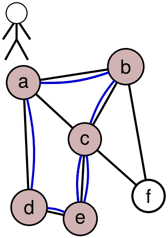
Covertime

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

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

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

Example:



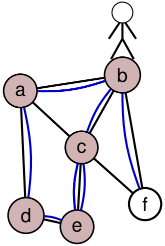
$|V| = 6$

Covertime

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

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

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



Example:

$|V| = 6$

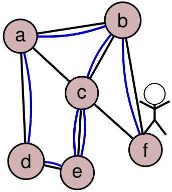
Covertime

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

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

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

Example:



$|V| = 6$

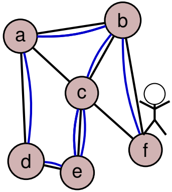
Covertime

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

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

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

Example:



$$|V| = 6$$
$$\tau_{cov}(G) = 9.$$

Stationary Distribution of a Random walk

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

Stationary Distribution of a Random walk

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 $x \in V$

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

□

Stationary Distribution of a Random walk

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 $x \in V$

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

Crossing time of an edge

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

Stationary Distribution of a Random walk

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 $x \in V$

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

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

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

Stationary Distribution of a Random walk

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 $x \in V$

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

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

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

By the Markov property we have

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

Stationary Distribution of a Random walk

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

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

By the Markov property we have

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

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

Stationary Distribution of a Random walk

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 $x \in V$

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

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

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

By the Markov property we have

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

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

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

Covertime bdd

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

Covertime bdd

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.

Covertime bdd

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.

Covertime bdd

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).

Covertime bdd

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

Proof: Any connected graph has a spanning tree T with $n-1$ edges.

Choose any root v_0 for T and fix a tour $v_0, \dots, v_{2n-2}$ 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

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

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

Covertime bdd

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

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

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

Matthews bound

For any graph G we have

$$t_{cov}(G) \leq \left(\sum_{m=1}^{n-1} \frac{1}{m} \right) \cdot \max_{x,y \in V} h_{x,y} \approx (\ln n) \cdot \max_{x,y \in V} h_{x,y}.$$

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 $0 \leq k \leq n$.

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 $0 \le k \le n$.

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

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

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 $0 \le k \le n$.

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

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

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

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

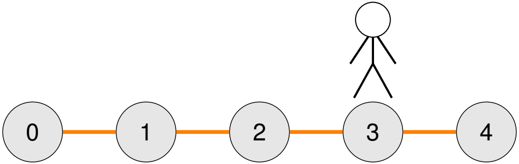
and for any $1 \le k \le n-1$ we have,

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

Covertime of the Path

For the path P_n on n vertices we have

$$n^2 \leq t_{cov}(P_n) \leq 2n^2.$$

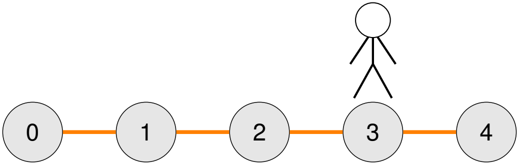


Covertime of the Path

For the path P_n on n vertices we have

$$n^2 \leq t_{cov}(P_n) \leq 2n^2.$$

Proof: For the lower bound, take the random walk from the left hand end point (vertex 0). To cover the path we must at reach the righthand end point (vertex n), this takes time n^2 in expectation.



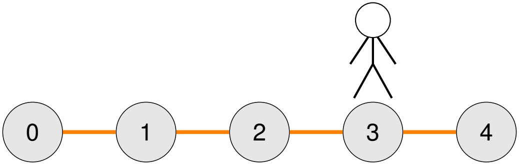
Covertime of the Path

For the path P_n on n vertices we have

$$n^2 \leq t_{cov}(P_n) \leq 2n^2.$$

Proof: For the lower bound, take the random walk from the left hand end point (vertex 0). To cover the path we must at reach the righthand end point (vertex n), this takes time n^2 in expectation.

For the upper bound the max time to reach one end point from any start point is at most n^2. Now from this end point if we reach the opposite end point we must have visited every vertex, this takes an additional n^2 expected time. □



Outline

Shuffling and Strong Stationary Times

Covertime

$s - t$ Connectivity

2-Sat

$s - t$ Connectivity

$s - t$ Connectivity Problem

$s - t$ Connectivity

$s - t$ 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

$s - t$ 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` 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.

$s - t$ 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. $\geq 1/2$. $\square$

$s - t$ 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. $\geq 1/2$. $\square$

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

$s - t$ 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.

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. $\geq 1/2$. $\square$

- 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

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*.

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.

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:

SAT: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$

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:

SAT: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$

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

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:

$$\text{SAT:}\quad (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- If each clause has k literals we call the problem *k-SAT*.

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:

$$\text{SAT:}\quad (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- 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:

$$\text{SAT:}\quad (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- 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

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:

$$\text{SAT:}\quad (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- 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:

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:

$$\text{SAT:}\quad (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- 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

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:

$$\text{SAT:}\quad (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- 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:

$$\text{SAT:}\quad (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- 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

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:

$$\text{SAT: } (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

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

- 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
  - → ...

2-SAT

RAND 2-SAT Algorithm

2-SAT

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:

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.

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:

   (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.

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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

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

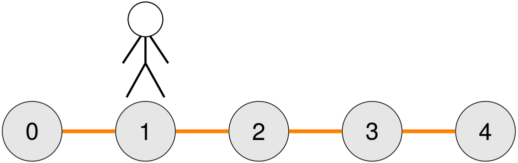
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F F T F T

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F

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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

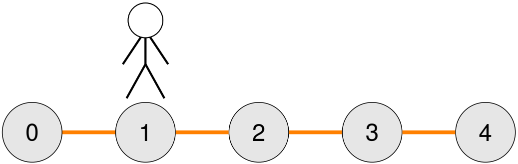
Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F F T F T

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F



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.

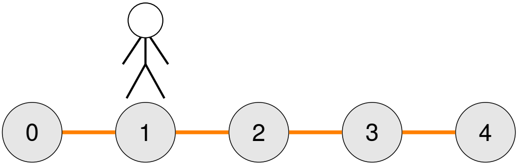
(3) If a valid solution is found `return` it. O/W `return` unsatisfiable

- 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 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F F T F T



$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F

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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

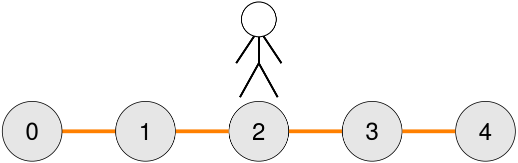
Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

F F T T F T F T F T

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	F	F



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

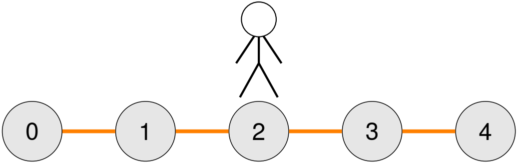
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

F F T T F T F T F T

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	F	F

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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

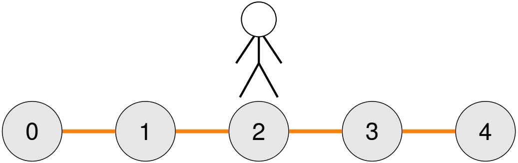
Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

F F T T F T F T F T

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	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 it's literals UAR and switch the variables value.

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

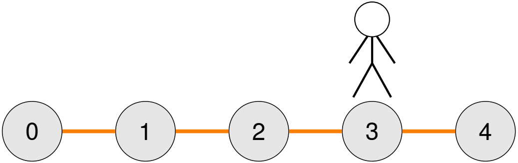
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

T F F T T T F T F F

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	F	F
2	T	T	F	F

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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

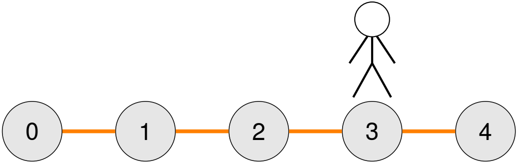
Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

T F F T T T F T F F

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	F	F
2	T	T	F	F



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

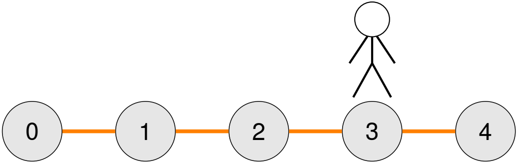
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

T F F T T T F T F F

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	F	F
2	T	T	F	F

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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

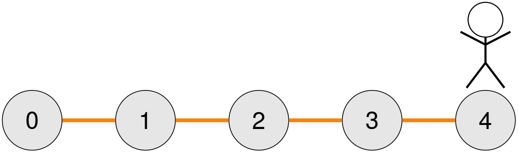
Example 1 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

T F F T T T T T T F

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	F	F
2	T	T	F	F
3	T	T	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 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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

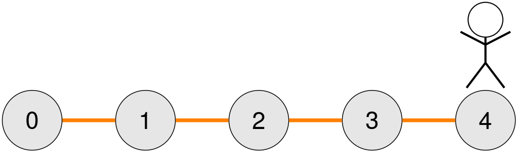
Example 1 : Solution Found

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee \overline{x_3}) \wedge (x_4 \vee \overline{x_1})$$

T F F T T T T T T F

$$S = (\mathtt{T}, \mathtt{T}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	T	F	F
2	T	T	F	F
3	T	T	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 it's literals UAR and switch the variables value.

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

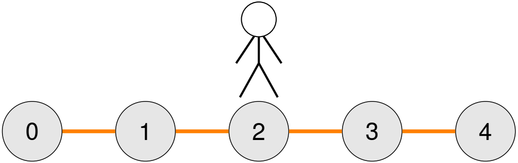
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F F F F T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F

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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

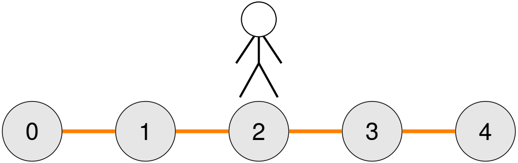
Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F F F F T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F



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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

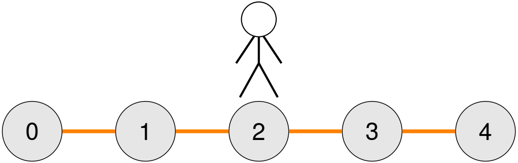
Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F F F F T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F



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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

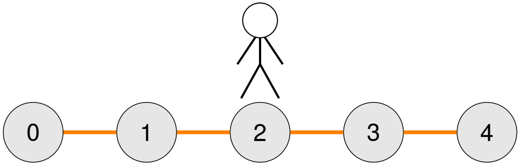
Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F F F F T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F



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

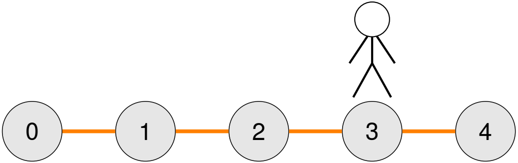
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F T F T T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	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 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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

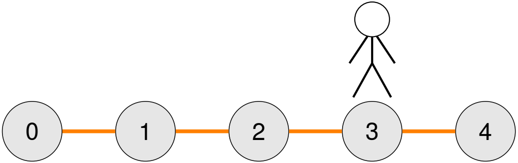
Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F T F T T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	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 it's literals UAR and switch the variables value.

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

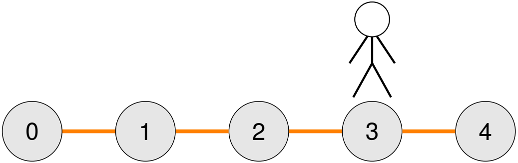
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F T T T F F T F T T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	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 it's literals UAR and switch the variables value.

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

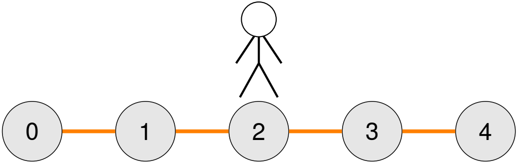
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F F T T F T T F T T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	F	F	T
2	F	T	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 it's literals UAR and switch the variables value.

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

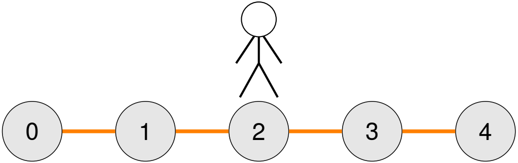
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F F T T F T T F T T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	F	F	T
2	F	T	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 it's literals UAR and switch the variables value.

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

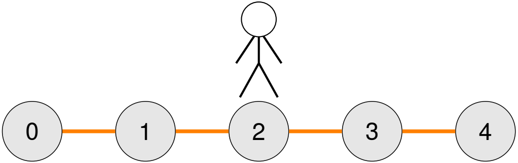
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

F F T T F T T F T T

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	F	F	T
2	F	T	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 it's literals UAR and switch the variables value.

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

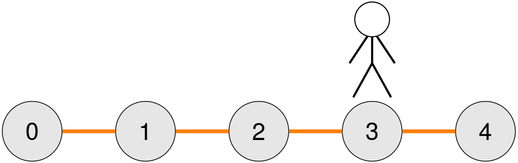
- Call each loop of (2) a *Step*. Let A_i be the variable assignment at step i.
- Let S be any solution and $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

Example 2 :

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

T F F T T T T F T F

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$



t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	F	F	T
2	F	T	F	T
3	T	T	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 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 $X_i = |\text{variable values shared by } A_i \text{ and } S|$.

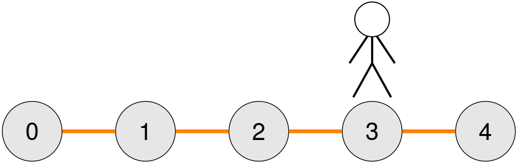
Example 2 : Solution Found

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee x_2) \wedge (x_4 \vee x_3) \wedge (x_4 \vee \overline{x_1})$$

T F F T T T T F T F

$$S = (\mathtt{T}, \mathtt{F}, \mathtt{F}, \mathtt{T}).$$

t	x_1	x_2	x_3	x_4
0	F	F	F	F
1	F	F	F	T
2	F	T	F	T
3	T	T	F	T



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.

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 $1 \leq k \leq n - 1$,

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

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

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

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

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

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

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

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

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

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

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

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

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

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).

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

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

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

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

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

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

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

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

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

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

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

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

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

Proposition

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