Exact Learning from Random Walk M.Sc. Seminar

Iddo Bentov

Advisor: Prof. Nader Bshouty

Department of Computer Science
Technion – Israel Institute of Technology

November 25, 2009

Overview

Computational Learning Theory

- Goal: learn from examples an unknown target function $f \in \mathcal{C}$
- $\mathcal{C}_n \subsetneq \{g \mid g : \{0,1\}^n \to \{0,1\}\}$, $\mathcal{C} = \cup_{n=1}^{\infty} \mathcal{C}_n$
- Learning any boolean function out of the 2^{2^n} possibilities is exponentially hard, since the size of a random function is $\Omega(2^n)$
- We assume that the class $\mathcal{C}$ is known to the learner.

Examples for the class $\mathcal{C}$

- monotone term: the class of conjunctions of unnegated variables, e.g. $f(x_1, x_2, x_3, x_4, x_5) = x_1 \wedge x_4 \wedge x_5 = x_1 x_4 x_5$
- read-once DNF: the class of DNF formulas in which every variable appears at most once, e.g. $x_1\overline{x_3} \vee x_4 \vee x_2\overline{x_8}x_9$
- poly-size DNF: the class of functions that can be represented as DNF with at most n^d terms, where d is a constant
- k-juntas: functions that depend on at most k (out of n) variables

The Online learning model

The Online learning model

- At each trial i the learner receives an example $x^{(i)} \in \{0,1\}^n$ from a teacher, and makes the prediction $h(x^{(i)})$ according to an hypothesis h that the learner updates.

The Online learning model

The Online learning model

- At each trial i the learner receives an example $x^{(i)} \in \{0,1\}^n$ from a teacher, and makes the prediction $h(x^{(i)})$ according to an hypothesis h that the learner updates.

$i = 1, 2, 3, \ldots$ ad infinitum

The Online learning model

The Online learning model

- At each trial i the learner receives an example $x^{(i)} \in \{0,1\}^n$ from a teacher, and makes the prediction $h(x^{(i)})$ according to an hypothesis h that the learner updates. $i = 1, 2, 3, \ldots$ ad infinitum
- If $h(x^{(i)}) \neq f(x^{(i)})$, the teacher sends a "mistake" message back to the learner after the i^{th} trial.

The Online learning model

The Online learning model

- At each trial i the learner receives an example $x^{(i)} \in \{0,1\}^n$ from a teacher, and makes the prediction $h(x^{(i)})$ according to an hypothesis h that the learner updates. $i = 1, 2, 3, \ldots$ ad infinitum
- If $h(x^{(i)}) \neq f(x^{(i)})$, the teacher sends a "mistake" message back to the learner after the i^{th} trial.
- Need to achieve **exact** learning: $h \equiv f$ with probability $1 - \delta$ after at most $poly(n, size_{\mathcal{C}}(f), \frac{1}{\delta})$ mistakes.

The Online learning model

The Online learning model

- At each trial i the learner receives an example $x^{(i)} \in \{0,1\}^n$ from a teacher, and makes the prediction $h(x^{(i)})$ according to an hypothesis h that the learner updates. $i = 1, 2, 3, \ldots$ ad infinitum
- If $h(x^{(i)}) \neq f(x^{(i)})$, the teacher sends a "mistake" message back to the learner after the i^{th} trial.
- Need to achieve **exact** learning: $h \equiv f$ with probability $1 - \delta$ after at most $poly(n, size_{\mathcal{C}}(f), \frac{1}{\delta})$ mistakes. $\log \frac{1}{\delta} \cdot poly(n, size_{\mathcal{C}}(f))$

The Online learning model

The Online learning model

- At each trial i the learner receives an example $x^{(i)} \in \{0,1\}^n$ from a teacher, and makes the prediction $h(x^{(i)})$ according to an hypothesis h that the learner updates. $i = 1, 2, 3, \ldots$ ad infinitum
- If $h(x^{(i)}) \neq f(x^{(i)})$, the teacher sends a "mistake" message back to the learner after the i^{th} trial.
- Need to achieve **exact** learning: $h \equiv f$ with probability $1 - \delta$ after at most $poly(n, size_{\mathcal{C}}(f), \frac{1}{\delta})$ mistakes. $\log \frac{1}{\delta} \cdot poly(n, size_{\mathcal{C}}(f))$
- $h(x^{(i)})$ has to be computed in $poly(n, size_{\mathcal{C}}(f))$ time.

The Online learning model

The Online learning model

- At each trial i the learner receives an example $x^{(i)} \in \{0,1\}^n$ from a teacher, and makes the prediction $h(x^{(i)})$ according to an hypothesis h that the learner updates. $i = 1, 2, 3, \ldots$ ad infinitum
- If $h(x^{(i)}) \neq f(x^{(i)})$, the teacher sends a "mistake" message back to the learner after the i^{th} trial.
- Need to achieve **exact** learning: $h \equiv f$ with probability $1 - \delta$ after at most $poly(n, size_{\mathcal{C}}(f), \frac{1}{\delta})$ mistakes. $\log \frac{1}{\delta} \cdot poly(n, size_{\mathcal{C}}(f))$
- $h(x^{(i)})$ has to be computed in $poly(n, size_{\mathcal{C}}(f))$ time.

Comparison with Probably Approximately Correct (PAC) learning

Online $\overset{\Rightarrow}{\not\Leftarrow}$ PAC, i.e. exact learning from a malicious teacher is strictly harder than approximate learning from a probabilistic teacher, assuming that (**P** $\neq$**NP** and) one-way functions exist.

Example of Online learning

Example: learning a non-monotone term, $|\mathcal{C}_n| = 3^n + 1 \ll 2^{2^n}$

1. start with $h \equiv 0$ and predict $h(x^{(i)}) = 0$ until the first mistake
2. set h to the full term that is consistent with the assignment that caused the prediction mistake, e.g. if $x^{(i)} = (1, 0, 1, 1, 0)$ and $f(x^{(i)}) = 1$ then set $h = x_1\overline{x_2}x_3x_4\overline{x_5}$
3. predict according to the hypothesis h, and upon mistakes remove the (irrelevant) variables that are inconsistent with h, for example if $f(0, 0, 1, 1, 1) = 1$ then remove the irrelevant variables x_1 and x_5, and continue to predict with $h = \overline{x_2}x_3x_4$

   note: $f(x) = 0 \Longrightarrow h(x) = 0$ because f is contained in h

$\Rightarrow$ Total number of prediction mistakes $\leq n + 1$

Online models that restrict the power of the teacher

UOnline - Uniform Online

The teacher has to select each example randomly uniformly.

RWOnline - Random Walk Online

Malicious teacher who is restricted to selecting successive examples that differ by at most one bit, e.g.:
$(0,0,0,0,0), (0,0,0,1,0), (1,0,0,1,0), (1,1,0,1,0), (1,1,0,0,0), \ldots$

URWOnline - Uniform Random Walk Online

The examples are selected via the random walk stochastic process, i.e. according to the following time-homogeneous Markov chain:

$$\Pr(x^{(t+1)} = y \mid x^{(t)} = x) = \begin{cases} \frac{1}{n+1} & \text{if } \mathrm{Ham}(y,x) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Motivation behind learning from a stochastic process

Practical motivation

- In models such as PAC and UROnline, it is assumed that the learner obtains independent examples from the environment. This assumption doesn't always hold in practice.
- In particular, successive examples that are generated by a physical process tend to differ only slightly, e.g. trajectory of robots. Therefore, learning models based on random walk or similar stochastic processes are more appropriate in such cases.

Theoretical motivation

- Learning in adversarial settings or PAC settings appears to be hard for many common classes. Even PAC under the uniform distribution is conjectured to be too hard for some classes.
- Passive learning in the URWOnline model is strictly harder than active learning via MQs, but still easier than UOnline.

The current state of knowledge

	Class	Model	Learnable?	Remarks
0	poly-size DNF	Proper Online (h is DNF)	No	combinatorial properties
1	poly-size DNF	Online, PAC	Open, $2^{\tilde{\mathcal{O}}(\sqrt[3]{n})}$	if **P** = **NP** then yes, even for $\mathcal{C}$ = circuits
2	poly-size DNF	UOnline, uniform-PAC	Open, $n^{\mathcal{O}(\log n)}$	no results for the UOnline model
3	$\mathcal{O}(\log n)$-juntas $\omega(1)$-juntas	UOnline, uniform-PAC	Open, $\approx n^{0.704 \cdot k}$	$(d \cdot \log n)$-junta is n^d-term DNF
4	ROM-DNF	Online, PAC	Open	composition lemma
5	ROM-DNF	UOnline	Open	no composition
6	RO-DNF	uniform-PAC	Yes	no composition
7	poly-size DNF	URWalk-PAC	Yes	MQ $\prec$ RW $\prec$ uPAC
8	$\mathcal{O}(\log n)$-juntas	URWOnline	Yes	straightforward
9	RO-DNF	URWOnline	Yes	
	poly-size DNF	URWOnline	No?	implies UOnline

Relation between the restricted Online models

$$\begin{array}{ccc} \text{Online} & \equiv & \text{RWOnline} \\ \Downarrow & & \Downarrow\not\Uparrow \\ \text{UOnline} & \begin{array}{c}\not\Leftarrow \\ \Rightarrow\end{array} & \text{URWOnline} \end{array}$$

- For $\Rightarrow$, we note that the RW stochastic process mixes rapidly
- For $\not\Leftarrow$, we prove URWOnline learnability of $\mathcal{O}(\log n)$-juntas
- For $\not\Uparrow$, we prove URWOnline learnability of RO(M)-DNF
- For $\equiv$, we prove that RWOnline implies Online
- We also prove that read-3 DNF learnability in URWOnline implies that any DNF can be learned in UOnline (under reasonable conditions)

Random Walk with an adversarial teacher

Definition: one variable override

We say that class $\mathcal{C}$ possesses the **one variable override** property if for every $f(x_1, \ldots, x_n) \in \mathcal{C}$ there exist constants $c_0, c_1 \in \{0, 1\}$ and $g(x_1, \ldots, x_{n+1}) \in \mathcal{C}$ such that

$$g \equiv \begin{cases} f & x_{n+1} = c_0 \\ c_1 & \text{otherwise} \end{cases}$$

Observation regarding non-probabilistic walk

If $\mathcal{C}$ has the one variable override property, a malicious teacher can

1. set the certain variable that overrides the function's value
2. choose arbitrary values for the other variables via a walk that flips one bit at a time
3. reset this certain variable and force the learner to make a prediction on an arbitrary assignment

The one variable override property

Classes that possess the one variable override property

Common classes indeed possess the one variable override property

- RO-DNF, decision list, decision tree, DFA:
  $f(x_1, \ldots, x_n) \in \mathcal{C} \Longrightarrow g(x_1, \ldots, x_{n+1}) = x_{n+1} \vee f(x_1, \ldots, x_n)$
- k-term DNF, k-term RSE:
  $f(x_1, \ldots, x_n) \in \mathcal{C} \Longrightarrow g(x_1, \ldots, x_{n+1}) = x_{n+1} \wedge f(x_1, \ldots, x_n)$
- halfspace:
  $f(x_1, \ldots, x_n) = [\sum_{i=1}^{n} a_i x_i \geq b] \in \mathcal{C} \Longrightarrow$
  $g(x_1, \ldots, x_{n+1}) = x_{n+1} \vee f(x_1, \ldots, x_n)$
  $= [(b + \sum_{i=1}^{n} |a_i|)x_{n+1} + \sum_{i=1}^{n} a_i x_i \geq b]$

Classes that don't possess the one variable override property

- boolean threshold functions (which are Online learnable):
  $f(x_1, \ldots, x_n) = [\sum_{i=1}^{n} a_i x_i \geq b]$ where $a_i \in \{0, 1\}$
- Flipping a single variable can affect the sum by only ± 1

Reducing RWOnline to Online

Formal proof of the observation

Suppose **A** is a RWOnline algorithm that learns the class $\mathcal{C}$ with a mistake bound $T(n, size_{\mathcal{C}}(f))$, and $\mathcal{C}$ possesses the one variable override property.

Algorithm **B** will learn $f(x_1, \ldots, x_n) \in \mathcal{C}$ in the Online model, by constructing

$$g \equiv \begin{cases} f & x_{n+1} = c_0 \\ c_1 & \text{otherwise} \end{cases}$$

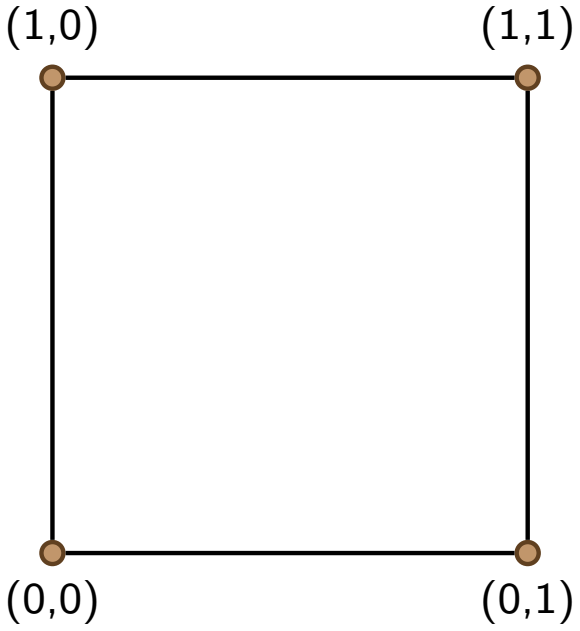
And simulating **A** as follows

1. receive $x^{(t)}$ from the teacher
2. $\tilde{x}^{(t-1)} \leftarrow (x_1^{(t-1)}, x_2^{(t-1)}, \ldots, x_n^{(t-1)}, \overline{c_0})$, $\tilde{x}^{(t)} \leftarrow (x_1^{(t)}, x_2^{(t)}, \ldots, x_n^{(t)}, \overline{c_0})$
3. Walk from $\tilde{x}^{(t-1)}$ to $\tilde{x}^{(t)}$, asking **A** for predictions, and informing $\mathbf{A}$ of mistakes in case it fails to predict c_1 after each bit flip
4. Send $(x^{(t)}, c_0)$ to $\mathbf{A}$, and let y be the answer of $\mathbf{A}$ on $(x^{(t)}, c_0)$
5. Send the answer y to the teacher, and inform $\mathbf{A}$ in case of a mistake

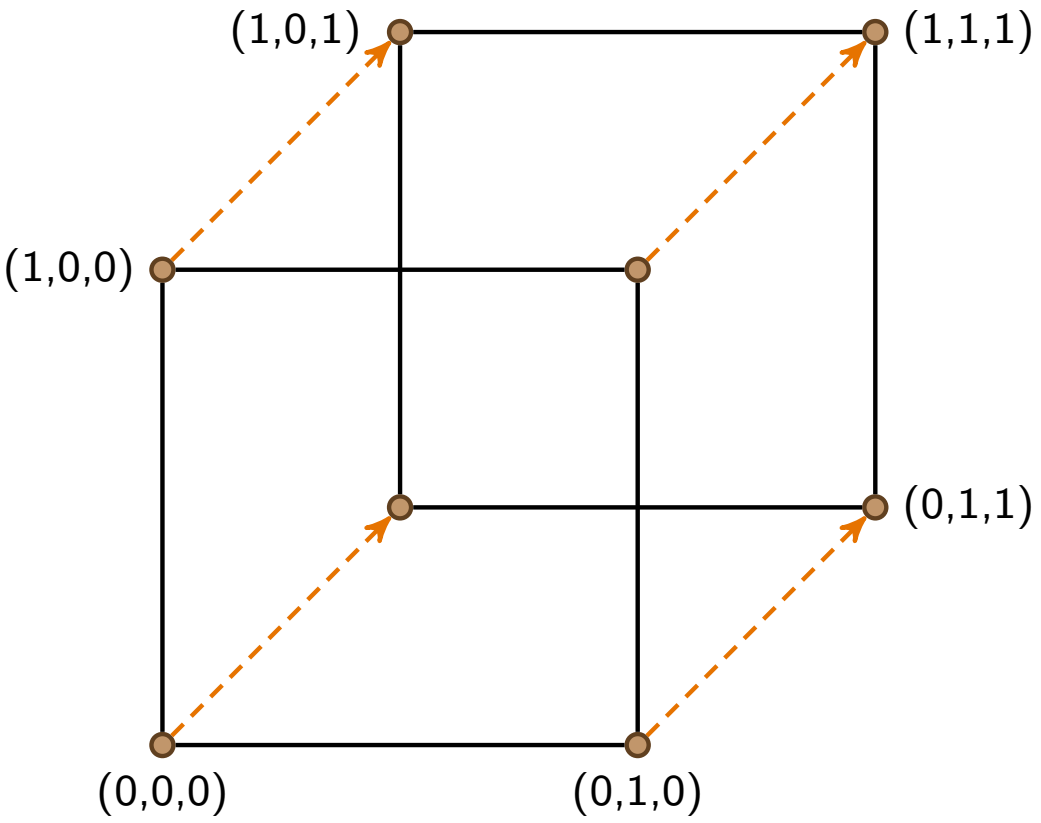
Thus, **A** receives examples that differ by one bit, and the correct "mistake" messages. Therefore, **A** learns g after at most $T(n+1, size_{\mathcal{C}}(g))$ mistakes.

$\Longrightarrow$ worst-case mistake bound for **B** is $2^2 \cdot T(n+1, size_{\mathcal{C}}(g))$

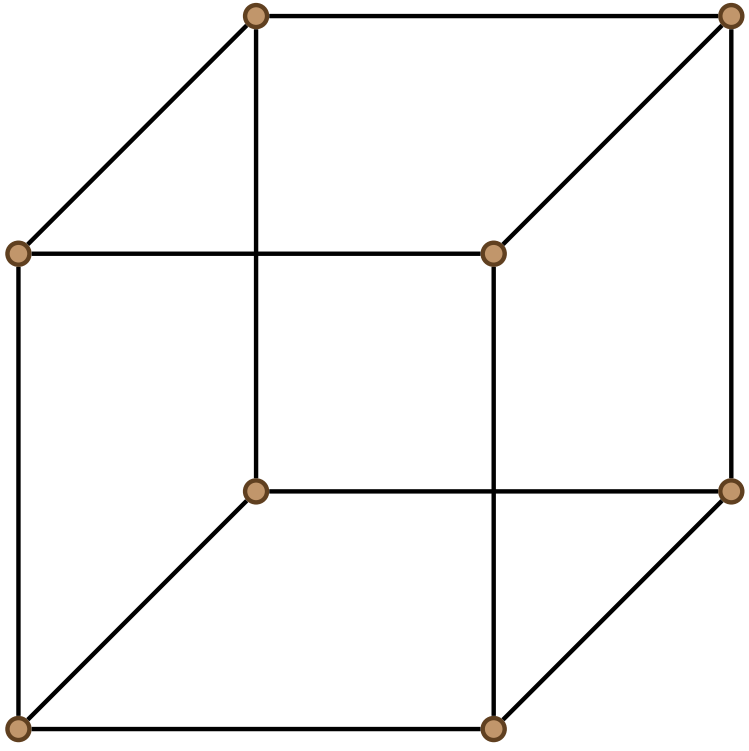
The hypercube (a.k.a. the Hamming cube) as a graph

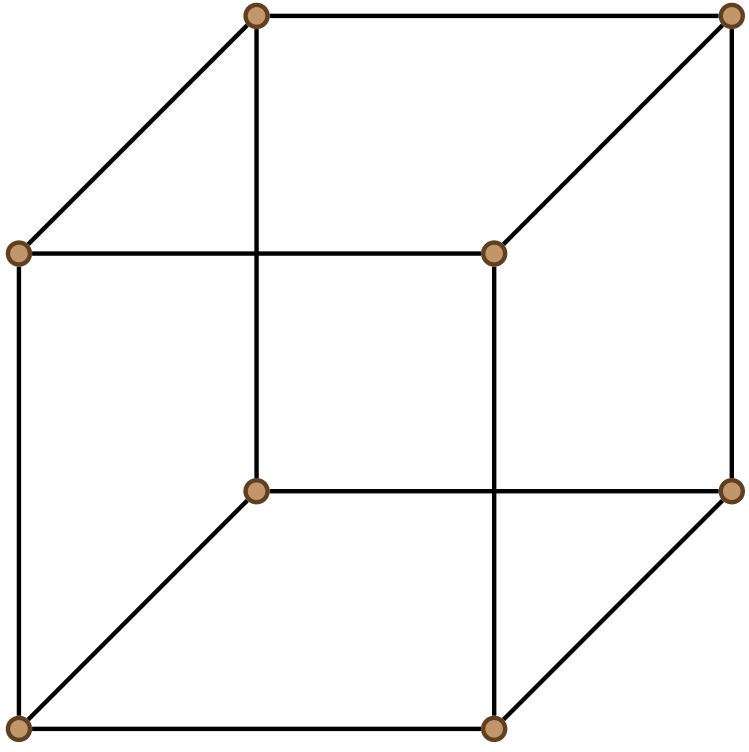


The hypercube (a.k.a. the Hamming cube) as a graph

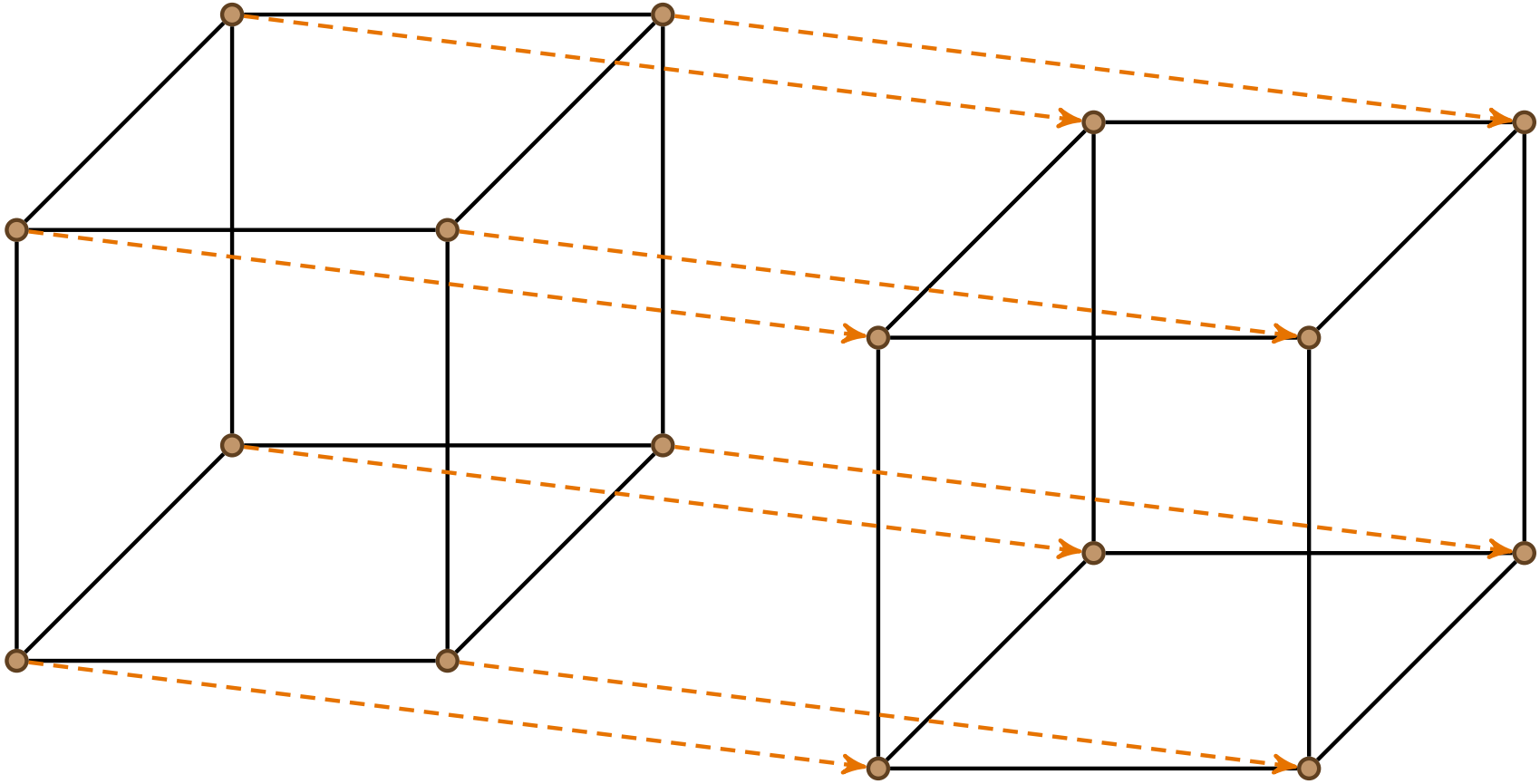


The hypercube (a.k.a. the Hamming cube) as a graph

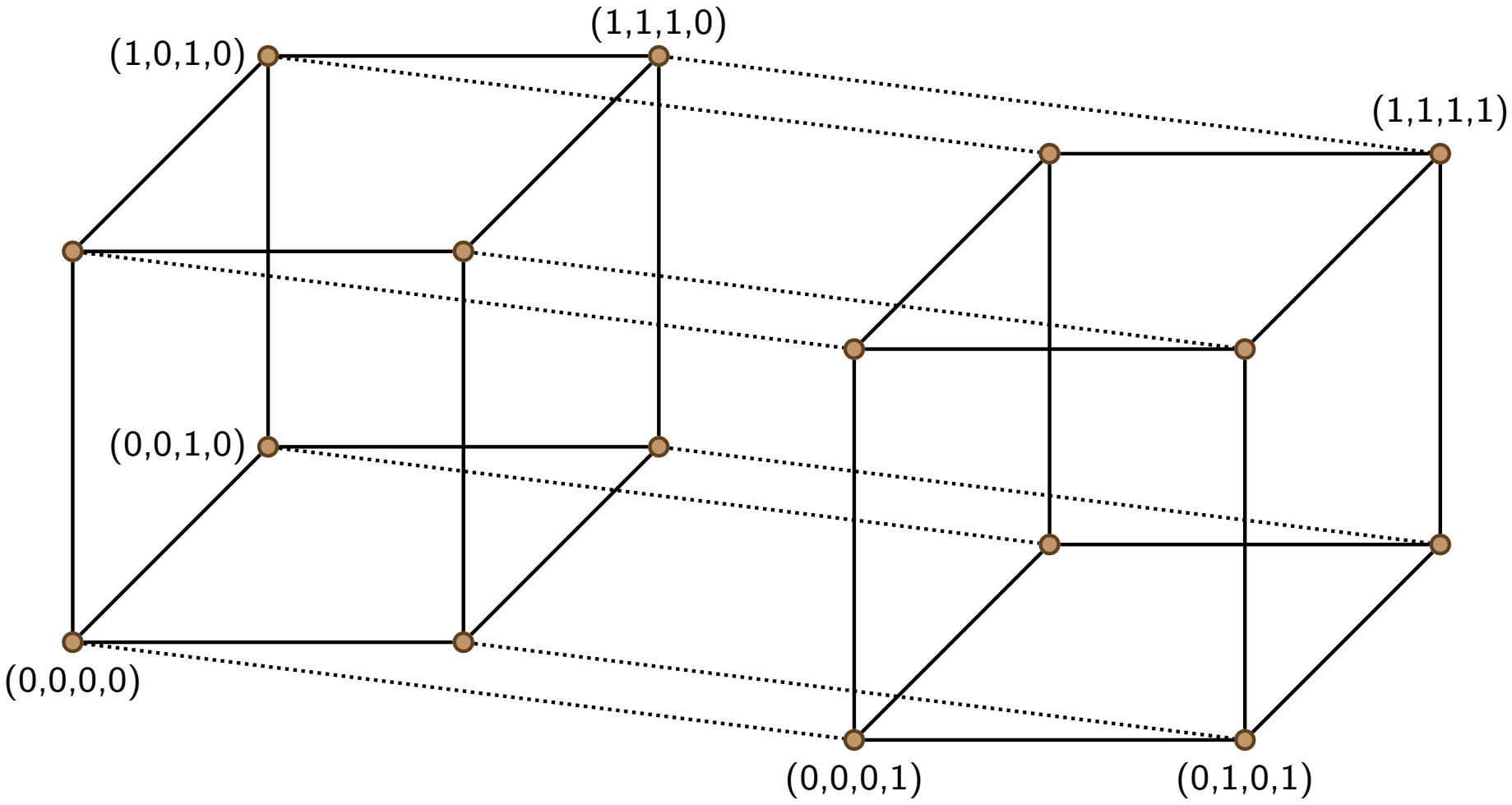




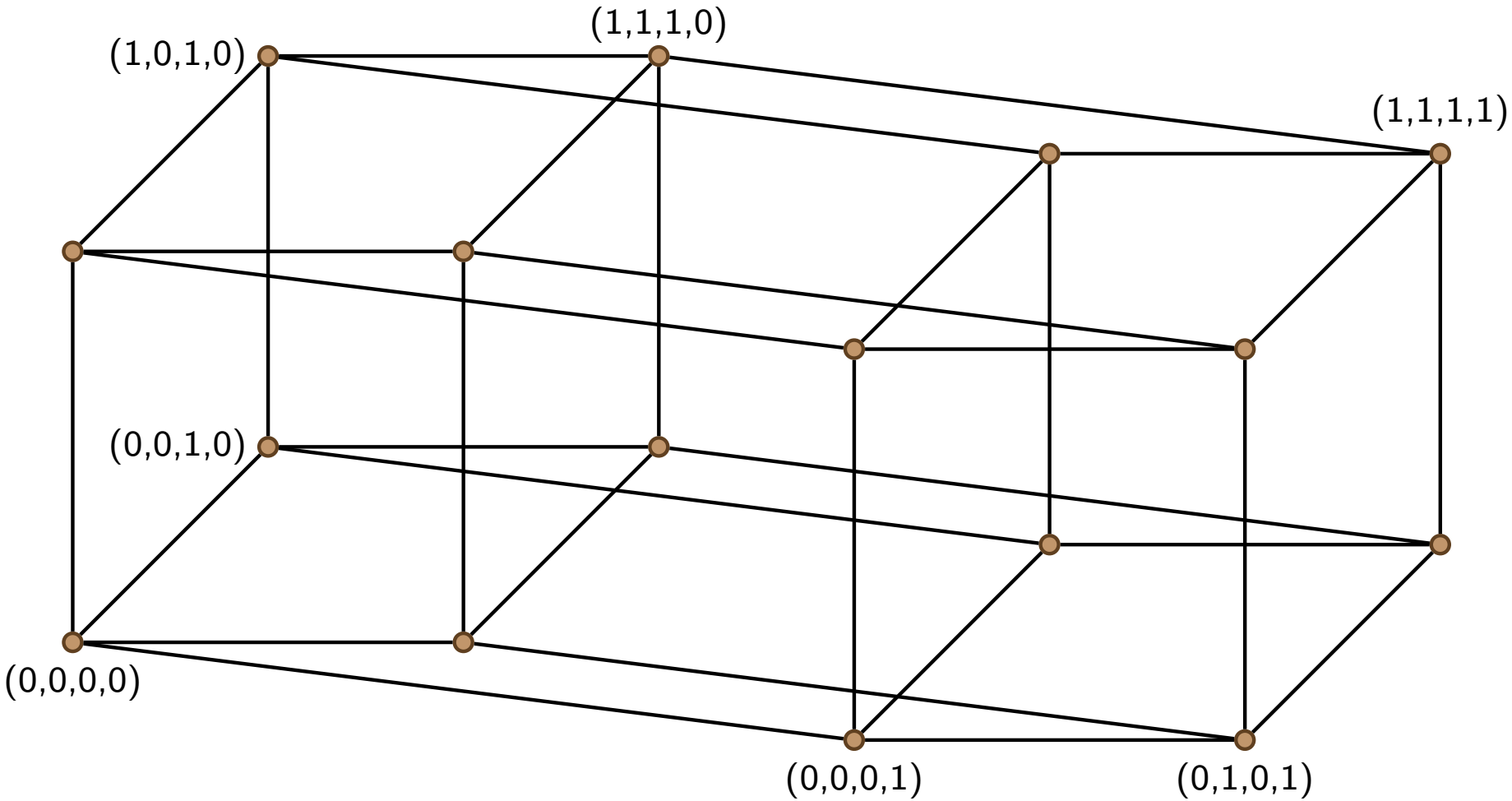
The hypercube (a.k.a. the Hamming cube) as a graph



The hypercube (a.k.a. the Hamming cube) as a graph



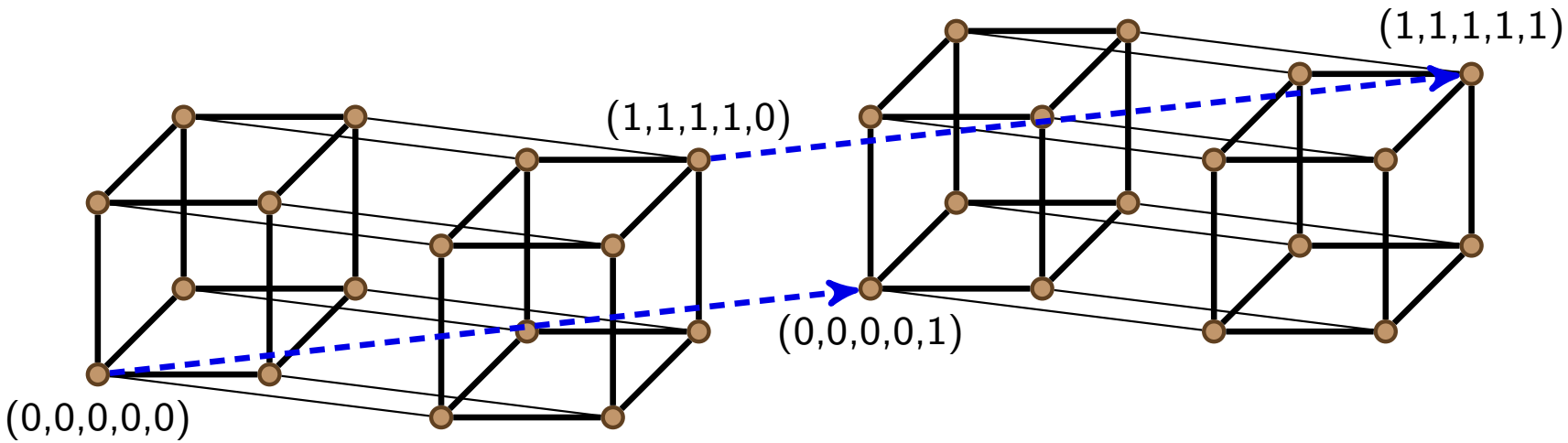
The hypercube (a.k.a. the Hamming cube) as a graph



- HYP_n is a n-regular graph with 2^n vertices (HYP_4 illustrated)

The mixing time and hitting time on HYP_n

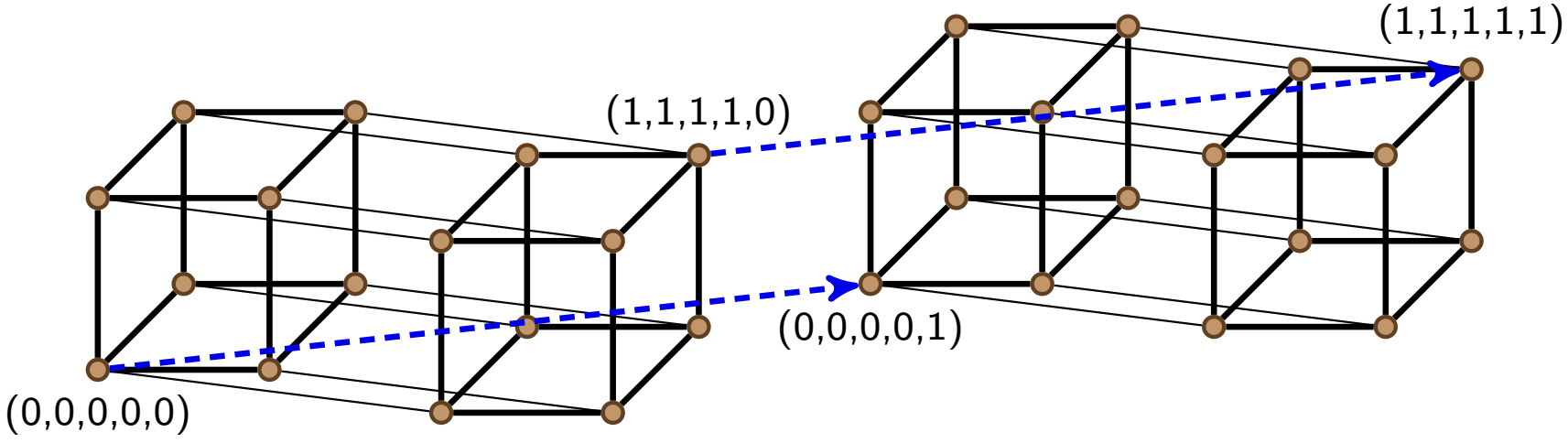
- Mixing time: after how many steps of random walk on HYP_n is the distribution of the vertex that we are at identical to the uniform distribution?



The mixing time and hitting time on HYP_n

- Mixing time: after how many steps of random walk on HYP_n is the distribution of the vertex that we are at identical to the uniform distribution?
- For **simple** random walk, never (parity of the bits is preserved)

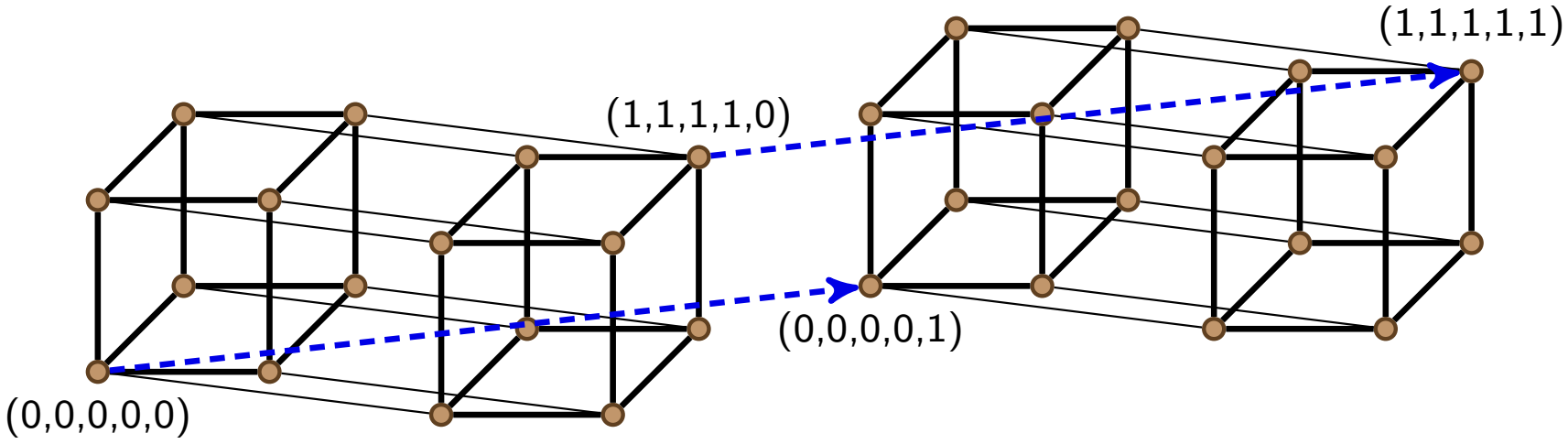
$$\Pr(x^{(t+1)} = y \mid x^{(t)} = x) = \begin{cases} \frac{1}{n} & \text{if } \mathrm{Ham}(y,x) = 1 \\ 0 & \text{otherwise} \end{cases}$$



The mixing time and hitting time on HYP_n

- Mixing time: after how many steps of random walk on HYP_n is the distribution of the vertex that we are at identical to the uniform distribution?
- For **uniform** random walk, $\approx \frac{1}{4} n \log n$, cutoff window of size n

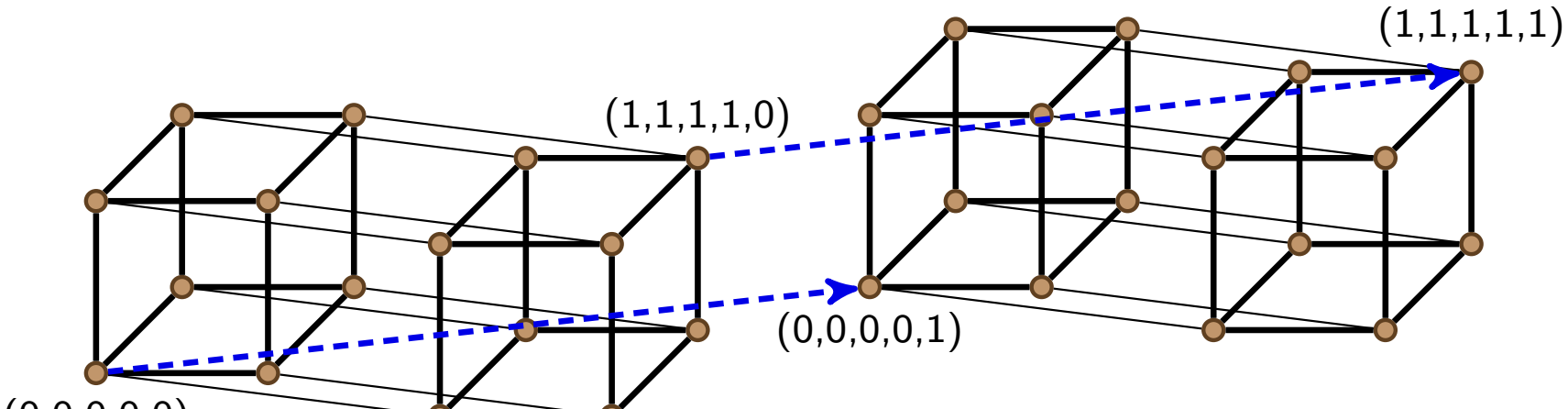
$$\Pr(x^{(t+1)} = y \mid x^{(t)} = x) = \begin{cases} \frac{1}{n+1} & \text{if } \mathrm{Ham}(y, x) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



The mixing time and hitting time on HYP_n

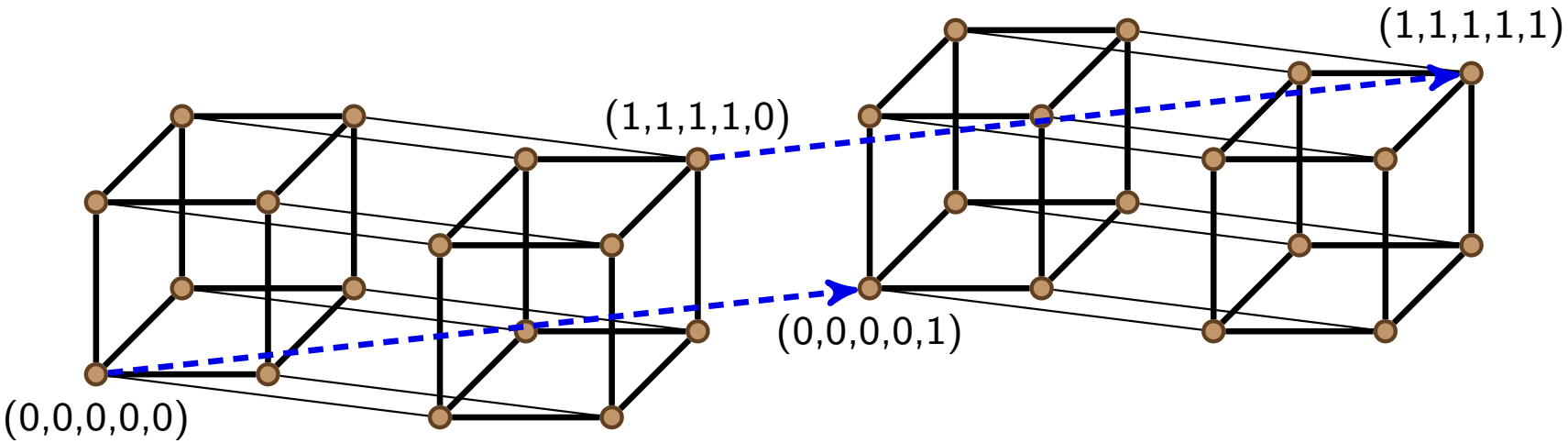
- Mixing time: after how many steps of random walk on HYP_n is the distribution of the vertex that we are at identical to the uniform distribution?
- For **lazy** random walk, $\approx \frac{1}{2} n \log n$, proof of expectation is easy

$$\Pr(x^{(t+1)} = y \mid x^{(t)} = x) = \begin{cases} \frac{1}{2n} & \text{if } \mathrm{Ham}(y, x) = 1 \\ \frac{1}{2} & \text{if } y = x \\ 0 & \text{otherwise} \end{cases}$$



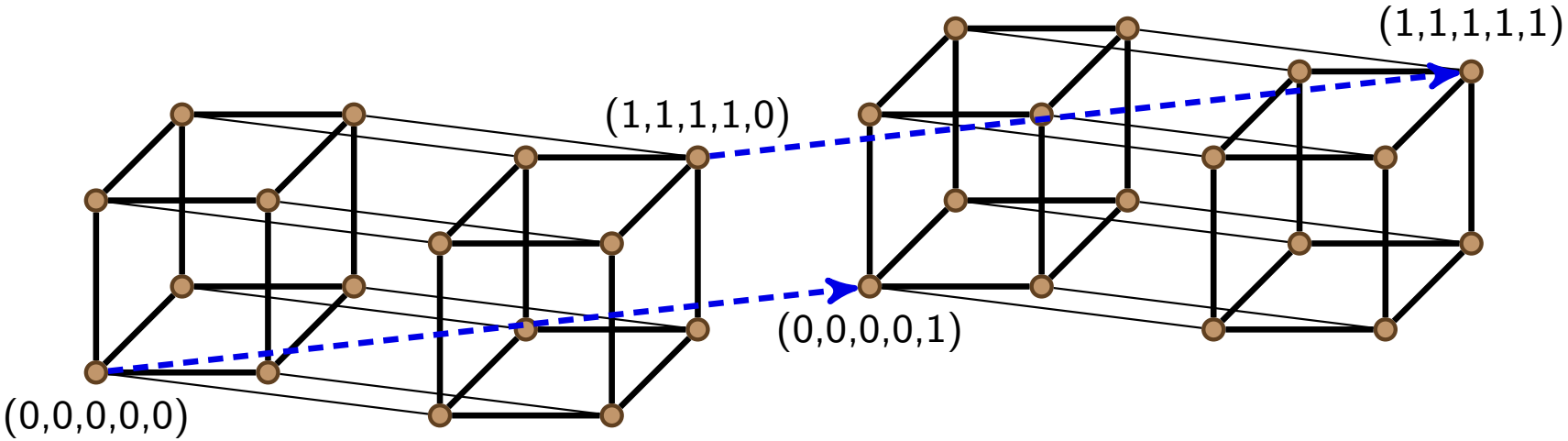
The mixing time and hitting time on HYP_n

- Hitting time: given two vertices, e.g. $x = (0, 0, \ldots, 0)$ and $y = (1, 1, \ldots, 1)$, how many random walk steps on HYP_n would it take to reach from x to y?



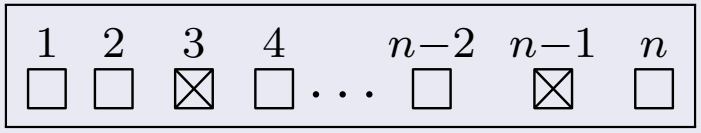
The mixing time and hitting time on HYP_n

- Hitting time: given two vertices, e.g. $x = (0, 0, \ldots, 0)$ and $y = (1, 1, \ldots, 1)$, how many random walk steps on HYP_n would it take to reach from x to y?
- Answer: $\Omega(2^n)$ expected time for any two different vertices
- In fact, hitting time between neighbors and hitting time between opposite vertices differ by a factor of at most $(1 + \frac{1}{n})$



Mixing time of the lazy random walk

The coupon collector's problem



- Consider the stochastic process on $\{0,1\}^n$ where in each step an index $1 \leq i \leq n$ is selected uniformly with probability $\frac{1}{n}$, and then with probability $\frac{1}{2}$ the i^{th} bit is flipped.
- Observe that this stochastic process is identical to the lazy random walk stochastic process.
- Let Y_j be the random variable that counts the number of steps since $j-1$ unique indices were already selected until a new unique index is selected.
- $Y_1 \sim \text{Geo}(1), Y_2 \sim \text{Geo}(\frac{n-1}{n}), \ldots, Y_n \sim \text{Geo}(\frac{1}{n})$
- All the bits are uniformly distributed after n unique indices were selected.

Mixing time of the lazy random walk (cont.)

The coupon collector's problem (cont.)

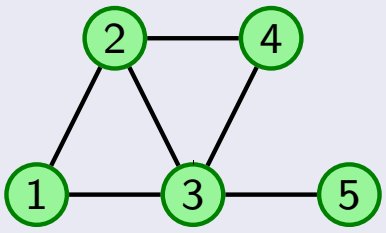
$\Rightarrow$ the expected mixing time is

$$\begin{aligned} \mathsf{Ex}\left[\sum_{j=1}^{n} Y_j\right] &= \sum_{j=1}^{n} \mathsf{Ex}[Y_j] = \sum_{j=1}^{n} \frac{n}{n-j+1} = n \cdot \sum_{j=1}^{n} \frac{1}{j} \\ &= n \cdot H_n \approx n(\gamma + \log n) < n(1 + \log n), \end{aligned}$$

where H_n is the partial harmonic sum, and $\gamma \approx 0.577$ is the Euler constant. $\square$

Hitting time of the simple random walk

Example graph



Transition matrix

$$P = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Step

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}^T P = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 0 \\ 0 \end{pmatrix}^T$$

Stationary distribution of simple random walk on a graph

$\forall y: \ \sum_{x \in V} deg(x) P(x,y) = \sum_{(x,y) \in E} \frac{deg(x)}{deg(x)} = deg(y)$

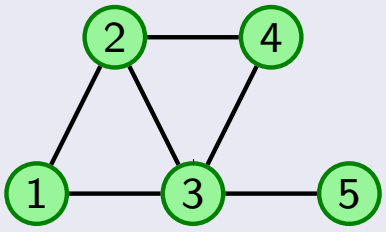
$\Rightarrow$ For $\tilde{\pi} = (deg(v_1), deg(v_2), \ldots, deg(v_{|V|})$, it holds that $\tilde{\pi} = \tilde{\pi} P$

$\Rightarrow$ The normalized vector $\pi = \frac{1}{2|E|} \tilde{\pi}$ is the stationary distribution

$\Rightarrow$ The expected return-time for any vertex v is $\frac{2|E|}{deg(v)}$

Hitting time of the simple random walk

Example graph



Transition matrix

$$P = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Step

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}^T \quad P^2 = \begin{pmatrix} \frac{1}{6} + \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{6} \\ \frac{1}{6} + \frac{1}{8} \\ \frac{1}{8} \end{pmatrix}^T$$

Stationary distribution of simple random walk on a graph

$\forall y: \ \sum_{x \in V} deg(x) P(x,y) = \sum_{(x,y) \in E} \frac{deg(x)}{deg(x)} = deg(y)$

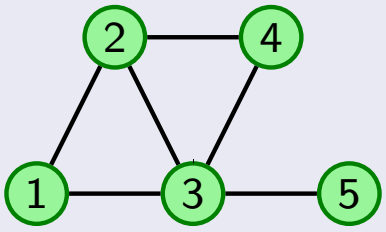
$\Rightarrow$ For $\tilde{\pi} = (deg(v_1), deg(v_2), \ldots, deg(v_{|V|})$, it holds that $\tilde{\pi} = \tilde{\pi} P$

$\Rightarrow$ The normalized vector $\pi = \frac{1}{2|E|} \tilde{\pi}$ is the stationary distribution

$\Rightarrow$ The expected return-time for any vertex v is $\frac{2|E|}{deg(v)}$

Hitting time of the simple random walk

Example graph



Transition matrix

$$P = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Step

$$\begin{pmatrix} 2 \\ 3 \\ 4 \\ 2 \\ 1 \end{pmatrix}^T P^i = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 2 \\ 1 \end{pmatrix}^T$$

Stationary distribution of simple random walk on a graph

$\forall y: \ \sum_{x \in V} deg(x) P(x,y) = \sum_{(x,y) \in E} \frac{deg(x)}{deg(x)} = deg(y)$

$\Rightarrow$ For $\tilde{\pi} = (deg(v_1), deg(v_2), \ldots, deg(v_{|V|})$, it holds that $\tilde{\pi} = \tilde{\pi} P$

$\Rightarrow$ The normalized vector $\pi = \frac{1}{2|E|} \tilde{\pi}$ is the stationary distribution

$\Rightarrow$ The expected return-time for any vertex v is $\frac{2|E|}{deg(v)}$

Hitting time of the simple random walk (cont.)

Remarkable fact that follows from $\pi_v = \frac{deg(v)}{2|E|}$

In an infinite simple random walk on any connected graph, each edge will be traversed the same proportion of the time.

Proof of exponential hitting time on the hypercube

- HYP_n is n-regular with 2^n vertices and $\frac{1}{2}n2^n$ edges
- The expected return-time for any vertex is $\frac{2\cdot|E|}{n} = 2^n$
- $hit_n(i) \triangleq$ the expected time to reach $\overrightarrow{0} = (0, 0, \ldots, 0)$ from an assignment with exactly i bits whose value is 1
- $hit_n(i)$ is monotone increasing in i
- $2^n = returntime_n(\overrightarrow{0}) = 1 + hit_n(1)$
- $\Rightarrow hit_n(1) = 2^n - 1$
- $\Rightarrow hit_n(n) > hit_n(1) = 2^n - 1$

Learning $\mathcal{O}(\log n)$-juntas

k-juntas: functions with at most $k \ll n$ relevant variables.

Motivation for practical applications

- Learnability in the presence of irrelevant information is a real-world task in the field of machine learning.
- For example, suppose that each query represents a long DNA sequence, and the boolean target function is some biological property that depends only on a small unknown active part.

Theoretical Motivation

- Shed light on the learnability of poly-size DNF, which is a major open question even for PAC under uniform distribution.
- Any $(d \cdot \log n)$-junta is a n^d-term DNF, so poly-size DNF learnability implies $\mathcal{O}(\log n)$-juntas learnability.
- Conversely, a decision tree with k leaves is a k-junta, therefore learning k-juntas implies learning k-size decision trees.

Learning $\mathcal{O}(\log n)$-juntas from membership queries (MQs)

- In passive learning models such as UOnline and uniform PAC, learning k-juntas is an open question even for $k = \omega(1)$
- if the learner can control the examples by actively making MQs, learning k-juntas in $poly(2^k, n)$ time becomes easy:
  - draw random examples, if all are positive or negative, done
  - discover a relevant variable by walking via MQs, and recurse

	x_1	x_2	x_3	x_4	x_5	x_6	
$x^{(1)}$	1	0	0	0	0	0	$f = 1$
	1	0	1	0	0	0	$f = 1$
	1	0	1	1	$\boxed{0}$	0	$f = \boxed{1}$
	1	0	1	1	$\boxed{1}$	0	$f = \boxed{0}$
$x^{(2)}$	1	0	1	1	1	1	$f = 0$

Here the relevant variable x_5 was discovered from a positive example $x^{(1)} = (1, 0, 0, 0, 0, 0)$ and a negative example $x^{(2)} = (1, 0, 1, 1, 1, 1)$

- Deterministic MQs algorithm can also be obtained by using a combinatorial construction known as (n, k)-universal set

Learning $\mathcal{O}(\log n)$-juntas in the URWOnline model - proof outline

Similarity between MQs and URWOnline implies $\leq \tilde{\mathcal{O}}(2^k)$ mistakes

- The learner in URWOnline cannot control the examples.
- But the random walk properties can be exploited to trigger the discovery of each of the k relevant variables.
- Suppose the learner simply always predicts $h(x^{(t)}) = f(x^{(t-1)})$
  1. After $\approx k \log k$ steps, the example $x^{(t)}$ is uniformly distributed.
  2. If x_i is a relevant variable, the example $x^{(t)}$ can trigger the discovery of x_i with probability $\geq \frac{2}{2^k}$
  3. With probability $\frac{1}{k}$, the learner will discover x_i at the next trial.
- $\Rightarrow$ after $\mathcal{O}(2^k k \cdot (k \log k))$ trials, any relevant variable that is still unknown would be discovered with constant probability.
- After the k variables are found, the learner can construct a truth table of size 2^k, and learn f by making $\leq 2^k$ mistakes.
- $k = d \cdot \log n \implies \tilde{\mathcal{O}}(2^{d \cdot \log n}) = \tilde{\mathcal{O}}(n^d)$ mistakes.

Learning $\mathcal{O}(\log n)$-juntas in the URWOnline model - extensions

Unknown k

- If k is unknown to the learner, we can still achieve $\tilde{\mathcal{O}}(2^k)$ bound.
- Increment k and re-invoke the algorithm in case no relevant variables were discovered after $\mathcal{O}(2^k k^2 \log k)$ mistakes, or in case the constructed truth table is erroneous.

Minimal sensitivity

- The influence of a variable x_i on f is defined as the probability that $f(x) \neq f(x \oplus e_i)$ where x is chosen uniformly randomly from $\{0,1\}^n$
- The minimal sensitivity of f is defined as the smallest influence among all of its (relevant) variables.
- If the minimal sensitivity is $\frac{1}{S}$, then $\frac{2}{2^k}$ can be replaced with $\frac{1}{S}$ in the analysis. If further knowledge is available, building the truth table can replaced by an Online learning algorithm for the class, which would be executed on the relevant variables.
- Minimal sensitivity of parity functions is $1 \Rightarrow$ exponential speedup.

Learning read-once monotone DNF in the URWOnline model

$f \in$ ROM-DNF, e.g. $f(x_1, x_2, \ldots, x_9) = x_1x_3 \vee x_4 \vee x_2x_8x_9$

Learning algorithm and data structure for ROM-DNF

- Initialize $T_{x_i} \leftarrow \{x_1, x_2, \ldots, x_n\}$ for every x_i, i.e. assume that the variables in the term that contains x_i are $\{x_1, x_2, \ldots, x_n\}$
- Whenever possible, we make a prediction that would eliminate variables from these term sets in case the prediction is wrong.
- For example, if $f(x^{(t-1)}) = 0$, x_i flipped $0 \to 1$, and T_{x_i} is unsatisfied at trial t, we predict $h(x^{(t)}) = 0$, and if we are mistaken we can remove the superfluous variables from T_{x_i}

	x_1	x_2	x_3	x_4	x_5	x_6	
$x^{(t-1)}$	0	1	0	1	0	0	$f = 0$
$x^{(t)}$	1	1	0	1	0	0	$f = 1$

Suppose $T_{x_1} = \{x_1, x_4, x_5, x_6\} \Rightarrow$ we remove x_5, x_6 from T_{x_1} and also remove x_1 from T_{x_5} and from T_{x_6}

Learning read-once monotone DNF in the URWOnline model (cont.)

Learning algorithm and data structure for ROM-DNF (cont.)

- If we could eliminate variables after every mistake, we would obtain $\mathcal{O}(n^2)$ mistake bound for adversarial RWOnline
- This would imply Online learnability of ROM-DNF (and DNF)
- As expected, there is a case where we have to make mistakes without guaranteed removal of variables from the term sets:
  $f(x^{(t-1)}) = 1,\ T_{x_i}(x^{(t-1)}) = 1,\ x_i$ flipped $1 \to 0$, and for every k, $T_{x_k}(x^{(t)}) = 0$

  We know that the term of x_i was satisfied at trial $t-1$ and is unsatisfied at trial t, but we don't know whether any of the terms is also satisfied. Extremes: $f = \wedge_{i=1}^{n} x_i$ vs $f = \vee_{i=1}^{n} x_i$
- Therefore the algorithm alternates between two modes.
- Mode "A" assumes that the sets T_{x_i} are correct $\Rightarrow$ predicts 0
- Mode "B" assumes that surplus variables in the sets T_{x_i} prevent the algorithm from seeing satisfied terms $\Rightarrow$ predicts 1

The read-once monotone DNF learning algorithm

1. $MODE \leftarrow$ "A", and for each variable x_i, $1 \leq i \leq n$, create the set $T_{x_i} \leftarrow \{x_1, x_2, \ldots, x_n\}$
2. Trial t: determine $f(x^{(t-1)})$ by checking if the teacher returned "mistake", let x_i be the flipped variable
3. If $T_{x_i} = \emptyset$ (meaning: x_i isn't relevant), then predict $h(x^{(t)}) = f(x^{(t-1)})$
4. Otherwise, if $f(x^{(t-1)}) = 0$
   (a) If x_i flipped $1 \rightarrow 0$, then predict 0
   (b) Otherwise, x_i flipped $0 \rightarrow 1$
      i. If $T_{x_i}(x^{(t)}) = 1$, then predict 1
         On mistake do: $T_{x_i} \leftarrow \emptyset$, and update the other term sets by removing x_i from them.
      ii. Otherwise, predict 0
         On mistake do: update the set T_{x_i} by removing the unsatisfied variables of $x^{(t)}$ from it, since they are unneeded, and update the rest of the term sets by removing x_i from any term set T_{x_k} such that x_k was an unneeded variable in T_{x_i}
5. Otherwise, $f(x^{(t-1)}) = 1$
   (a) If x_i flipped $0 \rightarrow 1$, then predict 1
   (b) Otherwise, x_i flipped $1 \rightarrow 0$
      i. If some $T_{x_k}(x^{(t)}) = 1$, then predict 1
         On mistake do: for each k such that $T_{x_k}(x^{(t)}) = 1$, do $T_{x_k} \leftarrow \emptyset$, and remove the irrelevant variable x_k from the rest of the term sets
      ii. Otherwise, if $T_{x_i}(x^{(t-1)}) = 0$, then predict 1
         On mistake do: update the set T_{x_i} by removing the unsatisfied variables of $x^{(t-1)}$ from it, since they are unneeded, and update the rest of the term sets by removing x_i in any term set T_{x_k} such that x_k was an unneeded variable in T_{x_i}
      iii. Otherwise, if $MODE =$ "A", then predict 0
         On mistake do: $MODE \leftarrow$ "B"
         Otherwise, $MODE =$ "B", then predict 1
         On mistake do: $MODE \leftarrow$ "A"
6. Goto 2

Analysis of the ROM-DNF algorithm in the URWOnline model

- For one of the modes "A" or "B", we bound the ratio between the number of assignments that could cause noninformative mistakes and the number of assignments that could cause informative mistakes during any stage of the learning process.
- Therefore, in URWOnline we make progress in learning the target function f by analysing examples after $\mathcal{O}(n \log n)$ trials.

At trial t, let $f = f_1 \vee f_2$ where

- $f_1 = \hat{T}_1^f \vee \hat{T}_2^f \vee \cdots \vee \hat{T}_{k_1}^f$ with $a_\ell \triangleq |\hat{T}_\ell^f|$ are the terms in f where for every term $\hat{T}_\ell^f$ there exists a variable x_j in that term such that $T_{x_j} = \hat{T}_\ell^f$. These terms that have been discovered.
- $f_2 = T_1^f \vee T_2^f \vee \cdots \vee T_{k_2}^f$ with $b_\ell \triangleq |T_\ell^f|$ are the terms in f where for every term T_ℓ^f and every variable x_j in that term, we have that T_{x_j} is a proper super-term of T_ℓ^f. For each x_i that belongs to such a term, the set T_{x_i} contains surplus variables.

Analysis of the ROM-DNF algorithm in the URWOnline model (cont.)

- $N_A \triangleq$ the number of noninformative assignments in mode "A"
- $N_B \triangleq$ the number of noninformative assignments in mode "B"

$$\begin{aligned} N_A &\leq |\{x^{(t)} \in \{0,1\}^n \mid f_1(x^{(t)}) = 0 \text{ and } f_2(x^{(t)}) = 1\}| \\ &= c2^d \left(\prod_{i=1}^{k_2} 2^{b_i} - \prod_{i=1}^{k_2} (2^{b_i} - 1) \right) \\ N_B &\leq |\{x^{(t)} \in X_n \mid f_1(x^{(t)}) = 0 \text{ and } f_2(x^{(t)}) = 0\}| \\ &= c2^d \prod_{i=1}^{k_2} (2^{b_i} - 1) \end{aligned}$$

- $c = \prod_{i=1}^{k_1} (2^{a_i} - 1)$ is the number of assignments for $f_1(x) = 0$
- d denotes the number of irrelevant variables

Analysis of the ROM-DNF algorithm in the URWOnline model (cont.)

- $N \triangleq$ the number of informative assignments
- Consider case (4(b)ii) of the algorithm
  - $f_1(x^{(t)}) = 0$
  - exactly one term of f_2 satisfies $x^{(t)}$
  - one superfluous variable of the corresponding term set is 0
- $\sum_{j=1}^{k} \prod_{i \neq j}^{k} (2^{b_i} - 1)$ is the number of assignments in which exactly one of the terms in f_2 is satisfied.
- At least half of these assignment have at least one superfluous variable set to 0, because f is monotone.
- $\Rightarrow \quad N \geq \frac{1}{2} c 2^d \sum_{j=1}^{k_2} \prod_{i \neq j}^{k_2} (2^{b_i} - 1)$
- $\frac{N_A}{N} \leq \frac{2(\prod_{i=1}^{k_2} 2^{b_i} - \prod_{i=1}^{k_2} (2^{b_i} - 1))}{\sum_{j=1}^{k_2} \prod_{i \neq j}^{k_2} (2^{b_i} - 1)}, \quad \frac{N_B}{N} \leq \frac{2 \prod_{i=1}^{k_2} (2^{b_i} - 1)}{\sum_{j=1}^{k_2} \prod_{i \neq j}^{k_2} (2^{b_i} - 1)}$

Analysis of the ROM-DNF algorithm in the URWOnline model (cont.)

$$w_i := 2^{b_i} - 1, \quad \alpha := \frac{\prod_{i=1}^k (w_i+1) - \prod_{i=1}^k w_i}{\sum_{j=1}^k \prod_{i\neq j}^k w_i}, \quad \beta := \frac{\prod_{i=1}^k w_i}{\sum_{j=1}^k \prod_{i\neq j}^k w_i}$$

$$\begin{aligned}
\beta &= \frac{\prod_{i=1}^k w_i}{\left(\prod_{i=1}^k w_i\right)\sum_{i=1}^k \frac{1}{w_i}} = \frac{1}{\sum_{i=1}^k \frac{1}{w_i}} \\
\alpha &= \frac{\prod_{i=1}^k (w_i+1) - \prod_{i=1}^k w_i}{\left(\prod_{i=1}^k w_i\right)\sum_{i=1}^k \frac{1}{w_i}} \\
&= \frac{1}{\sum_{i=1}^k \frac{1}{w_i}} \left(\frac{\prod_{i=1}^k (w_i+1)}{\prod_{i=1}^k w_i} - 1 \right) \\
&= \beta \left(\prod_{i=1}^k \left(1 + \frac{1}{w_i}\right) - 1 \right) \\
&\leq \beta \left(\prod_{i=1}^k e^{\frac{1}{w_i}} - 1 \right) = \beta \left(e^{\sum_{i=1}^k \frac{1}{w_i}} - 1 \right) = \beta(e^{\frac{1}{\beta}} - 1)
\end{aligned}$$

$$\min\left(\frac{N_A}{N}, \frac{N_B}{N}\right) = 2\min(\alpha,\beta) \;\leq\; 2\min(\beta(e^{\frac{1}{\beta}} - 1), \beta) = 2\frac{1}{\log 2} < 2 \times 1.443 \;<\; 3$$

Extending the ROM-DNF algorithm to handle non-monotone RO-DNF

- The analysis takes a penalty since the bound for N assumes that f is monotone.
- Initialize T_{x_i} to $\{\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \ldots, \tilde{x}_n\}$, meaning that we do not know yet whether the variables of the term that contains x_i are negated or not.
- Always predict $h(x^{(t)}) = f(x^{(t-1)})$ when x_i flips and T_{x_i} contain variables that are marked as unknown. If we make a mistake on such predictions, we can immediately update variables in relation to x_i as follows.

- if $f(x^{(t)}) = x_i^{(t)}$ then $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_i\}) \cup \{x_i\}$
  else $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_i\}) \cup \{\bar{x}_i\}$
- for each $j \neq i$
  - if $f(x^{(t)}) = x_i^{(t)}$
    - if $\bar{x}_i \in T_{x_j}$ then $T_{x_j} \leftarrow T_{x_j} \setminus \{\bar{x}_i\}$ else $T_{x_j} \leftarrow (T_{x_j} \setminus \{\tilde{x}_i\}) \cup \{x_i\}$
  - else $f(x^{(t)}) \neq x_i^{(t)}$
    - if $x_i \in T_{x_j}$ then $T_{x_j} \leftarrow T_{x_j} \setminus \{x_i\}$ else $T_{x_j} \leftarrow (T_{x_j} \setminus \{\tilde{x}_i\}) \cup \{\bar{x}_i\}$
- for each $j \neq i$
  - if $x_j^{(t)} = 1$
    - if $\bar{x}_j \in T_{x_i}$ then $T_{x_i} \leftarrow T_{x_i} \setminus \{\bar{x}_j\}$ else $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_j\}) \cup \{x_j\}$
  - else $x_j^{(t)} = 0$
    - if $x_j \in T_{x_i}$ then $T_{x_i} \leftarrow T_{x_i} \setminus \{x_j\}$ else $T_{x_i} \leftarrow (T_{x_i} \setminus \{\tilde{x}_j\}) \cup \{\bar{x}_j\}$

Could any poly-size DNF be Learned in the URWOnline model?

Required assumptions with regard to the read-3 DNF algorithm

- Conservative Online - the learning algorithm doesn't modify the hypothesis after successful predictions (this typically holds)
- White box that is susceptible to manual inclusion of knowledge
  - The new knowledge should be retained and utilized if it is consistent with the target function f
  - For example, adding terms from f to a list of terms or circuit that is maintained by h
  - Holds for our RO-DNF algorithm in the URWOnline model

Learning read-3 DNF in URWOnline implies learning any DNF in UOnline

- Suppose $f(x_1, x_2, \ldots, x_n) = T_1 \vee T_2 \vee \ldots \vee T_q$ is a general DNF formula that consists of q terms, and define

$$\begin{aligned} R3(f,k) &\triangleq f'(\overbrace{x_{11}, x_{12}, \ldots, x_{1k}}, \overbrace{x_{21}, x_{22}, \ldots, x_{2k}}, \ldots, \overbrace{x_{n1}, x_{n2}, \ldots, x_{nk}}) \\ &= \tilde{T}_1 \vee \tilde{T}_2 \vee \ldots \vee \tilde{T}_q \vee x_{11}\bar{x}_{12} \vee x_{12}\bar{x}_{13} \vee \ldots \vee x_{1(k-1)}\bar{x}_{1k} \vee x_{1k}\bar{x}_{11} \\ &\quad \vee \; x_{21}\bar{x}_{22} \vee x_{22}\bar{x}_{23} \vee \ldots \vee x_{2(k-1)}\bar{x}_{2k} \vee x_{2k}\bar{x}_{21} \\ &\quad \vee \; \ldots \\ &\quad \vee \; x_{n1}\bar{x}_{n2} \vee x_{n2}\bar{x}_{n3} \vee \ldots \vee x_{n(k-1)}\bar{x}_{nk} \vee x_{nk}\bar{x}_{n1} \end{aligned}$$

  where each $\tilde{T}_i$ contains only variables from $\{x_{1i}, x_{2i}, \ldots, x_{ni}\}$, corresponding to the variables from $\{x_1, x_2, \ldots, x_n\}$ that T_i contains.

- $R3(f,k)$ is a read-3 DNF formula of size $\mathcal{O}(q + kn)$, and for $k \geq q$ we have

$$R3(f,k) \equiv \begin{cases} 1 & \exists i_0, i_1, i_2 : x_{i_0 i_1} \neq x_{i_0 i_2} \\ f(x_{11}, x_{21}, \ldots, x_{n1}) & \text{otherwise} \end{cases}$$

Simulating the read-3 DNF algorithm in the UOnline model

The simulation of RWR3-L by UDNF-L

- Let RWR3-L(n, δ) be the read-3 DNF algorithm for the URWOnline model that is assumed to exist.
- UDNF-L(n, δ) will insert RWDNF-L$(kn, \frac{\delta}{2})$ the knowledge to predict 1 on all queries in which $\exists x_{i_0 i_1} \neq x_{i_0 i_2}$, e.g. by adding the kn needed terms to the RWDNF-L$(kn, \frac{\delta}{2})$ white box.
- UDNF-L(n, δ) will simulate RWDNF-L$(kn, \frac{\delta}{2})$ on $R3(f, k)$ by
  - Taking each uniformly distributed query that is received from the actual teacher
  - Choosing randomly uniformly an index $1 \leq i \leq kn$ as if it was the last to flip
  - Duplicating each variable k times
  - Invoking RWDNF-L$(kn, \frac{\delta}{2})$ on each such expanded query
  - Returning the prediction of RWDNF-L$(kn, \frac{\delta}{2})$ to the teacher
  - Updating the hypothesis according to the RWDNF-L$(kn, \frac{\delta}{2})$ algorithm in case of a prediction mistake

Correctness of the simulation

When is this simulation correct?

- Because RWDNF-L($kn, \frac{\delta}{2}$) neither makes mistakes nor updates its state whenever $\exists x_{i_0 i_1} \neq x_{i_0 i_2}$, this simulation makes the assumption that every time that the random walk stochastic process reaches an assignment for which $\nexists x_{i_0 i_1} \neq x_{i_0 i_2}$, and RWDNF-L($kn, \frac{\delta}{2}$) makes a prediction mistake on it, that assignment is uniformly distributed.
- We prove that this assumption holds with a very high probability, by using the fact that the expected hitting time on the hypercube is exponential.

Analysis of the simulation

$Y \triangleq \{\text{number of steps on } x_{11}, \ldots, x_{nk} \text{ (without } x_{d1}, \ldots, x_{dk}) \text{ until all were selected}\}$

$Z \triangleq \{\text{number of steps on } x_{d1}, \ldots, x_{dk} \text{ until reaching } x_{d1} = x_{d2} = \ldots = x_{dk} \text{ flipped}\}$

Let Y' denote the number of steps on $\{x_{11}, \ldots, x_{nk}\} \setminus \{x_{d1}, \ldots, x_{dk}\}$ until $\{x_{d1}, \ldots, x_{dk}\}$ reached the exact opposite assignment.

Here we assume $k \geq 9n$.

$$\begin{aligned}
\Pr(\overline{\mathcal{A}^1_{\text{bad-dist}}}) &\geq \Pr(\overline{\mathcal{A}^1_{\text{bad-dist}}} | Y' > 2^{5n}) \cdot \Pr(Y' > 2^{5n}) \\
&\geq \Pr(Y < 2^{5n}) \cdot \Pr(Y' > 2^{5n}) \\
&\geq (1 - \frac{\mathsf{Ex}[Y]}{2^{5n}}) \cdot \Pr(Y' > 2^{5n}) \\
&> (1 - \frac{9n^2 \log(9n^2)}{2^{5n}}) \cdot \Pr(Y' > 2^{5n} | Z > 2^{5n}) \cdot \Pr(Z > 2^{5n}) \\
&> (1 - \frac{1}{2^{4n}}) \cdot \Pr(Y' > 2^{5n} | Z = 2^{5n}) \cdot \Pr(Z > 2^{5n})
\end{aligned}$$

Notice that for $X \sim \mathsf{NegBin}(2^{5n}, \frac{1}{n})$, we have

$$\Pr(Y' \leq 2^{5n} | Z = 2^{5n}) = \Pr(X \leq 2 \cdot 2^{5n}),$$
$$\mathsf{Ex}[X] = 2^{5n}(n-1), \ \mathsf{Var}[X] = 2^{5n}(n-1)n.$$

Thank you.