Quantum and Classical Algorithms for Approximate Submodular Function Minimization

Yassine Hamoudi, Patrick Rebentrost,

Ansis Rosmanis, Miklos Santha

arXiv: 1907.05378

1. Approximate Submodular Function Minimization

2. Quantum speed-up for Importance Sampling

1

Approximate Submodular Function Minimization

Submodular Function

A **submodular function** is a set function $F : 2^{[n]} \to \mathbb{R}$ satisfying the **diminishing returns property**:

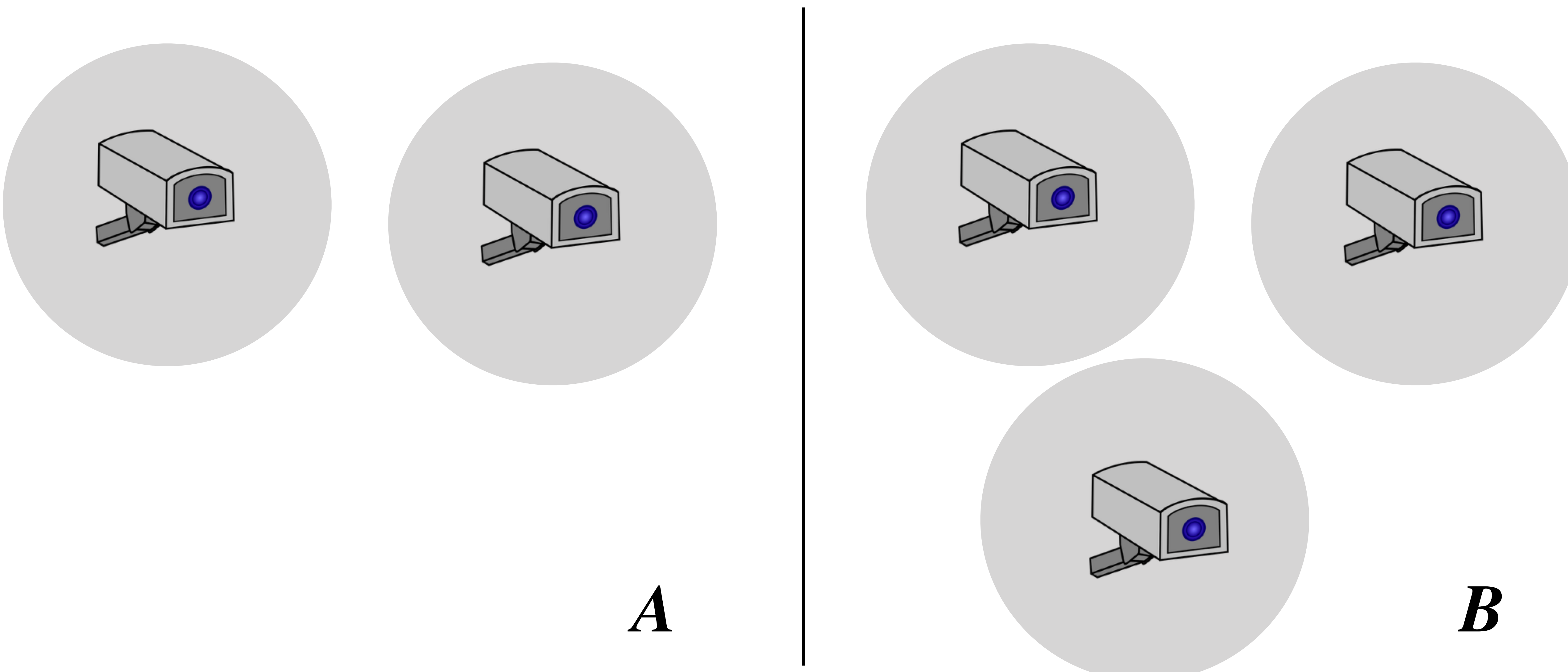
$$\forall A \subset B \subset [n] \text{ and } i \notin B, \quad F(A \cup \{i\}) - F(A) \geq F(B \cup \{i\}) - F(B)$$

Submodular Function

A **submodular function** is a set function $F : 2^{[n]} \to \mathbb{R}$ satisfying the **diminishing returns property**:

$$\forall A \subset B \subset [n] \text{ and } i \notin B, \;\; F(A \cup \{i\}) - F(A) \geq F(B \cup \{i\}) - F(B)$$

Example: area covered by cameras

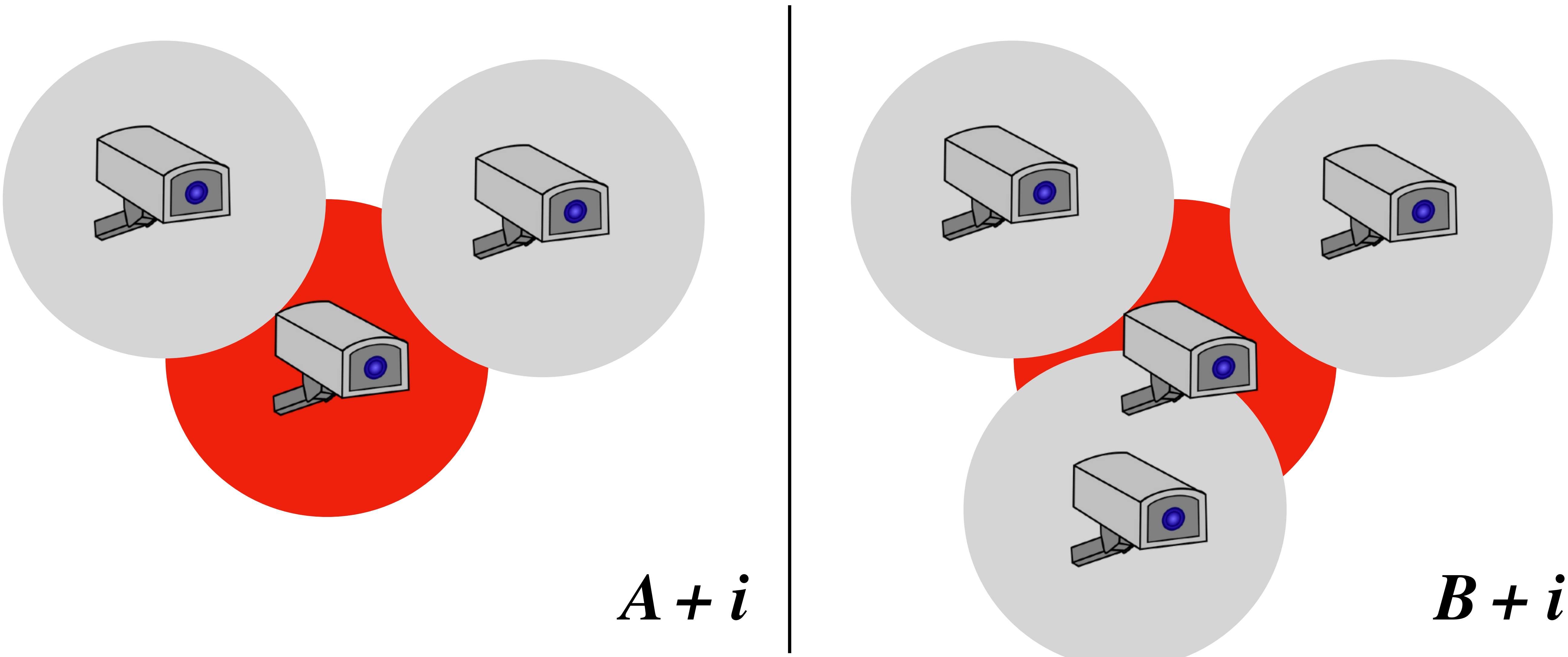


Submodular Function

A **submodular function** is a set function $F : 2^{[n]} \to \mathbb{R}$ satisfying the **diminishing returns property**:

$$\forall A \subset B \subset [n] \text{ and } i \notin B, \;\; F(A \cup \{i\}) - F(A) \geq F(B \cup \{i\}) - F(B)$$

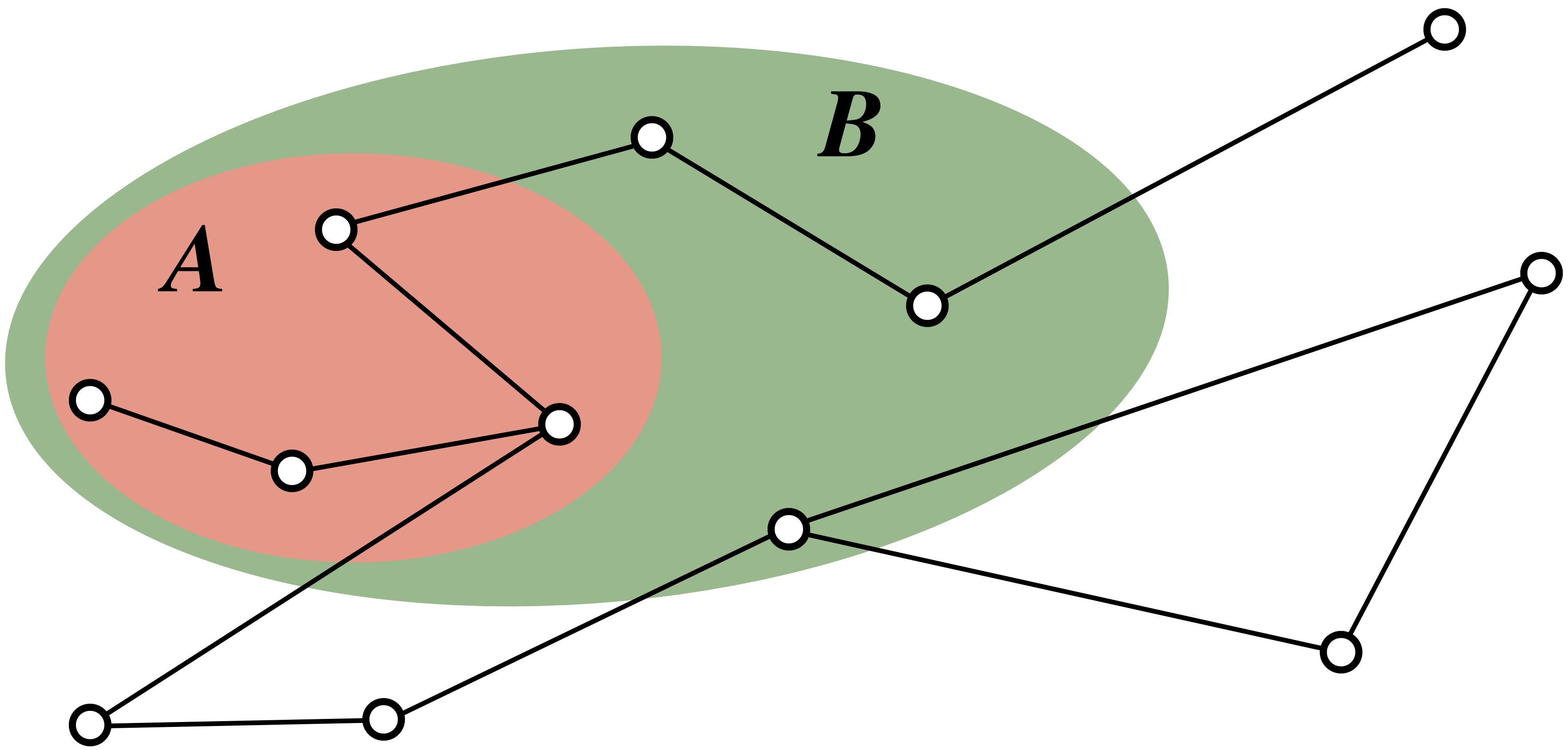
Example: area covered by cameras



A **submodular function** is a set function $F : 2^{[n]} \to \mathbb{R}$ satisfying the **diminishing returns property**:

$$\forall A \subset B \subset [n] \text{ and } i \notin B, \;\; F(A \cup \{i\}) - F(A) \geq F(B \cup \{i\}) - F(B)$$

Example: size of a cut



|cut(A)| = 2

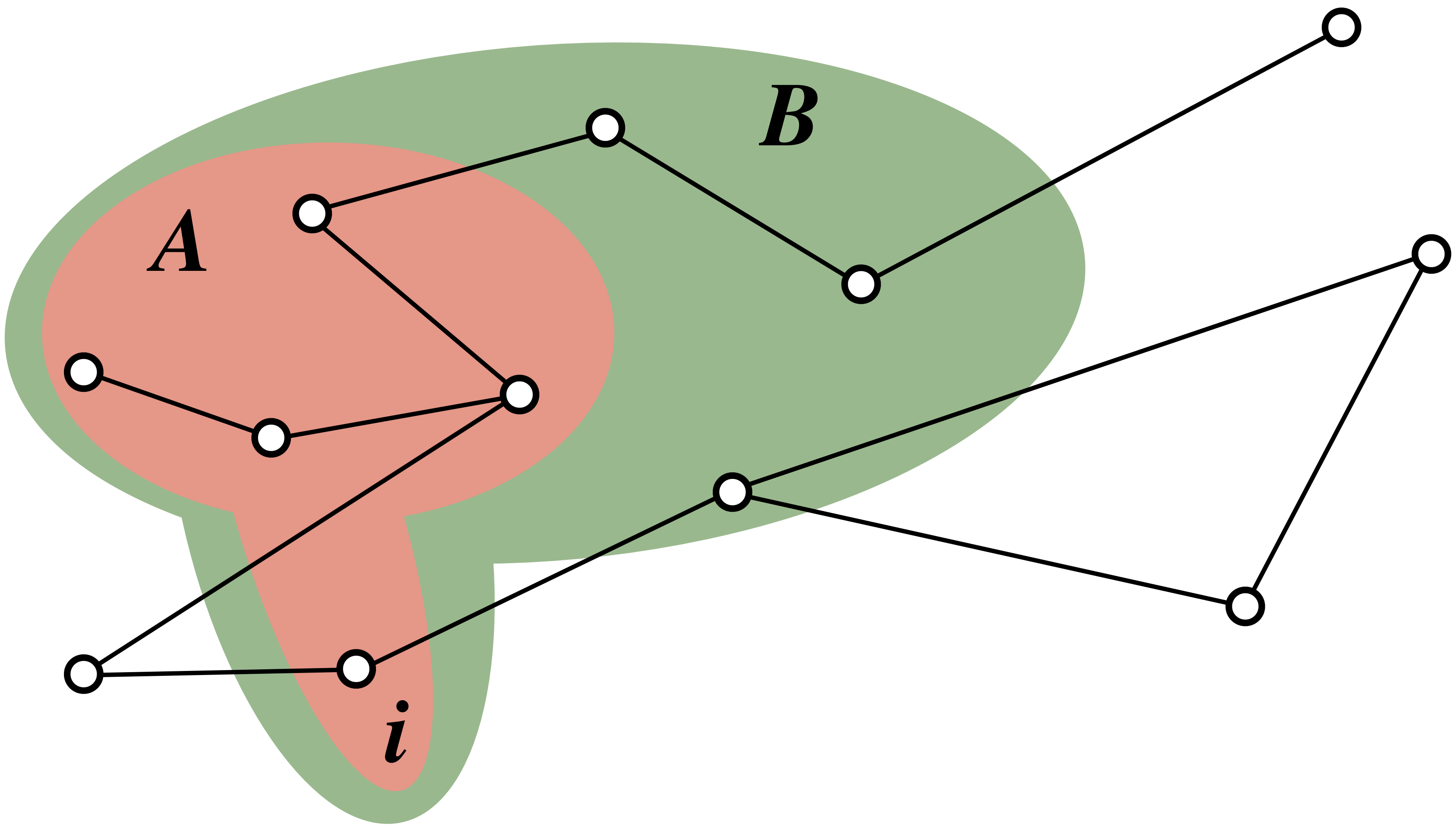
|cut(B)| = 5

Submodular Function

A **submodular function** is a set function $F : 2^{[n]} \to \mathbb{R}$ satisfying the **diminishing returns property**:

$$\forall A \subset B \subset [n] \text{ and } i \notin B,\;\; F(A \cup \{i\}) - F(A) \geq F(B \cup \{i\}) - F(B)$$

Example: size of a cut



|cut(A)| = 2

|cut(B)| = 5

|cut(A+i)| = 4

|cut(B+i)| = 6

Evaluation oracle access: given S obtain $F(S)$. (***time*** **=** *#queries to the oracle*)

Evaluation oracle access: given S obtain $F(S)$. *(**time** = #queries to the oracle)*

Submodular functions can be minimized in polynomial time

(Grotschel, Lovasz, Shrijver 1981)

Submodular Function Minimization

Evaluation oracle access: given S obtain $F(S)$. (***time*** **=** *#queries to the oracle*)

Submodular functions can be minimized in polynomial time

(Grotschel, Lovasz, Shrijver 1981)

Exact Minimization: find $S^\star$ such that $F(S^\star) = \min_{S \subset [n]} F(S)$

- **Lee, Sidford, Wong FOCS'15:** $\tilde{O}(n^3)$ or $\tilde{O}(n^2 \log M)$ where $M = \max |F(S)|$

Submodular Function Minimization

Evaluation oracle access: given S obtain $F(S)$. *(**time** = #queries to the oracle)*

<u>Submodular functions can be minimized in polynomial time</u>

(Grotschel, Lovasz, Shrijver 1981)

Exact Minimization: find $S^\star$ such that $F(S^\star) = \min_{S \subset [n]} F(S)$

- **Lee, Sidford, Wong FOCS'15:** $\tilde{O}(n^3)$ or $\tilde{O}(n^2 \log M)$ where $M = \max |F(S)|$

ε-Approx. Minimization: find $S^\star$ such that $F(S^\star) \leq \min_{S \subset [n]} F(S) + \epsilon$

$(F : 2^{[n]} \to [-1,1])$

Submodular Function Minimization

Evaluation oracle access: given S obtain $F(S)$. *(**time** = #queries to the oracle)*

Submodular functions can be minimized in polynomial time

(Grotschel, Lovasz, Shrijver 1981)

Exact Minimization: find $S^\star$ such that $F(S^\star) = \min_{S \subset [n]} F(S)$

- **Lee, Sidford, Wong FOCS'15:** $\tilde{O}(n^3)$ or $\tilde{O}(n^2 \log M)$ where $M = \max |F(S)|$

ε-Approx. Minimization: find $S^\star$ such that $F(S^\star) \leq \min_{S \subset [n]} F(S) + \epsilon$

$(\; F : 2^{[n]} \to [-1,1] \;)$

- **Previous work:** $\tilde{O}(n^{5/3}/\epsilon^2)$ (classical)

(Chakrabarty, Lee, Sidford, Wong STOC'17)

Submodular Function Minimization

Evaluation oracle access: given S obtain $F(S)$. (***time*** *= #queries to the oracle*)

Submodular functions can be minimized in polynomial time
(Grotschel, Lovasz, Shrijver 1981)

Exact Minimization: find $S^\star$ such that $F(S^\star) = \min_{S \subset [n]} F(S)$

- **Lee, Sidford, Wong FOCS'15:** $\tilde{O}(n^3)$ or $\tilde{O}(n^2 \log M)$ where $M = \max |F(S)|$

ε-Approx. Minimization: find $S^\star$ such that $F(S^\star) \leq \min_{S \subset [n]} F(S) + \epsilon$

$(F : 2^{[n]} \to [-1,1])$

- **Previous work:** $\tilde{O}(n^{5/3}/\epsilon^2)$ (classical)

(Chakrabarty, Lee, Sidford, Wong STOC'17)

- **Our result:** $\tilde{O}(n^{3/2}/\epsilon^2)$ (classical) or $\tilde{O}(n^{5/4}/\epsilon^{5/2})$ (quantum)

Discrete Optimization

Set function: $F : 2^{[n]} \to \mathbb{R}$

Discrete Optimization

Set function: $F : 2^{[n]} \rightarrow \mathbb{R}$

$\downarrow$

Continuous Optimization

Lovász extension: $f : [0,1]^n \rightarrow \mathbb{R}$

Discrete Optimization

Set function: $F : 2^{[n]} \to \mathbb{R}$

$\downarrow$

Continuous Optimization

Lovász extension: $f : [0,1]^n \to \mathbb{R}$

$$n = 2$$

$$F(\emptyset) = 0$$
$$F(\{1\}) = 10$$
$$F(\{2\}) = 6$$
$$F(\{1,2\}) = 3$$

Discrete Optimization

Set function: $F : 2^{[n]} \to \mathbb{R}$

Continuous Optimization

Lovász extension: $f : [0,1]^n \to \mathbb{R}$

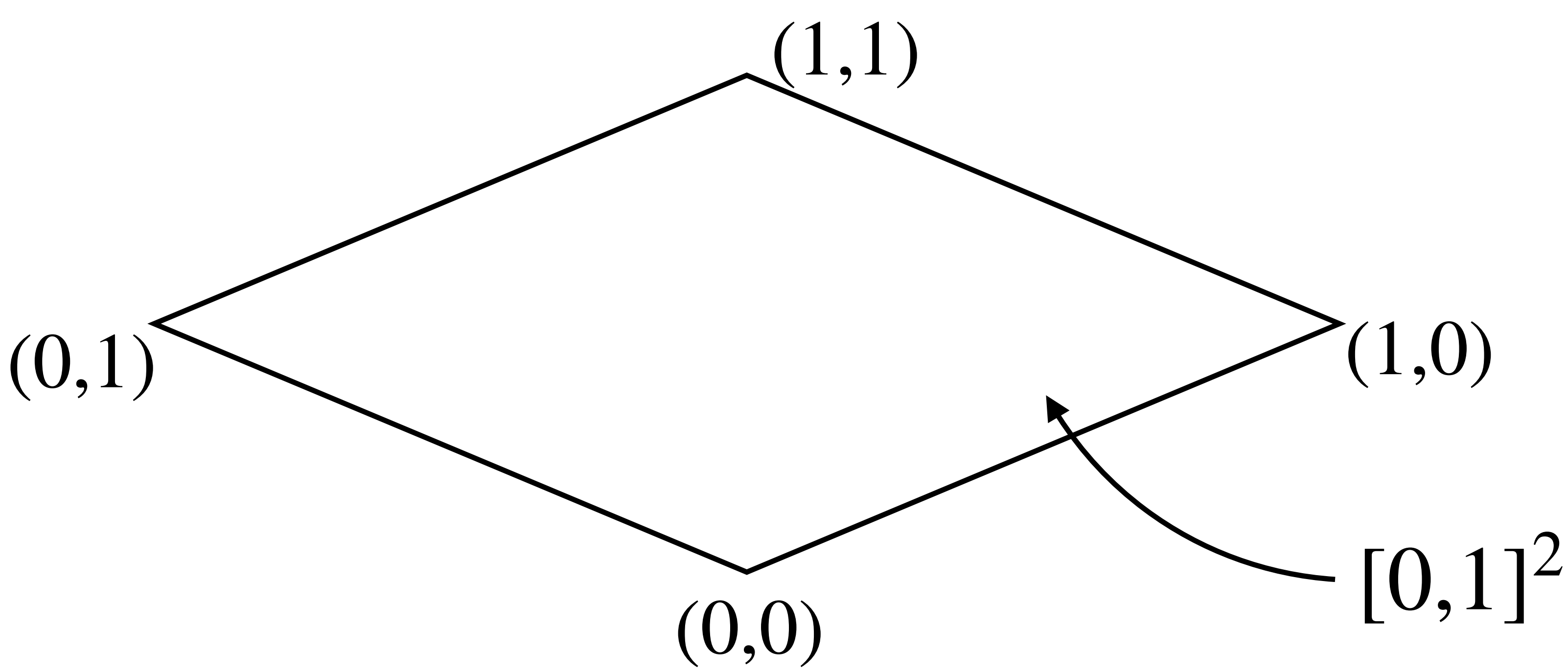
$n = 2$

$F(\emptyset) = 0$

$F(\{1\}) = 10$

$F(\{2\}) = 6$

$F(\{1,2\}) = 3$



Discrete Optimization

Set function: $F : 2^{[n]} \rightarrow \mathbb{R}$

Continuous Optimization

Lovász extension: $f : [0,1]^n \rightarrow \mathbb{R}$

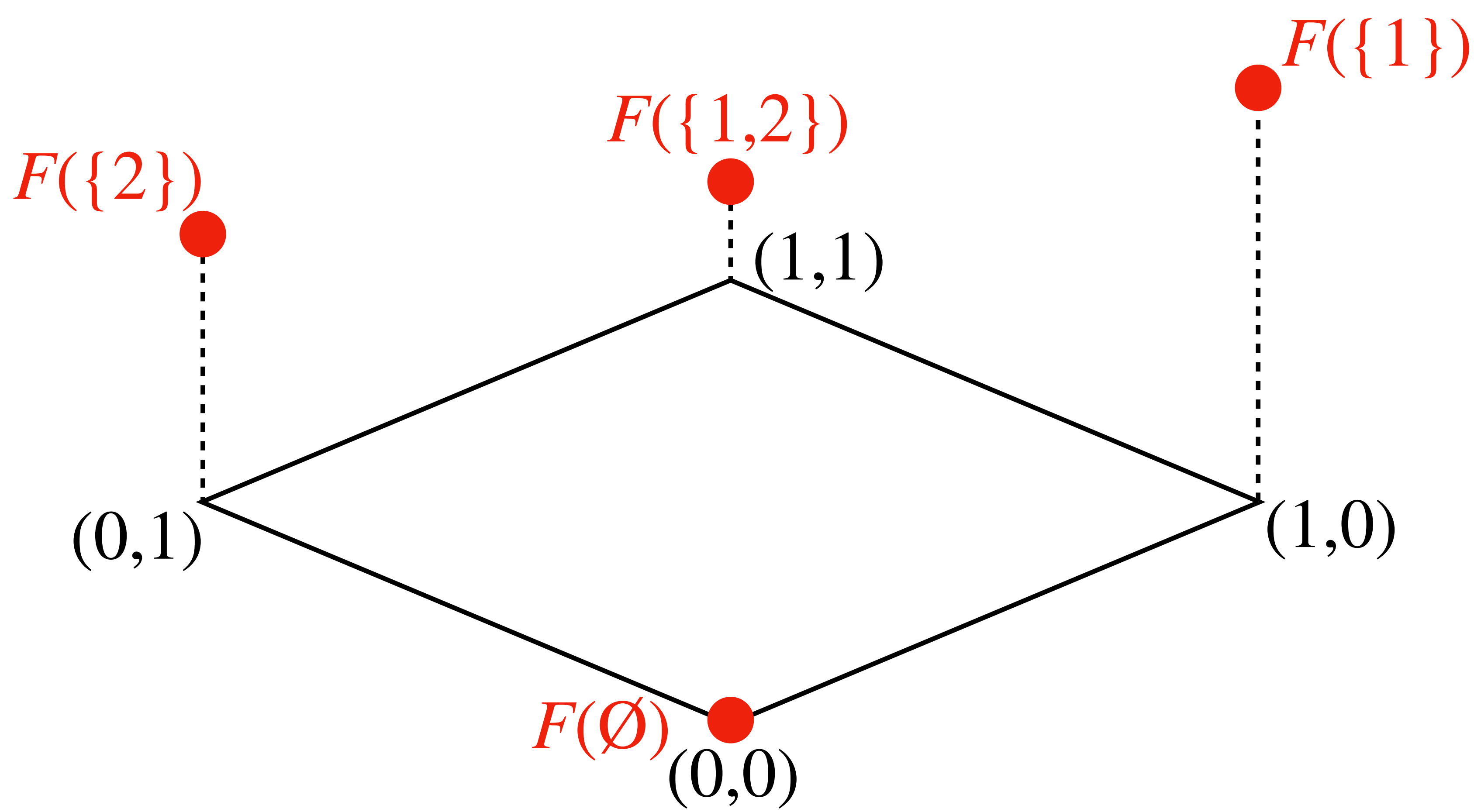
$n = 2$

$F(\emptyset) = 0$

$F(\{1\}) = 10$

$F(\{2\}) = 6$

$F(\{1,2\}) = 3$



Lovász Extension

Discrete Optimization

Set function: $F : 2^{[n]} \to \mathbb{R}$

Continuous Optimization

Lovász extension: $f : [0,1]^n \to \mathbb{R}$

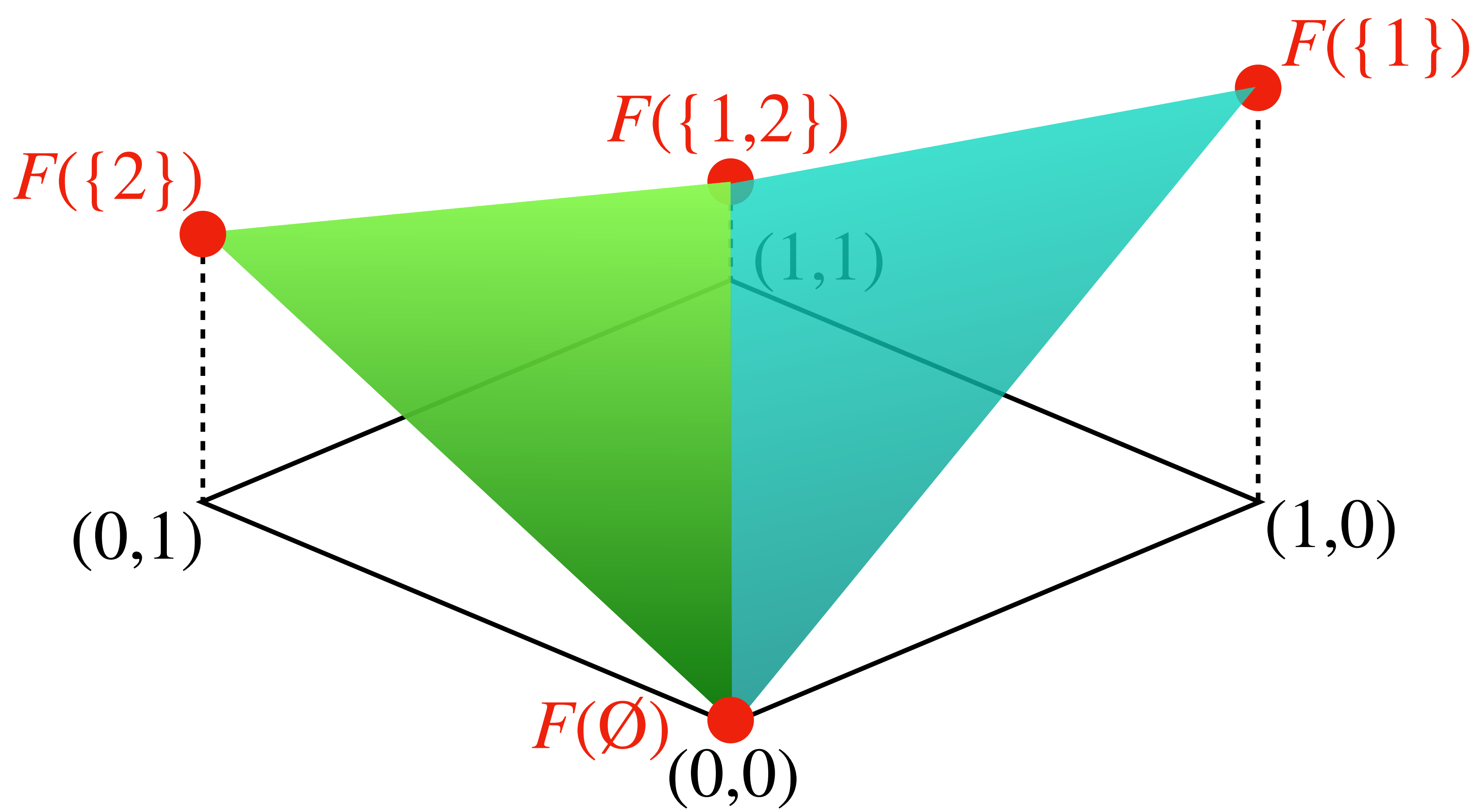
$n = 2$

$F(\emptyset) = 0$

$F(\{1\}) = 10$

$F(\{2\}) = 6$

$F(\{1,2\}) = 3$



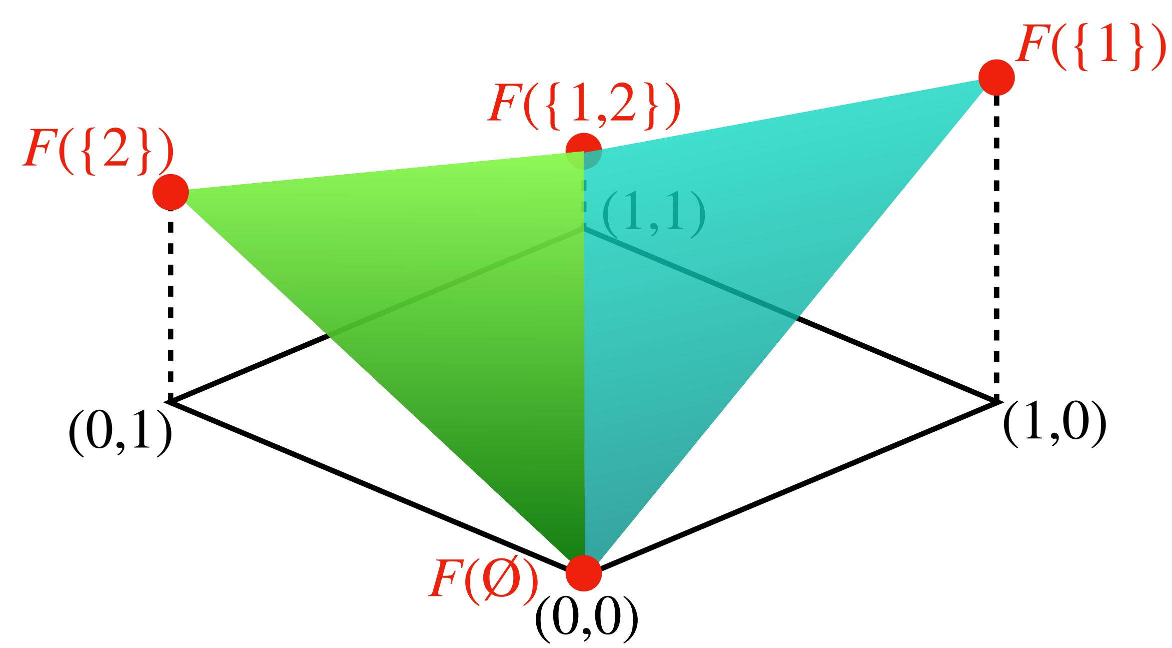
Discrete Optimization

Set function: $F : 2^{[n]} \to \mathbb{R}$

Continuous Optimization

Lovász extension: $f : [0,1]^n \to \mathbb{R}$

The Lovász extension is:



Lovász Extension

Discrete Optimization

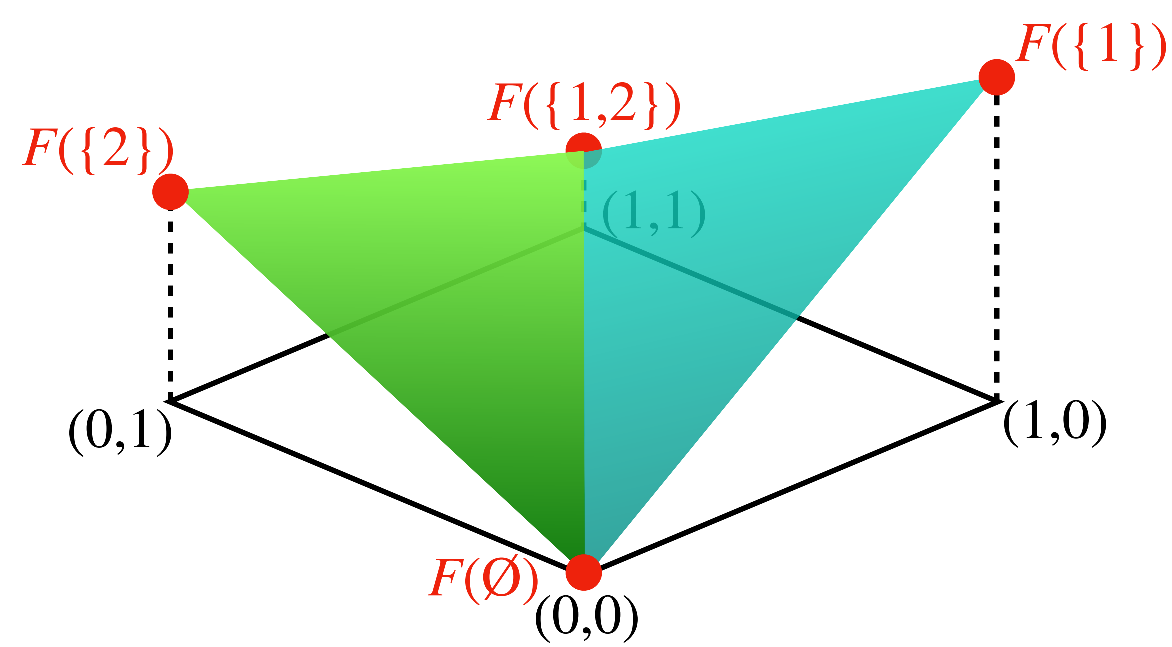
Set function: $F : 2^{[n]} \to \mathbb{R}$

Continuous Optimization

Lovász extension: $f : [0,1]^n \to \mathbb{R}$

The Lovász extension is:

- Piecewise linear



Lovász Extension

Discrete Optimization

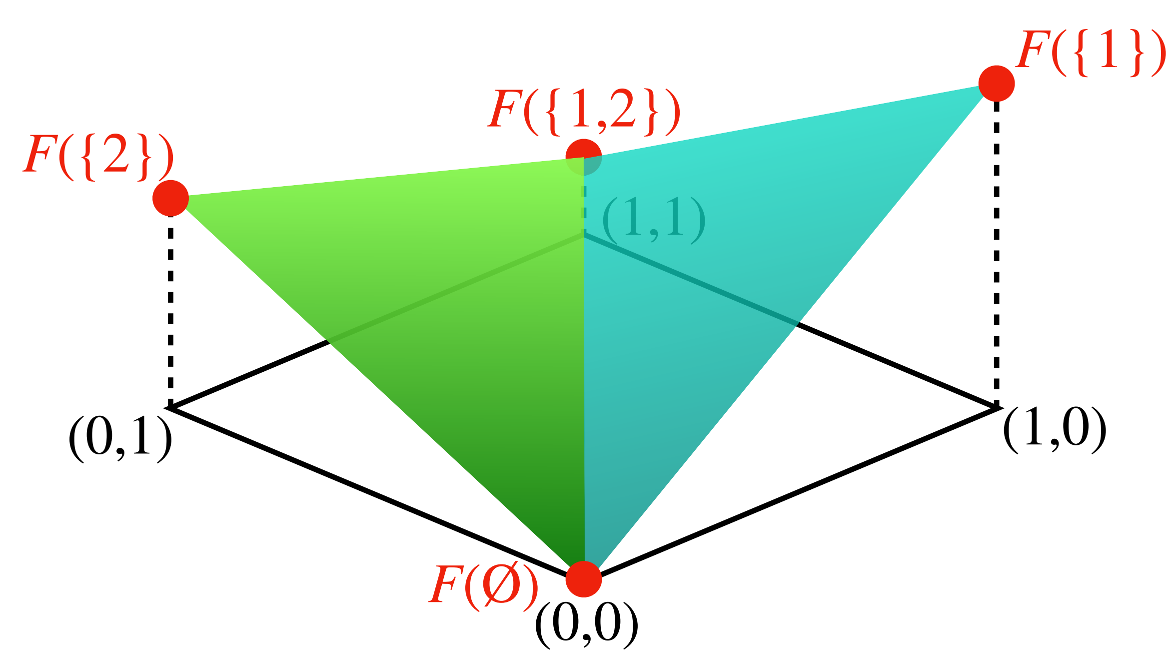
Set function: $F : 2^{[n]} \to \mathbb{R}$

Continuous Optimization

Lovász extension: $f : [0,1]^n \to \mathbb{R}$

The Lovász extension is:

- Piecewise linear
- **Convex** iff F is submodular (Lovász 1983)



Discrete Optimization

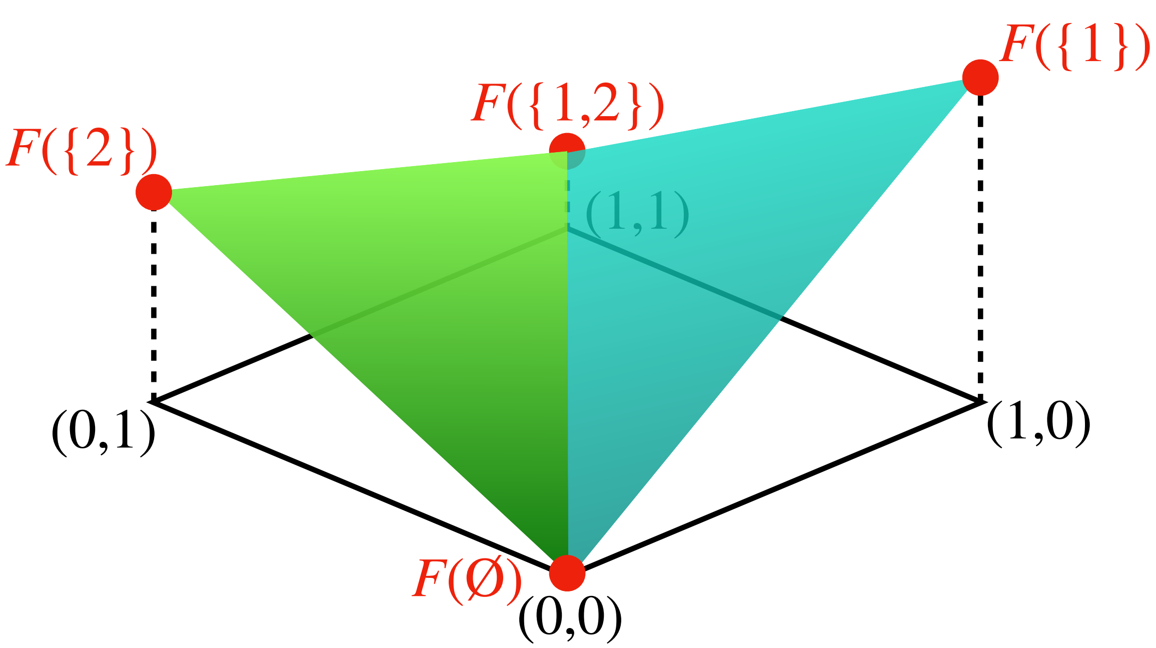
Set function: $F : 2^{[n]} \to \mathbb{R}$

Continuous Optimization

Lovász extension: $f : [0,1]^n \to \mathbb{R}$

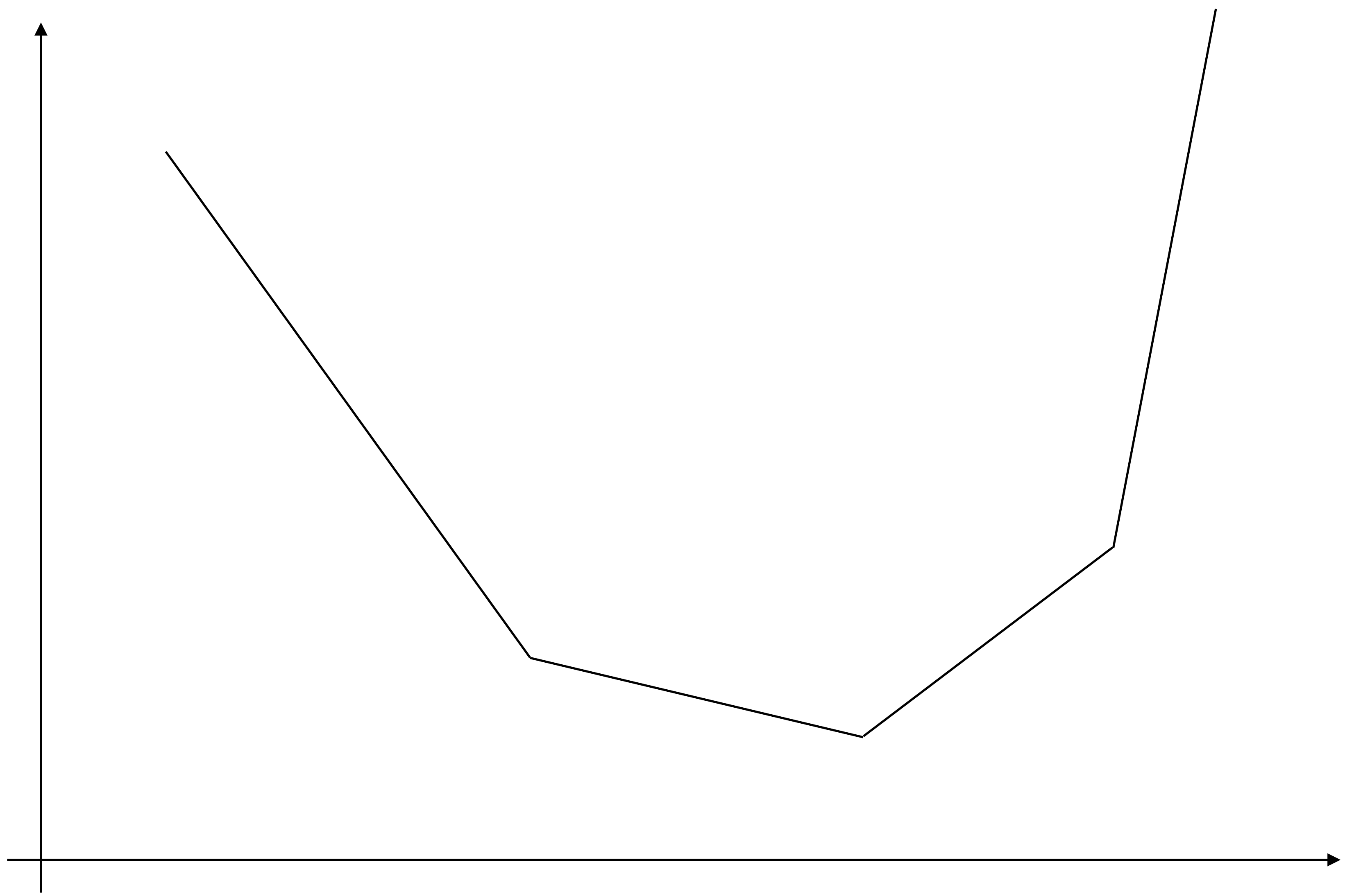
The Lovász extension is:

- Piecewise linear
- **Convex** iff F is submodular (Lovász 1983)
- Evaluable using n queries to F.



Stochastic Subgradient Descent

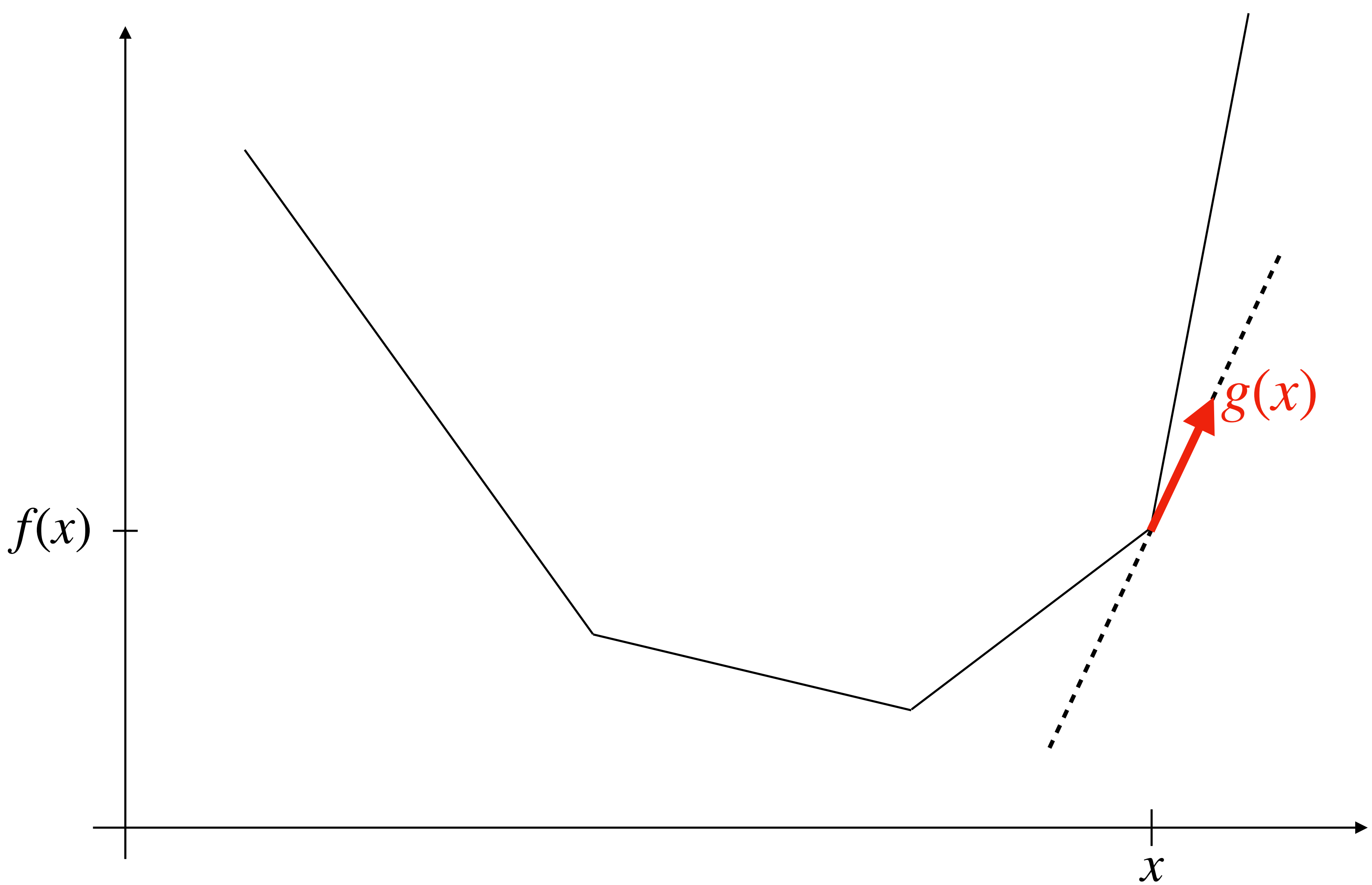
Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*



Stochastic Subgradient Descent

Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

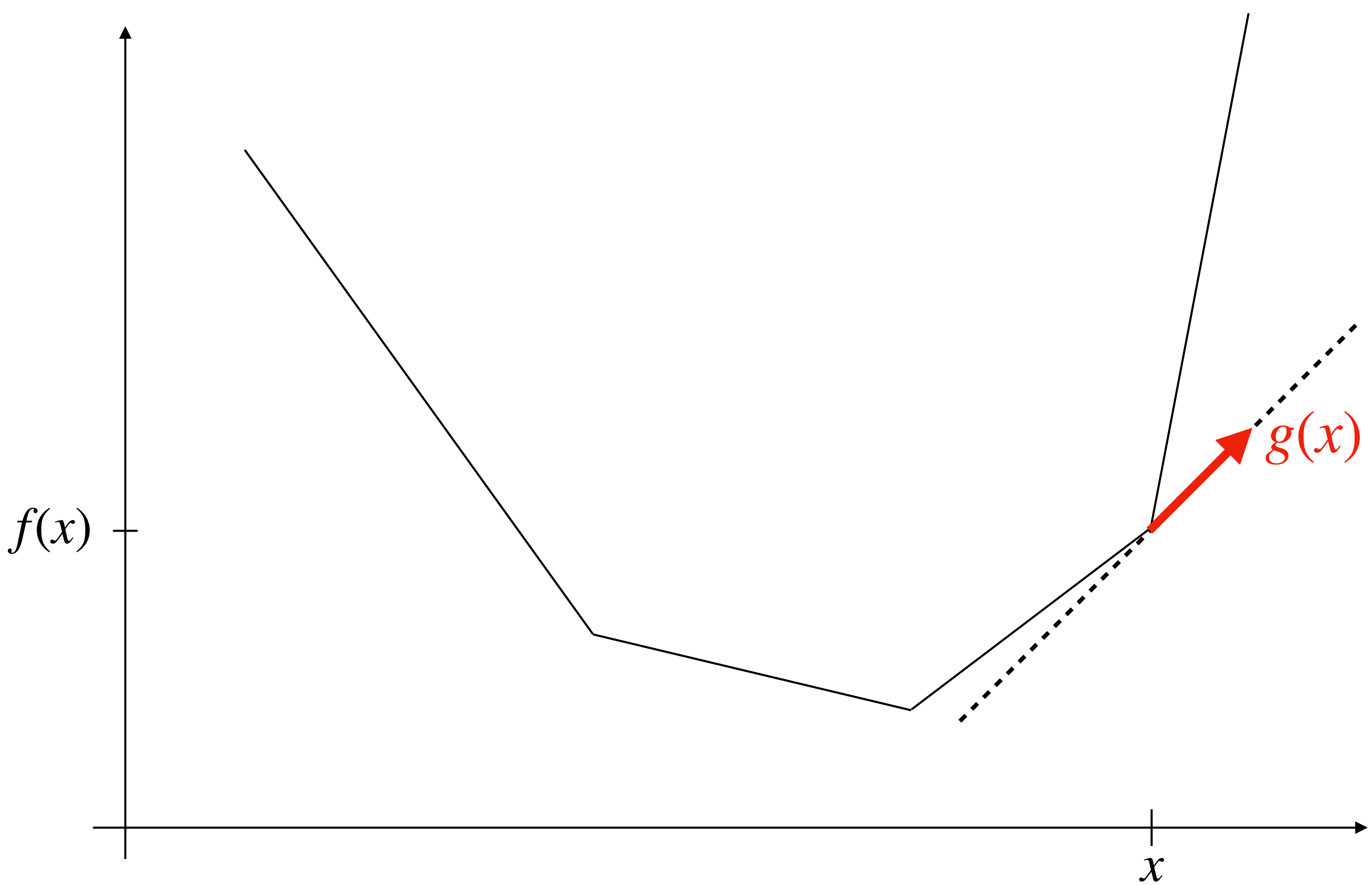
Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.



Stochastic Subgradient Descent

Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.

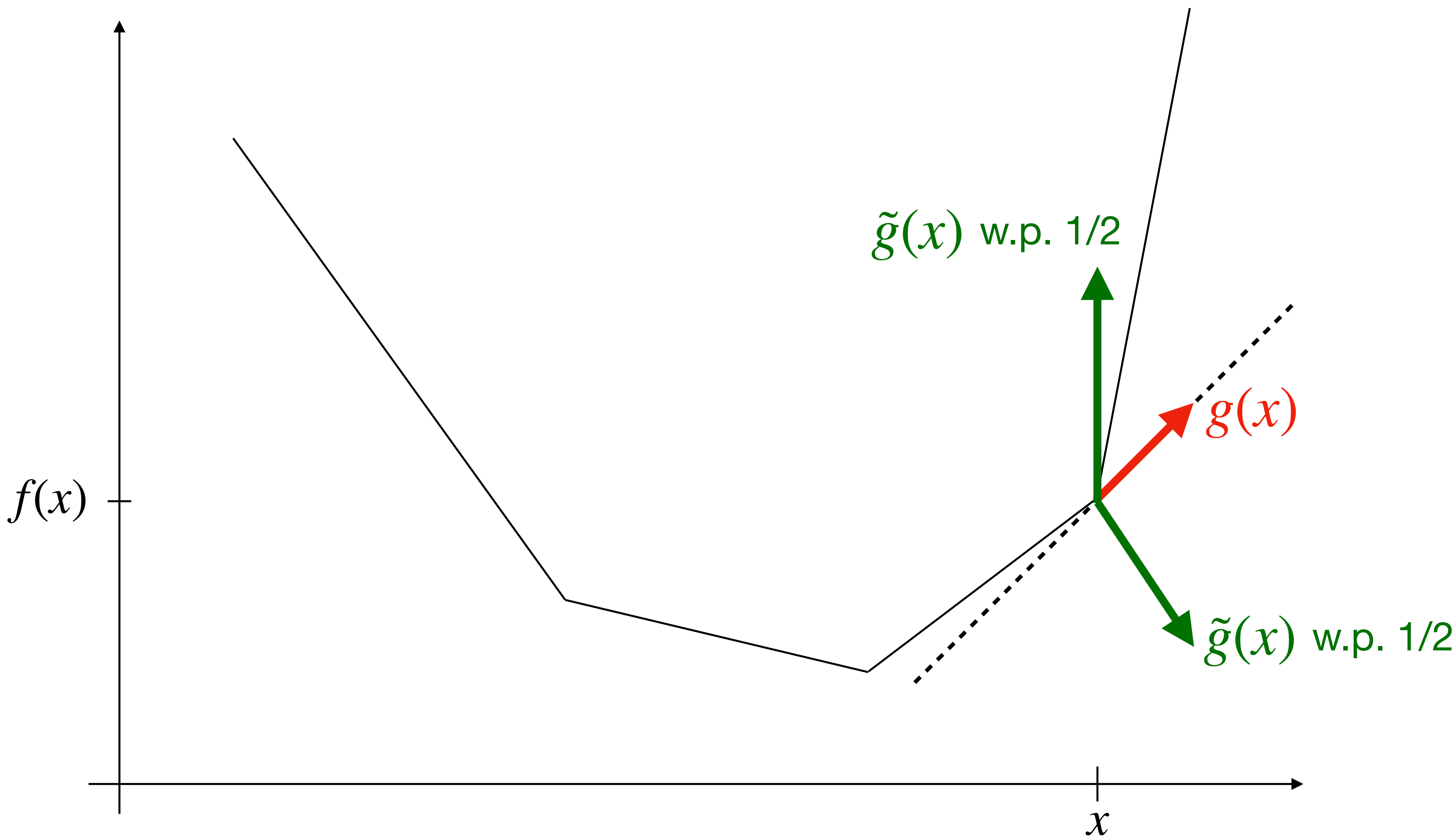


Stochastic Subgradient Descent

Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.

Stochastic Subgradient at x: random variable $\tilde{g}(x)$ satisfying $E[\tilde{g}(x)] = g(x)$



Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.

Stochastic Subgradient at x: random variable $\tilde{g}(x)$ satisfying $E[\tilde{g}(x)] = g(x)$

(projected) **Stochastic Subgradient Descent**

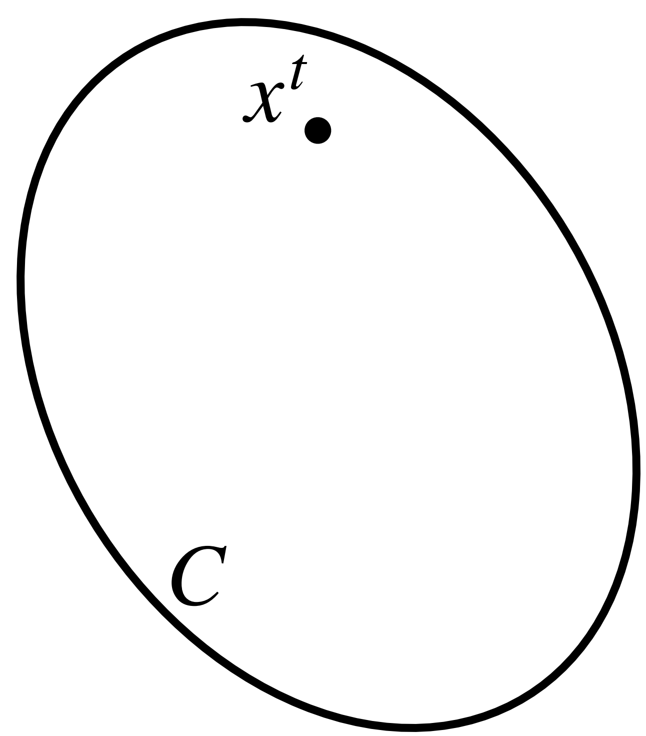
Stochastic Subgradient Descent

Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.

Stochastic Subgradient at x: random variable $\tilde{g}(x)$ satisfying $E[\tilde{g}(x)] = g(x)$

(projected) **Stochastic Subgradient Descent**



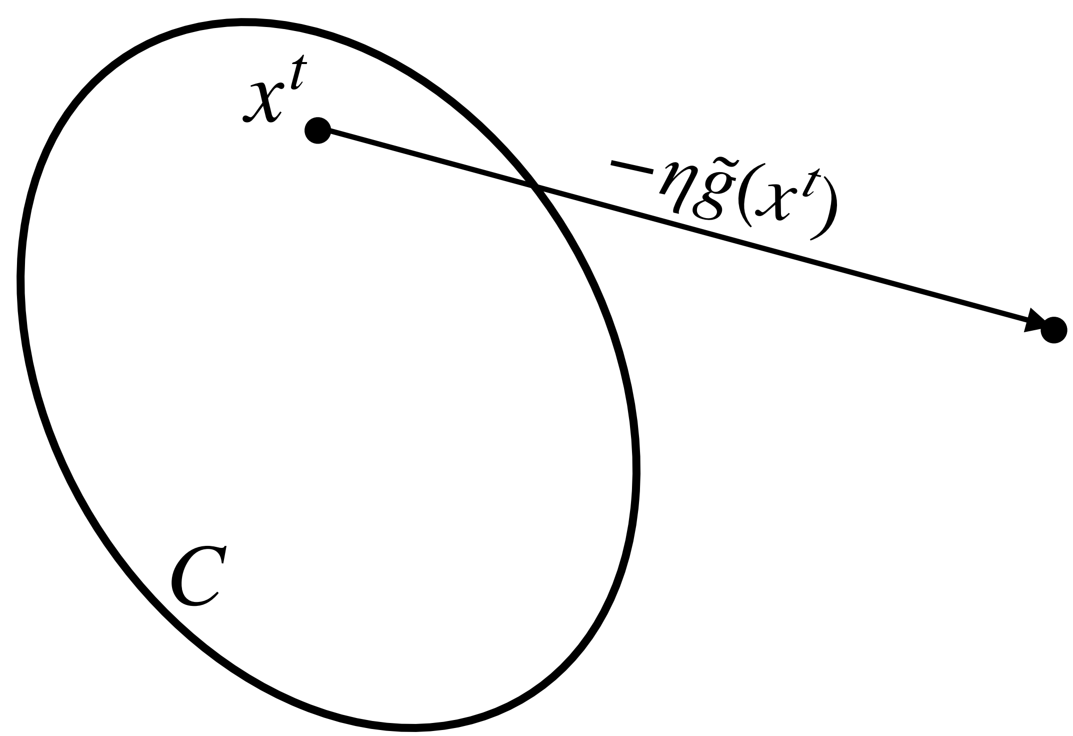
Stochastic Subgradient Descent

Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.

Stochastic Subgradient at x: random variable $\tilde{g}(x)$ satisfying $E[\tilde{g}(x)] = g(x)$

(projected) **Stochastic Subgradient Descent**



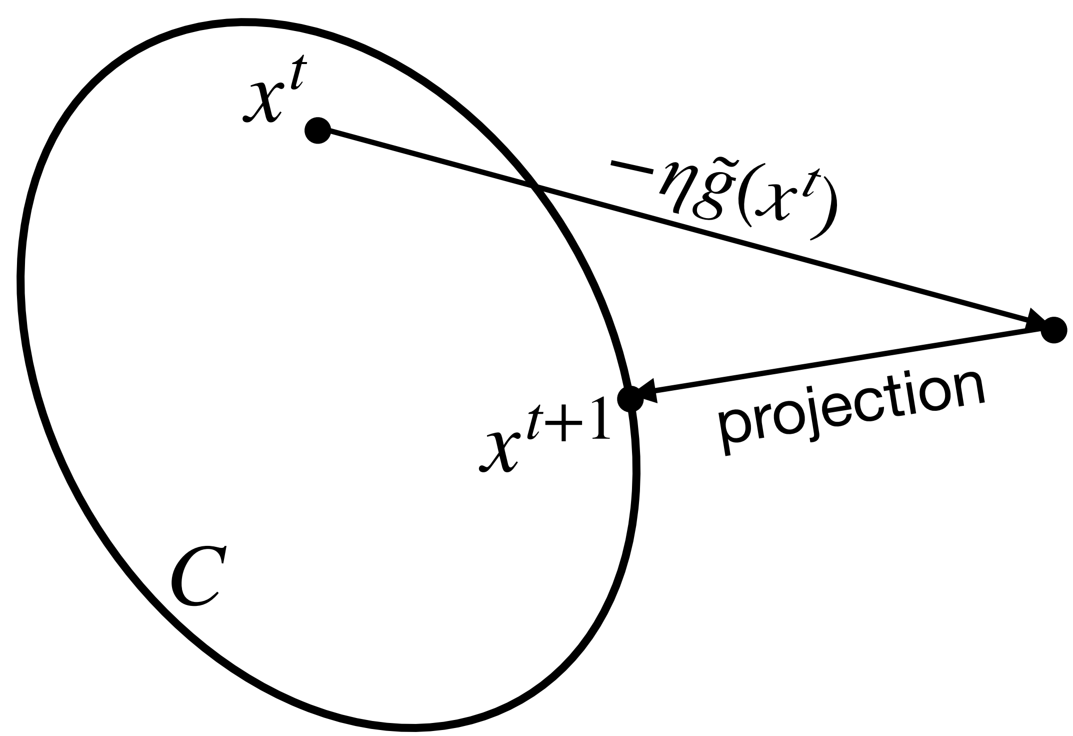
Stochastic Subgradient Descent

Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.

Stochastic Subgradient at x: random variable $\tilde{g}(x)$ satisfying $E[\tilde{g}(x)] = g(x)$

(projected) **Stochastic Subgradient Descent**



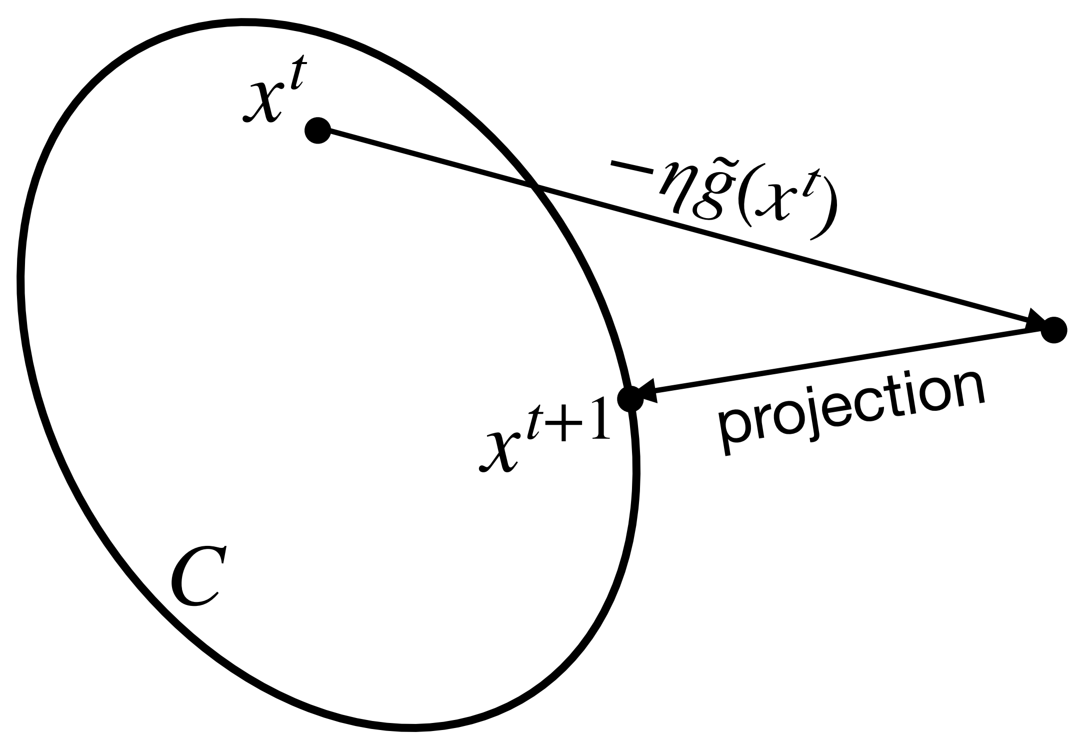
Stochastic Subgradient Descent

Convex function $f : C \to \mathbb{R}$ on a convex set C. *(not necessarily differentiable)*

Subgradient at x: slope $g(x)$ of any line that is below the graph of f and intersects it at x.

Stochastic Subgradient at x: random variable $\tilde{g}(x)$ satisfying $E[\tilde{g}(x)] = g(x)$

(projected) **Stochastic Subgradient Descent**



If $\tilde{g}(x)$ has **low variance** then the number of steps is the same as if we were using $g(x)$.

Stochastic Subgradient for the Lovász extension

For the Lovász extension f, there exists a subgradient $g(x)$ such that:

Stochastic Subgradient for the Lovász extension

For the Lovász extension f, there exists a subgradient $g(x)$ such that:

- $g(x)$ can be computed in time $O(n)$

(Jegelka, Bilmes 2011) and (Hazan, Kale 2012)

Stochastic Subgradient for the Lovász extension

For the Lovász extension f, there exists a subgradient $g(x)$ such that:

- $g(x)$ can be computed in time $O(n)$
- subgradient descent requires $O(n/\epsilon^2)$ steps to get an ε-minimizer of f

(Jegelka, Bilmes 2011) and (Hazan, Kale 2012)

Stochastic Subgradient for the Lovász extension

For the Lovász extension f, there exists a subgradient $g(x)$ such that:

- $g(x)$ can be computed in time $O(n)$
- subgradient descent requires $O(n/\epsilon^2)$ steps to get an ε-minimizer of f

(Jegelka, Bilmes 2011) and (Hazan, Kale 2012)

→ **Approximate minimization in time** $O(n \cdot n/\epsilon^2)$

Stochastic Subgradient for the Lovász extension

For the Lovász extension f, there exists a subgradient $g(x)$ such that:

- $g(x)$ can be computed in time $O(n)$
- subgradient descent requires $O(n/\epsilon^2)$ steps to get an ε-minimizer of f

(Jegelka, Bilmes 2011) and (Hazan, Kale 2012)

→ **Approximate minimization in time** $O(n \cdot n/\epsilon^2)$

A **stochastic subgradient** $\tilde{g}(x)$ can be computed in time $Q =$

- **Previous work:** $\tilde{O}(n^{2/3})$

(Chakrabarty, Lee, Sidford, Wong STOC'17)

Stochastic Subgradient for the Lovász extension

For the Lovász extension f, there exists a subgradient $g(x)$ such that:

- $g(x)$ can be computed in time $O(n)$
- subgradient descent requires $O(n/\epsilon^2)$ steps to get an ε-minimizer of f

(Jegelka, Bilmes 2011) and (Hazan, Kale 2012)

→ **Approximate minimization in time** $O(n \cdot n/\epsilon^2)$

A **stochastic subgradient** $\tilde{g}(x)$ can be computed in time **Q =**

- **Previous work:** $\tilde{O}(n^{2/3})$
  (Chakrabarty, Lee, Sidford, Wong STOC'17)

- **Our result:** $\tilde{O}(n^{1/2})$ (classical) or $\tilde{O}(n^{1/4}/\epsilon^{1/2})$ (quantum)

Stochastic Subgradient for the Lovász extension

For the Lovász extension f, there exists a subgradient $g(x)$ such that:

- $g(x)$ can be computed in time $O(n)$
- subgradient descent requires $O(n/\epsilon^2)$ steps to get an ε-minimizer of f

(Jegelka, Bilmes 2011) and (Hazan, Kale 2012)

→ **Approximate minimization in time** $O(n \cdot n/\epsilon^2)$

A **stochastic subgradient** $\tilde{g}(x)$ can be computed in time $Q =$

- **Previous work:** $\tilde{O}(n^{2/3})$

(Chakrabarty, Lee, Sidford, Wong STOC'17)

- **Our result:** $\tilde{O}(n^{1/2})$ (classical) or $\tilde{O}(n^{1/4}/\epsilon^{1/2})$ (quantum)

→ **Approximate minimization in time** $O(Q \cdot n/\epsilon^2)$

Stochastic Subgradient for the Lovász extension

One central idea in the construction of $\tilde{g}(x)$:

Importance Sampling according to $g(x)$.

Stochastic Subgradient for the Lovász extension

One central idea in the construction of $\tilde{g}(x)$:

Importance Sampling according to $g(x)$.

||

Sampling from the distribution that gives

i $\in$ **[n]** with probability $p_i = \frac{|g(x)_i|}{\|g(x)\|_1}$

Stochastic Subgradient for the Lovász extension

One central idea in the construction of $\tilde{g}(x)$:

Importance Sampling according to $g(x)$.

||

Sampling from the distribution that gives

i $\in$ **[n]** with probability $p_i = \dfrac{|g(x)_i|}{\|g(x)\|_1}$

This is where quantum computing comes in!

2

Quantum speed-up for Importance Sampling

Problem

Input: discrete probability distribution $\mathbf{D = (p_1,\ldots,p_n)}$ on [n].

Output: **T** independent samples $i_1,\ldots,i_T \sim D$.

Input: discrete probability distribution $\mathbf{D = (p_1,\dots,p_n)}$ on [n].

Output: **T** independent samples $i_1,\dots,i_T \sim D$.

Evaluation oracle access

Classical

$$i \mapsto p_i$$

Quantum

$$U(|i\rangle|0\rangle) = |i\rangle|p_i\rangle$$

Cost = # queries to the evaluation oracle

Input: discrete probability distribution **$D = (p_1, \ldots, p_n)$** on [n].

Output: **T** independent samples $i_1, \ldots, i_T \sim D$.

<u>Evaluation oracle access</u>

Classical

$$i \mapsto p_i$$

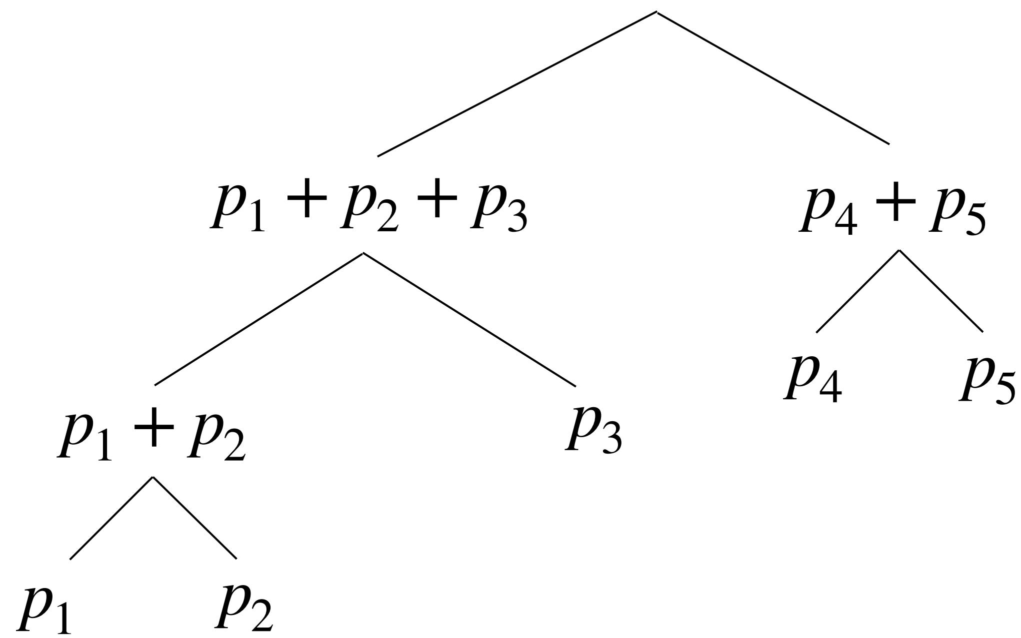
Quantum

$$U(|i\rangle|0\rangle) = |i\rangle|p_i\rangle$$

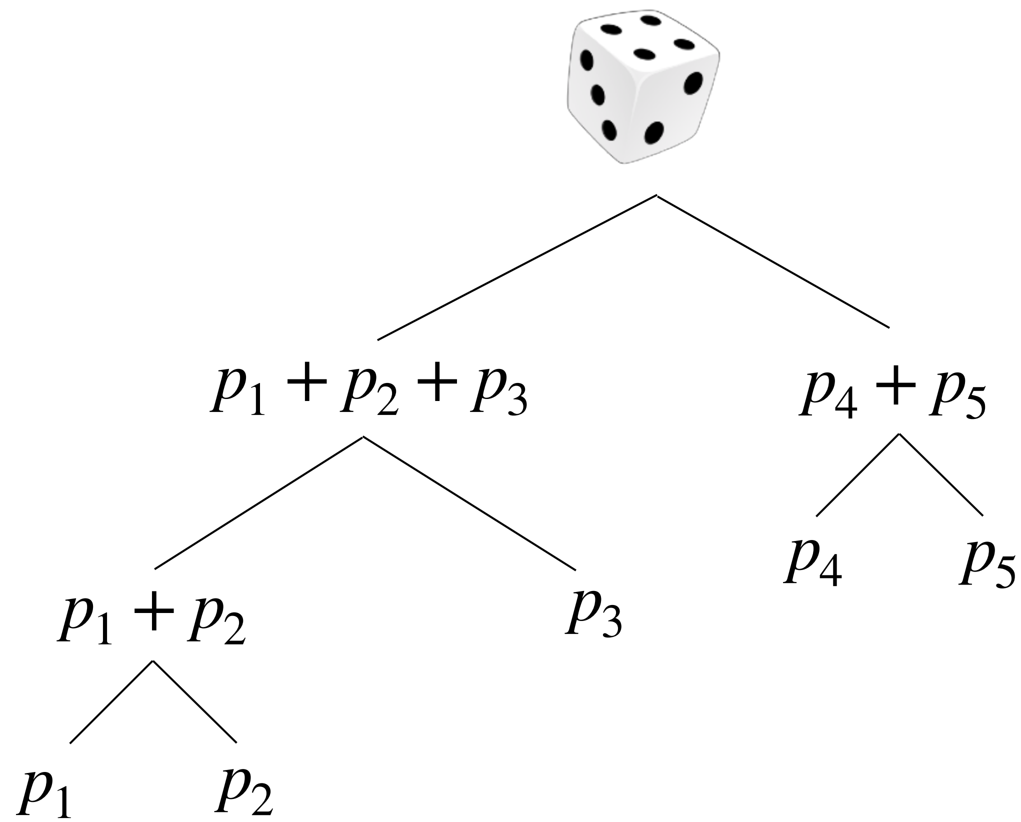
Cost = # queries to the evaluation oracle

Can quantum computing help to sample faster?

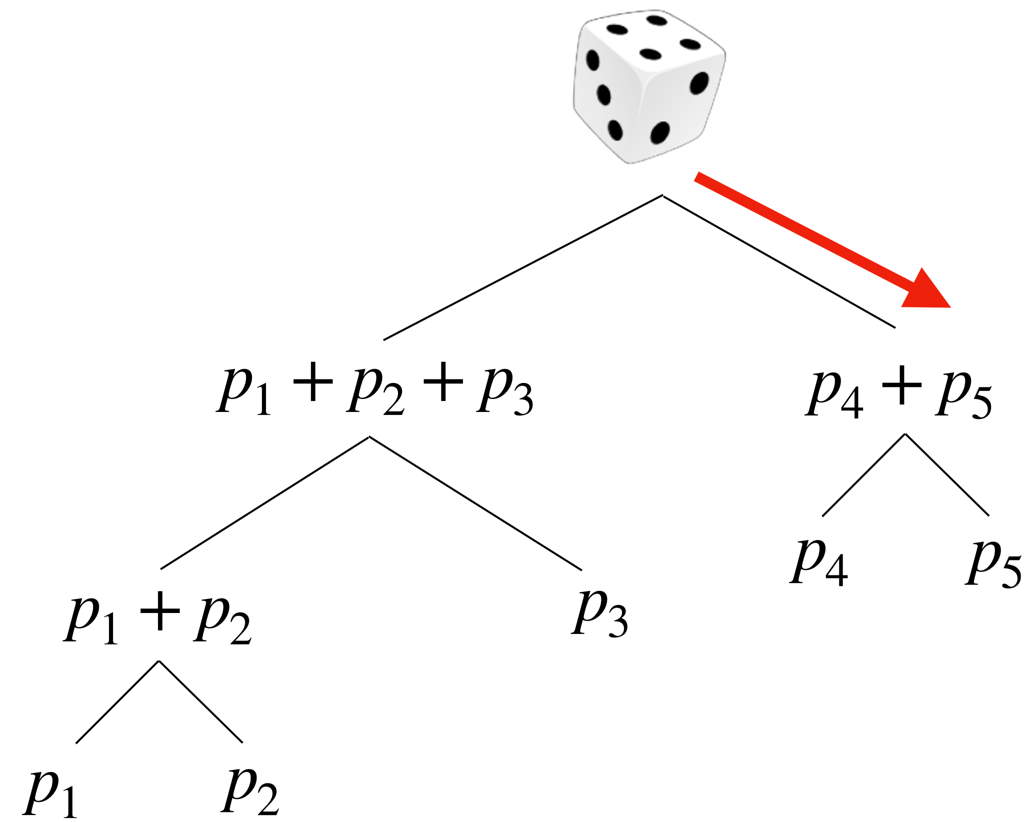
Importance Sampling with a Binary Tree



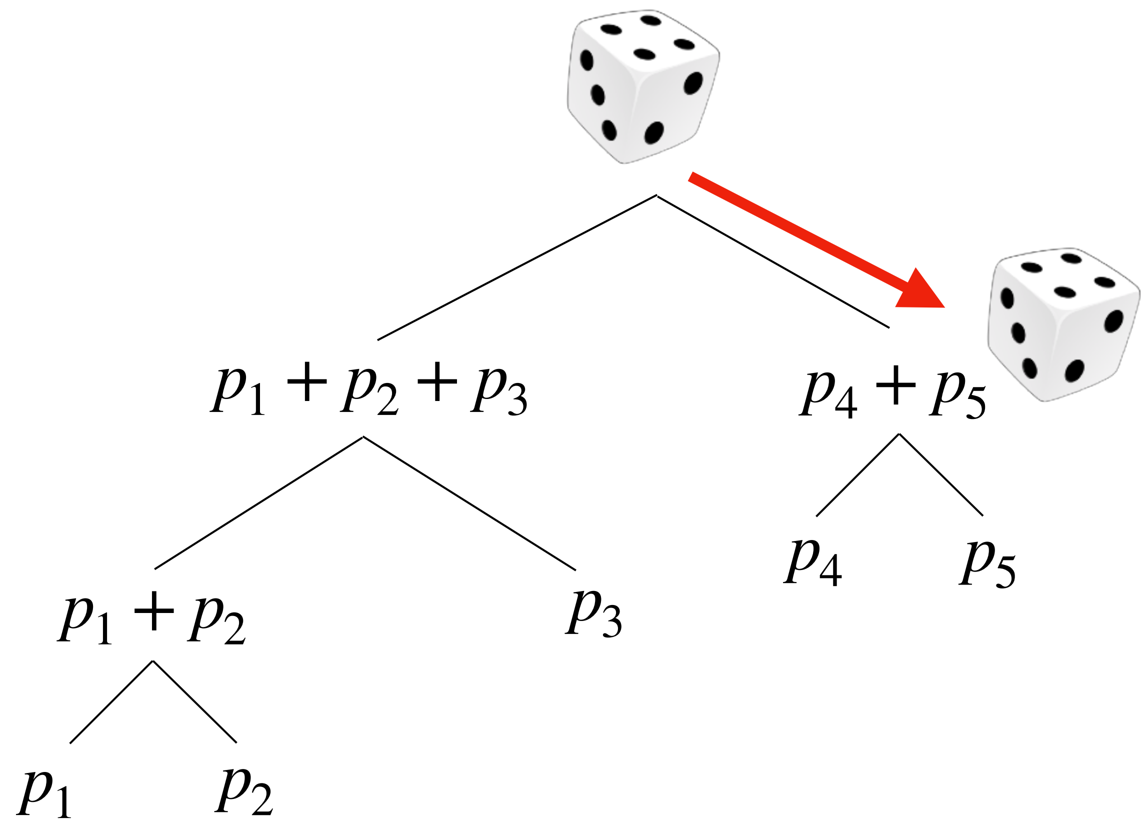
Importance Sampling with a Binary Tree



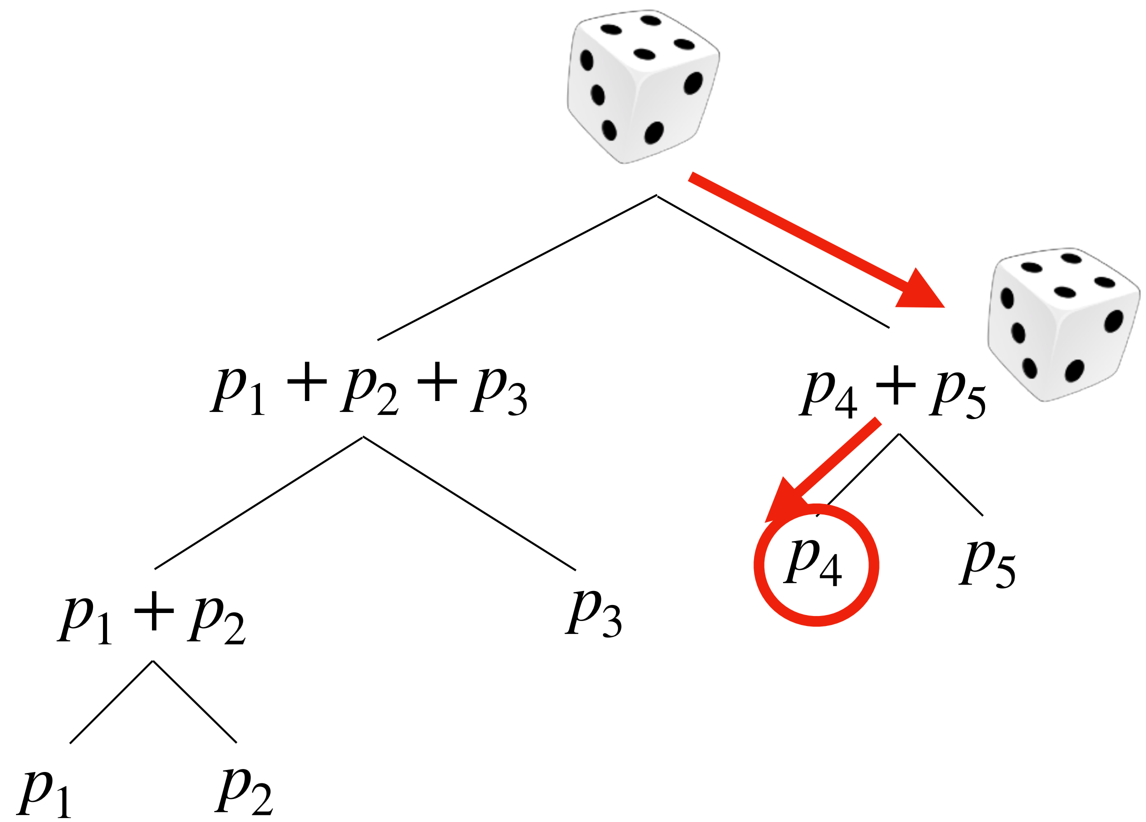
Importance Sampling with a Binary Tree



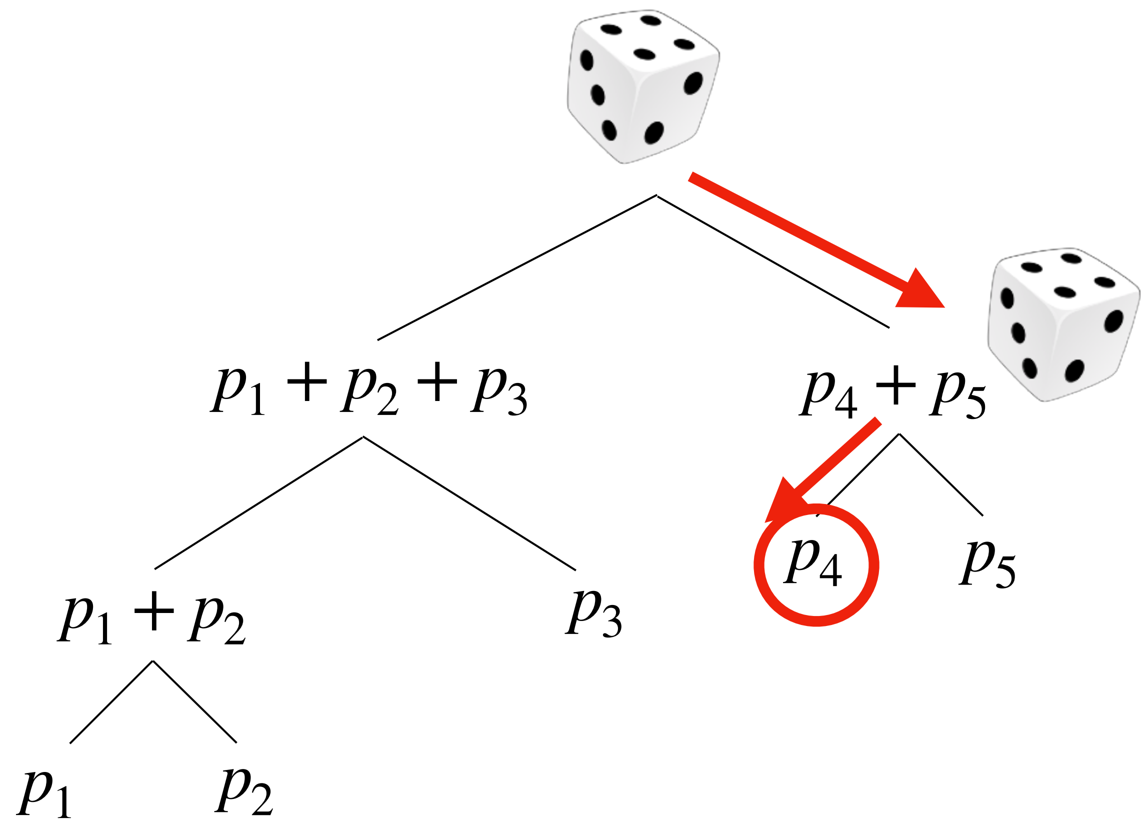
Importance Sampling with a Binary Tree



Importance Sampling with a Binary Tree



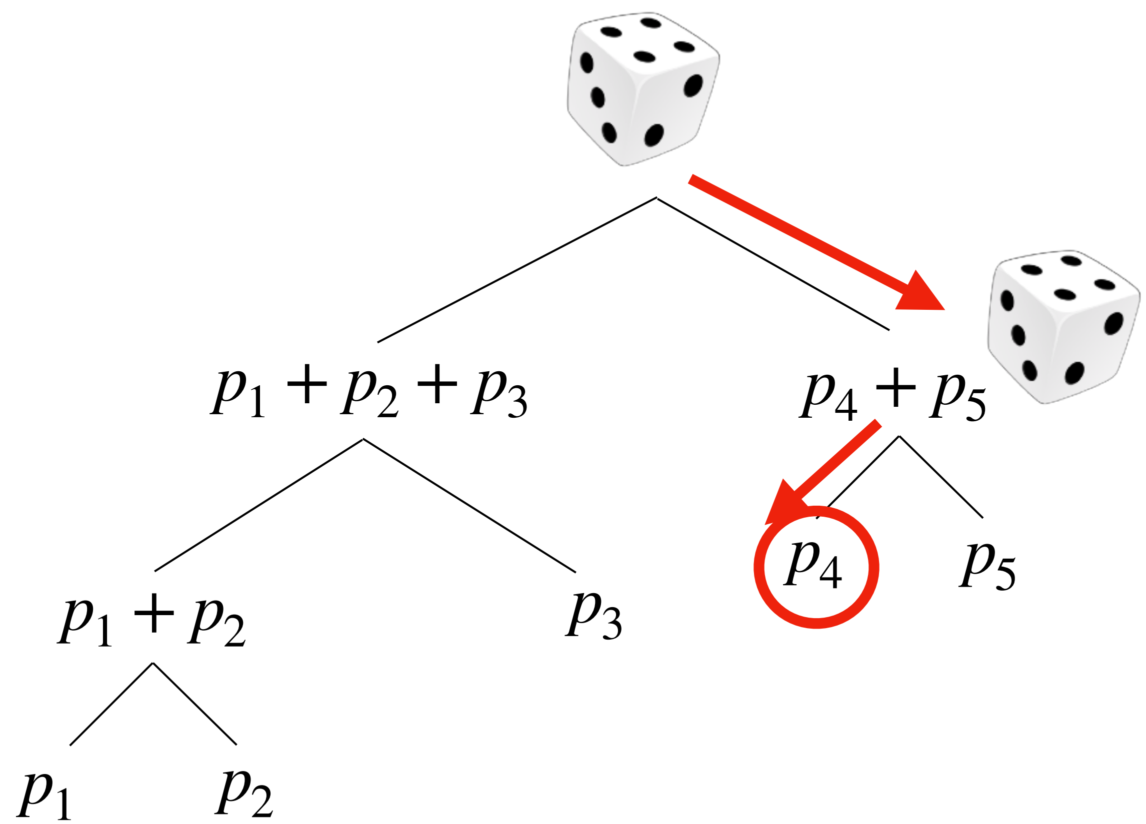
Importance Sampling with a Binary Tree



Preprocessing time: $O(n)$

Cost per sample: $O(\log n)$

Importance Sampling with a Binary Tree



Preprocessing time: $O(n)$

Cost per sample: $O(\log n)$

Cost for T samples: $O(n + T \log n)$

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

Sampling **(repeat T times):**

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

Sampling **(repeat T times)**:

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

2. Construct the unitary $V(|0\rangle|0\rangle) \longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$

Sampling **(repeat T times):**

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

2. Construct the unitary $V(|0\rangle|0\rangle) \longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$

$$\longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle \left(\sqrt{\frac{p_i}{p_{\max}}}|0\rangle + \sqrt{1 - \frac{p_i}{p_{\max}}}|1\rangle \right)$$

Sampling (repeat T times):

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

2. Construct the unitary $V(|0\rangle|0\rangle) \longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$

$$\longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle \left(\sqrt{\frac{p_i}{p_{\max}}} |0\rangle + \sqrt{1 - \frac{p_i}{p_{\max}}} |1\rangle \right)$$

$$= \frac{1}{\sqrt{n p_{\max}}} \left(\sum_i \sqrt{p_i} |i\rangle \right) |0\rangle + \ldots |1\rangle$$

Sampling **(repeat T times):**

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

2. Construct the unitary $V(|0\rangle|0\rangle) \longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$

$$\longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle \left(\sqrt{\frac{p_i}{p_{\max}}} |0\rangle + \sqrt{1 - \frac{p_i}{p_{\max}}} |1\rangle \right)$$

$$= \frac{1}{\sqrt{np_{\max}}} \left(\sum_i \sqrt{p_i} |i\rangle \right) |0\rangle + \ldots |1\rangle$$

Sampling **(repeat T times):**

1. Prepare $\sum_i \sqrt{p_i} |i\rangle$ with **Amplitude Amplification** on V, and measure it.

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

2. Construct the unitary $V(|0\rangle|0\rangle) \longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$

$$\longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle \left(\sqrt{\frac{p_i}{p_{\max}}} |0\rangle + \sqrt{1 - \frac{p_i}{p_{\max}}} |1\rangle \right)$$

$$= \frac{1}{\sqrt{n p_{\max}}} \left(\sum_i \sqrt{p_i} |i\rangle \right) |0\rangle + \ldots |1\rangle$$

Sampling (repeat T times):

1. Prepare $\sum_i \sqrt{p_i} |i\rangle$ with **Amplitude Amplification** on V, and measure it.

Preprocessing time: $O(\sqrt{n})$

Cost per sample: $O(\sqrt{n p_{\max}})$

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

2. Construct the unitary $V(|0\rangle|0\rangle) \longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$

$$\longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle \left(\sqrt{\frac{p_i}{p_{\max}}} |0\rangle + \sqrt{1 - \frac{p_i}{p_{\max}}} |1\rangle \right)$$

$$= \frac{1}{\sqrt{n p_{\max}}} \left(\sum_i \sqrt{p_i} |i\rangle \right) |0\rangle + \ldots |1\rangle$$

Sampling **(repeat T times):**

1. Prepare $\sum_i \sqrt{p_i} |i\rangle$ with **Amplitude Amplification** on V, and measure it.

Preprocessing time: $O(\sqrt{n})$

Cost per sample: $O(\sqrt{n p_{\max}})$

Cost for T samples: $O(\sqrt{n} + T\sqrt{n p_{\max}})$

Importance Sampling with Quantum State preparation

(Grover 2000)

Preprocessing:

1. Compute $p_{\max} = \max\{p_1, \ldots, p_n\}$ with quantum **Maximum Finding**

2. Construct the unitary $V(|0\rangle|0\rangle) \longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$

$$\longmapsto \frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle \left(\sqrt{\frac{p_i}{p_{\max}}} |0\rangle + \sqrt{1 - \frac{p_i}{p_{\max}}} |1\rangle \right)$$

$$= \frac{1}{\sqrt{n p_{\max}}} \left(\sum_i \sqrt{p_i} |i\rangle \right) |0\rangle + \ldots |1\rangle$$

Sampling **(repeat T times)**:

1. Prepare $\sum_i \sqrt{p_i}|i\rangle$ with **Amplitude Amplification** on V, and measure it.

Preprocessing time: $O(\sqrt{n})$

Cost per sample: $O(\sqrt{n p_{\max}})$

Cost for T samples: $O(\sqrt{n} + T\sqrt{n p_{\max}}) = O(T\sqrt{n})$

Binary Tree

$$O(n + T \log n)$$

Quantum State Preparation

$$O(T\sqrt{n})$$

Importance Sampling

Binary Tree

$$O(n + T\log n)$$

Quantum State Preparation

$$O(T\sqrt{n})$$

For our submodular function minimization algorithm, we need **$T = \sqrt{n}$.**

Importance Sampling

Binary Tree

$O(n + T \log n)$

Quantum State Preparation

$O(T\sqrt{n})$

For our submodular function minimization algorithm, we need **T = √n.**

↓

New quantum multi-sampling algorithm in $O\left(\sqrt{Tn}\right)$

Importance Sampling with a Quantum Oracle

Our result: $O\left(\sqrt{Tn}\right)$ for obtaining T independent samples from D = $(p_1,\ldots,p_n)$.

Importance Sampling with a Quantum Oracle

Our result: $O(\sqrt{Tn})$ for obtaining T independent samples from D = (p_1,…,p_n).

Element	1	2	3	4	5	6	7
Probability	p_1	p_2	p_3	p_4	p_5	p_6	p_7

Distribution D

Importance Sampling with a Quantum Oracle

Our result: $O(\sqrt{Tn})$ for obtaining T independent samples from D = (p_1,…,p_n).

Element	1	2	3	4	5	6	7
Probability	p_1	p_2	p_3	p_4	p_5	p_6	p_7

Distribution D

$p_i \geq 1/T$

$P_{\text{Heavy}} = \sum_{i \in \text{Heavy}} p_i$

Element	1	3	4
Probability	$\frac{p_1}{P_{\text{Heavy}}}$	$\frac{p_3}{P_{\text{Heavy}}}$	$\frac{p_4}{P_{\text{Heavy}}}$

Distribution D_{Heavy}

Importance Sampling with a Quantum Oracle

Our result: $O(\sqrt{Tn})$ for obtaining T independent samples from D = $(p_1,\ldots,p_n)$.

Element	1	2	3	4	5	6	7
Probability	p_1	p_2	p_3	p_4	p_5	p_6	p_7

Distribution D

$p_i \geq 1/T$

$P_{\text{Heavy}} = \sum_{i \in \text{Heavy}} p_i$

$p_i < 1/T$

$P_{\text{Light}} = \sum_{i \in \text{Light}} p_i$

Element	1	3	4
Probability	$\frac{p_1}{P_{\text{Heavy}}}$	$\frac{p_3}{P_{\text{Heavy}}}$	$\frac{p_4}{P_{\text{Heavy}}}$

Distribution D_{Heavy}

Element	2	5	6	7
Probability	$\frac{p_2}{P_{\text{Light}}}$	$\frac{p_5}{P_{\text{Light}}}$	$\frac{p_6}{P_{\text{Light}}}$	$\frac{p_7}{P_{\text{Light}}}$

Distribution D_{Light}

Importance Sampling with a Quantum Oracle

Our result: $O(\sqrt{Tn})$ for obtaining T independent samples from D = ($p_1,\ldots,p_n$).

Element	1	2	3	4	5	6	7
Probability	p_1	p_2	p_3	p_4	p_5	p_6	p_7

Distribution D

$p_i \geq 1/T$

$P_{\text{Heavy}} = \sum_{i \in \text{Heavy}} p_i$

$p_i < 1/T$

$P_{\text{Light}} = \sum_{i \in \text{Light}} p_i$

Element	1	3	4
Probability	$\frac{p_1}{P_{\text{Heavy}}}$	$\frac{p_3}{P_{\text{Heavy}}}$	$\frac{p_4}{P_{\text{Heavy}}}$

Distribution D_{Heavy}

Element	2	5	6	7
Probability	$\frac{p_2}{P_{\text{Light}}}$	$\frac{p_5}{P_{\text{Light}}}$	$\frac{p_6}{P_{\text{Light}}}$	$\frac{p_7}{P_{\text{Light}}}$

Distribution D_{Light}

Use a Binary Tree

Importance Sampling with a Quantum Oracle

Our result: $O(\sqrt{Tn})$ for obtaining T independent samples from D = (p_1,…,p_n).

Element	1	2	3	4	5	6	7
Probability	p_1	p_2	p_3	p_4	p_5	p_6	p_7

Distribution D

$p_i \geq 1/T$ $\quad P_{\text{Heavy}} = \sum_{i \in \text{Heavy}} p_i$

$p_i < 1/T$ $\quad P_{\text{Light}} = \sum_{i \in \text{Light}} p_i$

Element	1	3	4
Probability	$\frac{p_1}{P_{\text{Heavy}}}$	$\frac{p_3}{P_{\text{Heavy}}}$	$\frac{p_4}{P_{\text{Heavy}}}$

Distribution D_{Heavy}

Use a Binary Tree

Element	2	5	6	7
Probability	$\frac{p_2}{P_{\text{Light}}}$	$\frac{p_5}{P_{\text{Light}}}$	$\frac{p_6}{P_{\text{Light}}}$	$\frac{p_7}{P_{\text{Light}}}$

Distribution D_{Light}

Use Quantum State Preparation

Importance Sampling with a Quantum Oracle

Preprocessing:

Sampling **(repeat T times):**

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** $\subset [n]$ of indices i such that $\mathbf{p_i \geq 1/T}$, using **Grover Search**.

Sampling (repeat T times):

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that **$p_i \geq 1/T$**, using **Grover Search**.

2. Compute **P_{Heavy}** $= \sum_{i \in \text{Heavy}} p_i$

Sampling **(repeat T times):**

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that $p_i \geq 1/T$, using **Grover Search**.

2. Compute $P_{\text{Heavy}} = \sum_{i \in \text{Heavy}} p_i$

3. Apply the preprocessing step of the **Binary Tree Method** on D_{Heavy}.

Sampling **(repeat T times):**

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that $p_i \geq 1/T$, using **Grover Search**.

2. Compute $P_{Heavy} = \sum_{i \in \text{Heavy}} p_i$

3. Apply the preprocessing step of the **Binary Tree Method** on D_{Heavy}.

4. Apply the preprocessing step of the **Quant. State Preparation** method on D_{Light}.

Sampling **(repeat T times):**

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** $\subset [n]$ of indices i such that $p_i \geq 1/T$, using **Grover Search**.

2. Compute $P_{Heavy} = \sum_{i \in \text{Heavy}} p_i$

3. Apply the preprocessing step of the **Binary Tree Method** on D_{Heavy}.

4. Apply the preprocessing step of the **Quant. State Preparation** method on D_{Light}.

Sampling (repeat T times)**:**

Flip a coin that is head with probability P_{Heavy} :

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that $p_i \geq 1/T$, using **Grover Search**.

2. Compute $P_{Heavy} = \sum_{i \in \mathrm{Heavy}} p_i$

3. Apply the preprocessing step of the **Binary Tree Method** on D_{Heavy}.

4. Apply the preprocessing step of the **Quant. State Preparation** method on D_{Light}.

Sampling (repeat T times):

Flip a coin that is head with probability P_{Heavy} :

- **Head**: sample $i \sim D_{Heavy}$ with the **Binary Tree Method.**

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that $p_i \geq 1/T$, using **Grover Search**.
2. Compute $P_{Heavy} = \sum_{i \in \text{Heavy}} p_i$
3. Apply the preprocessing step of the **Binary Tree Method** on D_{Heavy}.
4. Apply the preprocessing step of the **Quant. State Preparation** method on D_{Light}.

Sampling (repeat T times):

Flip a coin that is head with probability P_{Heavy} :

- **Head**: sample $i \sim D_{Heavy}$ with the **Binary Tree Method.**
- **Tail**: sample $i \sim D_{Light}$ with **Quantum State Preparation.**

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** $\subset [n]$ of indices i such that $\mathbf{p_i \geq 1/T}$, using **Grover Search**.

   Cost:

2. Compute $\mathbf{P_{Heavy}} = \sum_{i \in \mathrm{Heavy}} p_i$

   Cost:

3. Apply the preprocessing step of the **Binary Tree Method** on $\mathbf{D_{Heavy}}$.

   Cost:

4. Apply the preprocessing step of the **Quant. State Preparation** method on $\mathbf{D_{Light}}$.

   Cost:

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that **$p_i \geq 1/T$**, using **Grover Search**.

   Cost: $O(\sqrt{nT})$ since **|Heavy| ≤ T**

2. Compute $\mathbf{P_{Heavy}} = \sum_{i \in \text{Heavy}} p_i$

   Cost:

3. Apply the preprocessing step of the **Binary Tree Method** on $\mathbf{D_{Heavy}}$.

   Cost:

4. Apply the preprocessing step of the **Quant. State Preparation** method on $\mathbf{D_{Light}}$.

   Cost:

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that **$p_i \geq 1/T$**, using **Grover Search**.

   Cost: $O(\sqrt{nT})$ since **|Heavy| ≤ T**

2. Compute **P_{Heavy}** $= \sum_{i \in \text{Heavy}} p_i$

   Cost: $O(T)$

3. Apply the preprocessing step of the **Binary Tree Method** on **D_{Heavy}**.

   Cost:

4. Apply the preprocessing step of the **Quant. State Preparation** method on **D_{Light}**.

   Cost:

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that **$p_i \geq 1/T$**, using **Grover Search**.

   Cost: $O(\sqrt{nT})$ since **|Heavy| ≤ T**

2. Compute $\mathbf{P_{Heavy}} = \sum_{i \in \text{Heavy}} p_i$

   Cost: $O(T)$

3. Apply the preprocessing step of the **Binary Tree Method** on $\mathbf{D_{Heavy}}$.

   Cost: $O(T)$

4. Apply the preprocessing step of the **Quant. State Preparation** method on $\mathbf{D_{Light}}$.

   Cost:

Importance Sampling with a Quantum Oracle

Preprocessing:

1. Compute the set **Heavy** ⊂ [n] of indices i such that **$p_i \geq 1/T$**, using **Grover Search**.

   Cost: $O(\sqrt{nT})$ since **|Heavy| ≤ T**

2. Compute $\mathbf{P_{Heavy}} = \sum_{i \in \mathrm{Heavy}} p_i$

   Cost: $O(T)$

3. Apply the preprocessing step of the **Binary Tree Method** on $\mathbf{D_{Heavy}}$.

   Cost: $O(T)$

4. Apply the preprocessing step of the **Quant. State Preparation** method on $\mathbf{D_{Light}}$.

   Cost: $O(\sqrt{n})$

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability P_{Heavy} :

- **Head**: sample $i \sim D_{Heavy}$ with the **Binary Tree Method.**

Cost per sample:

- **Tail**: sample $i \sim D_{Light}$ with **Quantum State Preparation.**

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability P_{Heavy} :

- **Head**: sample $i \sim D_{Heavy}$ with the **Binary Tree Method.**

Cost per sample: $O(\log n)$

- **Tail**: sample $i \sim D_{Light}$ with **Quantum State Preparation.**

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability **P_{Heavy}** :

- **Head**: sample **i ~ D_{Heavy}** with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample **i ~ D_{Light}** with **Quantum State Preparation.**

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability **P_{Heavy}** :

- **Head**: sample **i ~ D_{Heavy}** with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample **i ~ D_{Light}** with **Quantum State Preparation.**

Cost per sample:

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability **P_{Heavy}** :

- **Head**: sample **$i \sim D_{Heavy}$** with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample **$i \sim D_{Light}$** with **Quantum State Preparation.**

Cost per sample: $O\left(\sqrt{np_{\max}}\right)$ where $p_{\max} = \max\left\{\frac{p_i}{P_{\text{Light}}} : i \in \text{Light}\right\}$

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability **P_{Heavy}** :

- **Head**: sample **$i \sim D_{Heavy}$** with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample **$i \sim D_{Light}$** with **Quantum State Preparation.**

Cost per sample: $O\left(\sqrt{np_{\max}}\right)$ where $p_{\max} = \max\left\{\frac{p_i}{P_{\text{Light}}} : i \in \text{Light}\right\} \leq \frac{1}{T \cdot P_{\text{Light}}}$

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability P_{Heavy} :

- **Head**: sample **i ~ D_{Heavy}** with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample **i ~ D_{Light}** with **Quantum State Preparation.**

Cost per sample: $O\left(\sqrt{np_{\max}}\right)$ where $p_{\max} = \max\left\{\frac{p_i}{P_{\text{Light}}} : i \in \text{Light}\right\} \leq \frac{1}{T \cdot P_{\text{Light}}}$

Total expected cost:

Sampling **(repeat T times):**

Flip a coin that is head with probability **P_{Heavy}** :

- **Head**: sample **i ~ D_{Heavy}** with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample **i ~ D_{Light}** with **Quantum State Preparation.**

Cost per sample: $O\left(\sqrt{np_{\max}}\right)$ where $p_{\max} = \max\left\{\frac{p_i}{P_{\text{Light}}} : i \in \text{Light}\right\} \leq \frac{1}{T \cdot P_{\text{Light}}}$

Total expected cost: $O\left(T \cdot P_{\text{Light}} \cdot \sqrt{np_{\max}}\right)$

Importance Sampling with a Quantum Oracle

Sampling **(repeat T times):**

Flip a coin that is head with probability P_{Heavy} :

- **Head**: sample $i \sim D_{Heavy}$ with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample $i \sim D_{Light}$ with **Quantum State Preparation.**

Cost per sample: $O\left(\sqrt{np_{\max}}\right)$ where $p_{\max} = \max\left\{\frac{p_i}{P_{\text{Light}}} : i \in \text{Light}\right\} \leq \frac{1}{T \cdot P_{\text{Light}}}$

Total expected cost: $O\left(T \cdot P_{\text{Light}} \cdot \sqrt{np_{\max}}\right) = O\left(\sqrt{n \cdot T \cdot P_{\text{Light}}}\right)$

Importance Sampling with a Quantum Oracle

Sampling (repeat T times):

Flip a coin that is head with probability **P_{Heavy}** :

- **Head**: sample **$i \sim D_{Heavy}$** with the **Binary Tree Method.**

Cost per sample: $O(\log n)$ **Total cost:** $O(T \log n)$

- **Tail**: sample **$i \sim D_{Light}$** with **Quantum State Preparation.**

Cost per sample: $O\left(\sqrt{np_{\max}}\right)$ where $p_{\max} = \max\left\{\frac{p_i}{P_{\text{Light}}} : i \in \text{Light}\right\} \leq \frac{1}{T \cdot P_{\text{Light}}}$

Total expected cost: $O\left(T \cdot P_{\text{Light}} \cdot \sqrt{np_{\max}}\right) = O\left(\sqrt{n \cdot T \cdot P_{\text{Light}}}\right) = O\left(\sqrt{nT}\right)$

Conclusion

Recent improvement:

- Axelrod, Liu, Sidford 2019: classical $\tilde{O}(n/\epsilon^2)$ algorithm for approximate submodular function minimization

Recent improvement:

- Axelrod, Liu, Sidford 2019: classical $\tilde{O}(n/\epsilon^2)$ algorithm for approximate submodular function minimization

Open questions:

- Can we improve the upper/lower bounds for exact/approximate submodular function minimization?

- What are other applications of our quantum multi-sampling algorithm? (*ongoing work: solving linear systems)*

- Can we prepare T copies of the state $\sum_{i \in [n]} \sqrt{p_i} \, | \, i \rangle$ in time $O\big(\sqrt{nT}\big)$.

arXiv: 1907.05378