The Multiplicative Quantum Adversary

Robert Špalek

Quantum query complexity

Quantum query complexity

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$

Quantum query complexity

not necessarily
Boolean output

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$

Quantum query complexity

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$
- **Task:** compute f(x)

Quantum query complexity

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x

Quantum query complexity

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x
- *Query* is a unitary oracle operator mapping

$$O : |x\rangle_I |i\rangle_Q |w\rangle_W \rightarrow (-1)^{x_i} |x\rangle |i\rangle |w\rangle$$

Quantum query complexity

- Given a function f: $\{0,1\}^n \to \{0,1\}^m$
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x
- *Query* is a unitary oracle operator mapping

$$O : |x\rangle_I |i\rangle_Q |w\rangle_W \to (-1)^{x_i} |x\rangle |i\rangle |w\rangle$$

input register holding
$x \in \{0,1\}^n$

Quantum query complexity

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x
- *Query* is a unitary oracle operator mapping

$$O : |x\rangle_I |i\rangle_Q |w\rangle_W \rightarrow (-1)^{x_i} |x\rangle |i\rangle |w\rangle$$

query register holding
$i \in \{0,1,\ldots, n\}$

Quantum query complexity

- Given a function f: {0,1}^{n}→{0,1}^{m}
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x
- *Query* is a unitary oracle operator mapping

$$O : |x\rangle_I |i\rangle_Q |w\rangle_W \rightarrow (-1)^{x_i} |x\rangle |i\rangle |w\rangle$$

workspace register holding arbitrary algorithm data

Quantum query complexity

- Given a function f: $\{0,1\}^n \rightarrow \{0,1\}^m$
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x
- *Query* is a unitary oracle operator mapping

$$O : |x\rangle_I |i\rangle_Q |w\rangle_W \rightarrow (-1)^{x_i} |x\rangle |i\rangle |w\rangle$$

the value of the input
is stored in the phase

Quantum query complexity

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x
- *Query* is a unitary oracle operator mapping

$$O: |x\rangle_I |i\rangle_Q |w\rangle_W \rightarrow (-1)^{x_i} |x\rangle |i\rangle |w\rangle$$

- The algorithm can perform arbitrary *unitary operations* on its workspace and the query register for free

Quantum query complexity

- Given a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$
- **Task:** compute f(x)
- *Query complexity* $Q_\epsilon(f)$ is the minimal T such that there exists a T-query quantum algorithm that computes f(x) with error probability at most ϵ on each input x
- *Query* is a unitary oracle operator mapping

$$O : |x\rangle_I |i\rangle_Q |w\rangle_W \rightarrow (-1)^{x_i} |x\rangle |i\rangle |w\rangle$$

- The algorithm can perform arbitrary *unitary operations* on its workspace and the query register for free
- At the end, it *measures* its workspace, outputs an outcome, and then we measure the input register and verify the outcome

Adversary bounds

lower-bound quantum query complexity

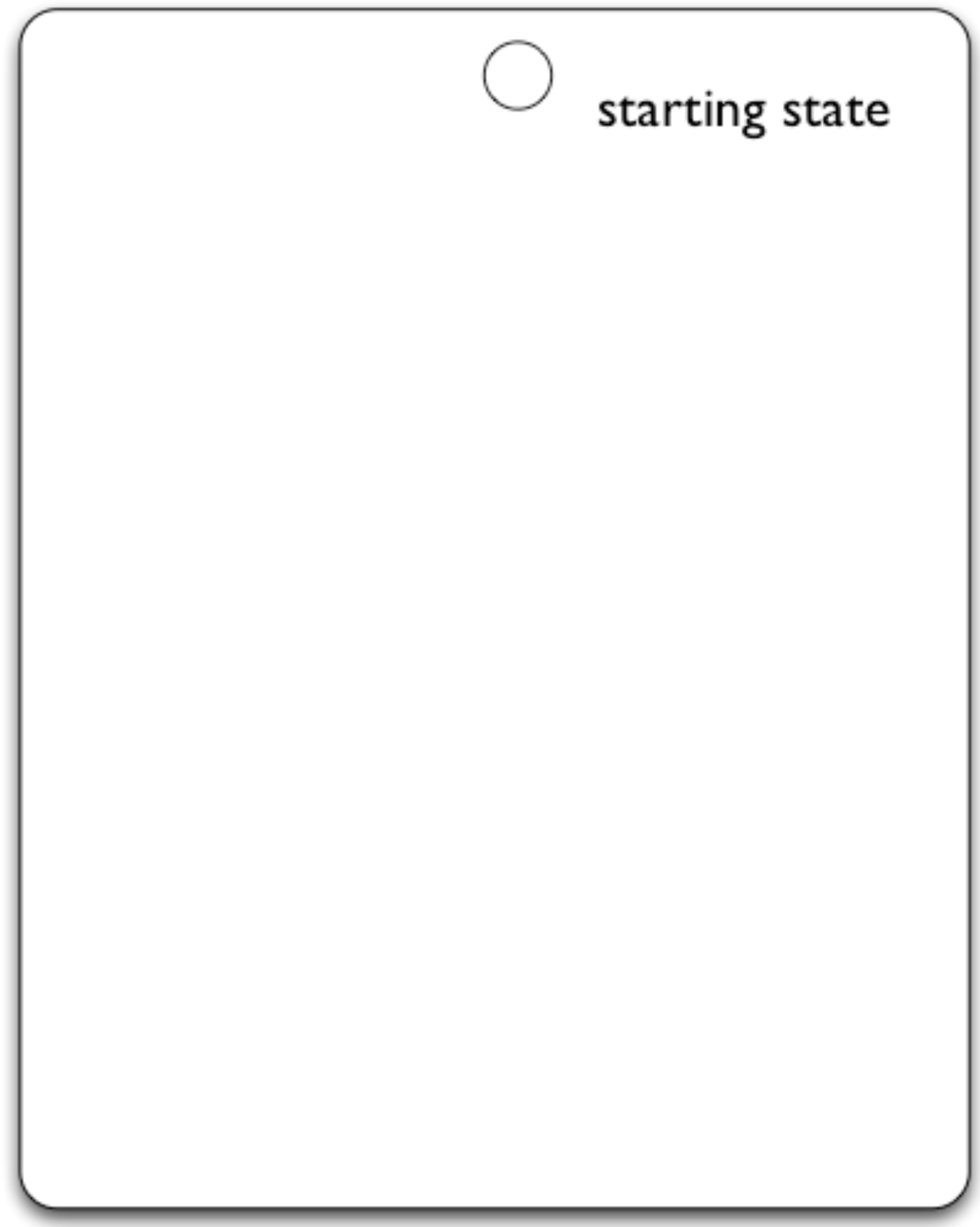
Adversary bounds

lower-bound quantum query complexity

state of computation on input x at time 0

Idea:

- computation starts in a fixed state $|\varphi_x^0\rangle = |\varphi\rangle$ independent of input x



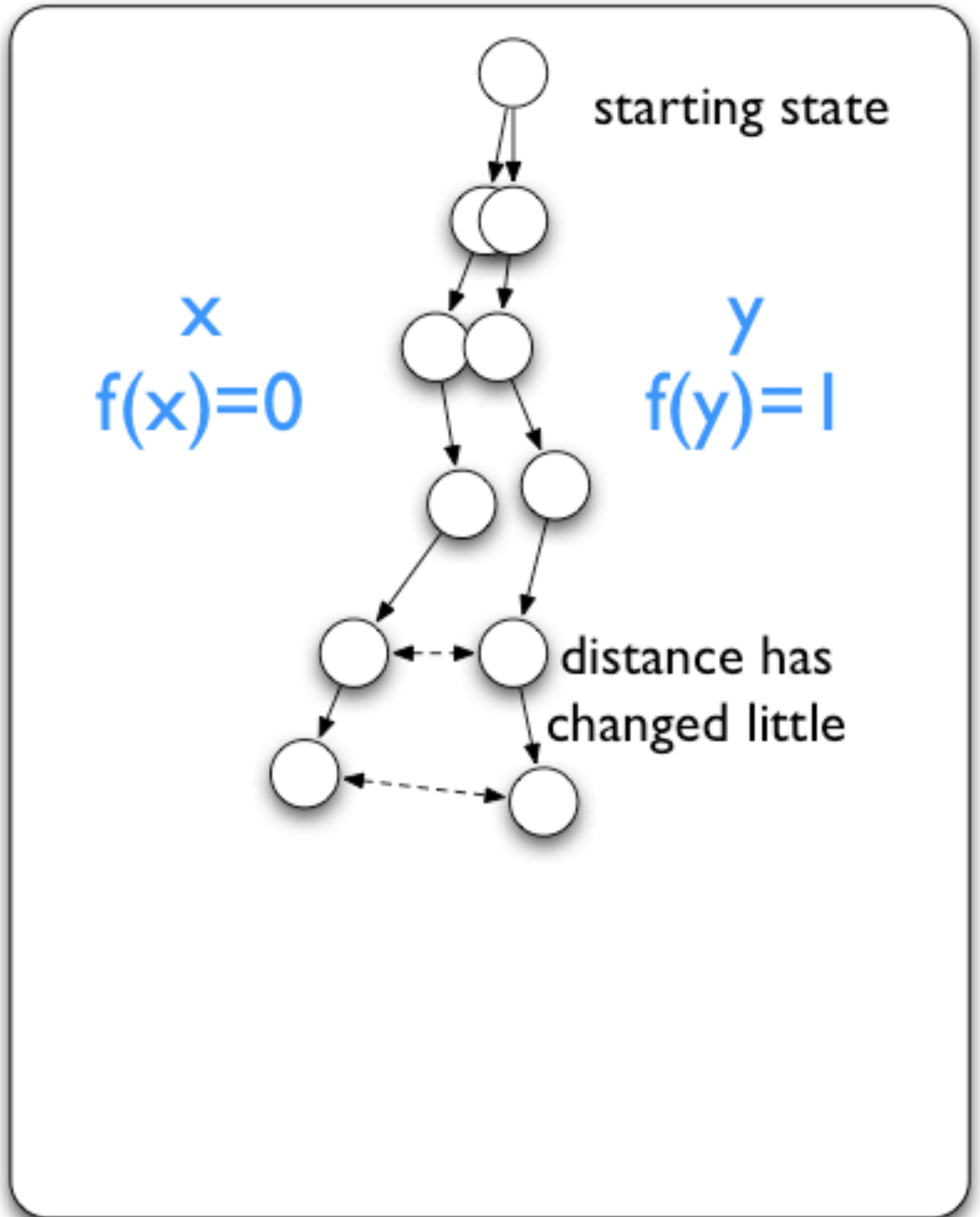
Adversary bounds

lower-bound quantum query complexity

Idea:

scalar product of the states on inputs x and y

- computation starts in a fixed state $|\varphi_x^0\rangle = |\varphi\rangle$ independent of input x
- one query can only change $\langle\varphi_x^t|\varphi_y^t\rangle$ by a small amount, on the average

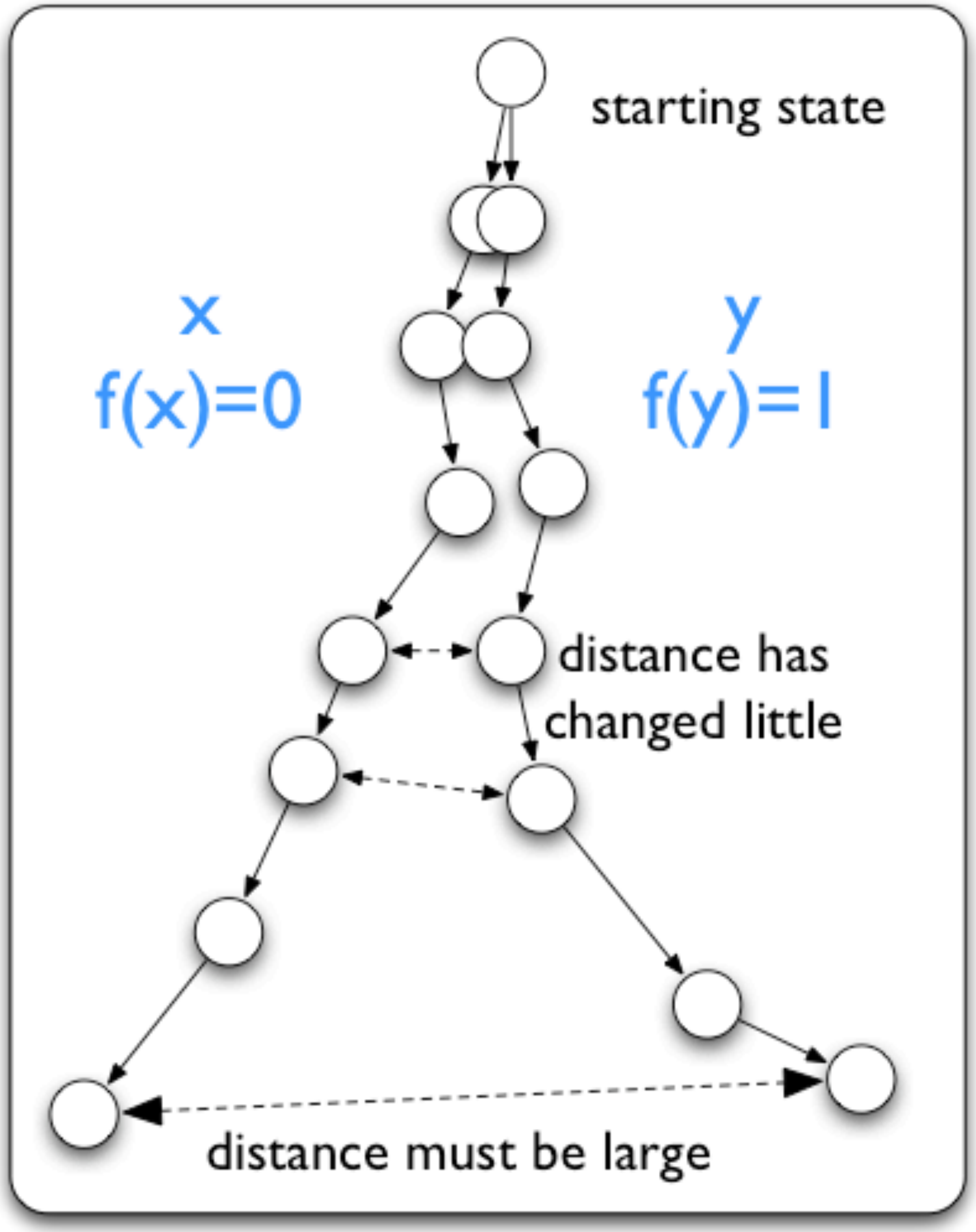


Adversary bounds

lower-bound quantum query complexity

Idea:

- computation starts in a fixed state $|\varphi_x^0\rangle = |\varphi\rangle$ independent of input x
- one query can only change $\langle\varphi_x^t|\varphi_y^t\rangle$ by a small amount, on the average
- at the end, $\langle\varphi_x^T|\varphi_y^T\rangle$ must be small for each input pair x, y with $f(x) \neq f(y)$, otherwise the algorithm cannot distinguish x and y



Adversary bounds

lower-bound quantum query complexity

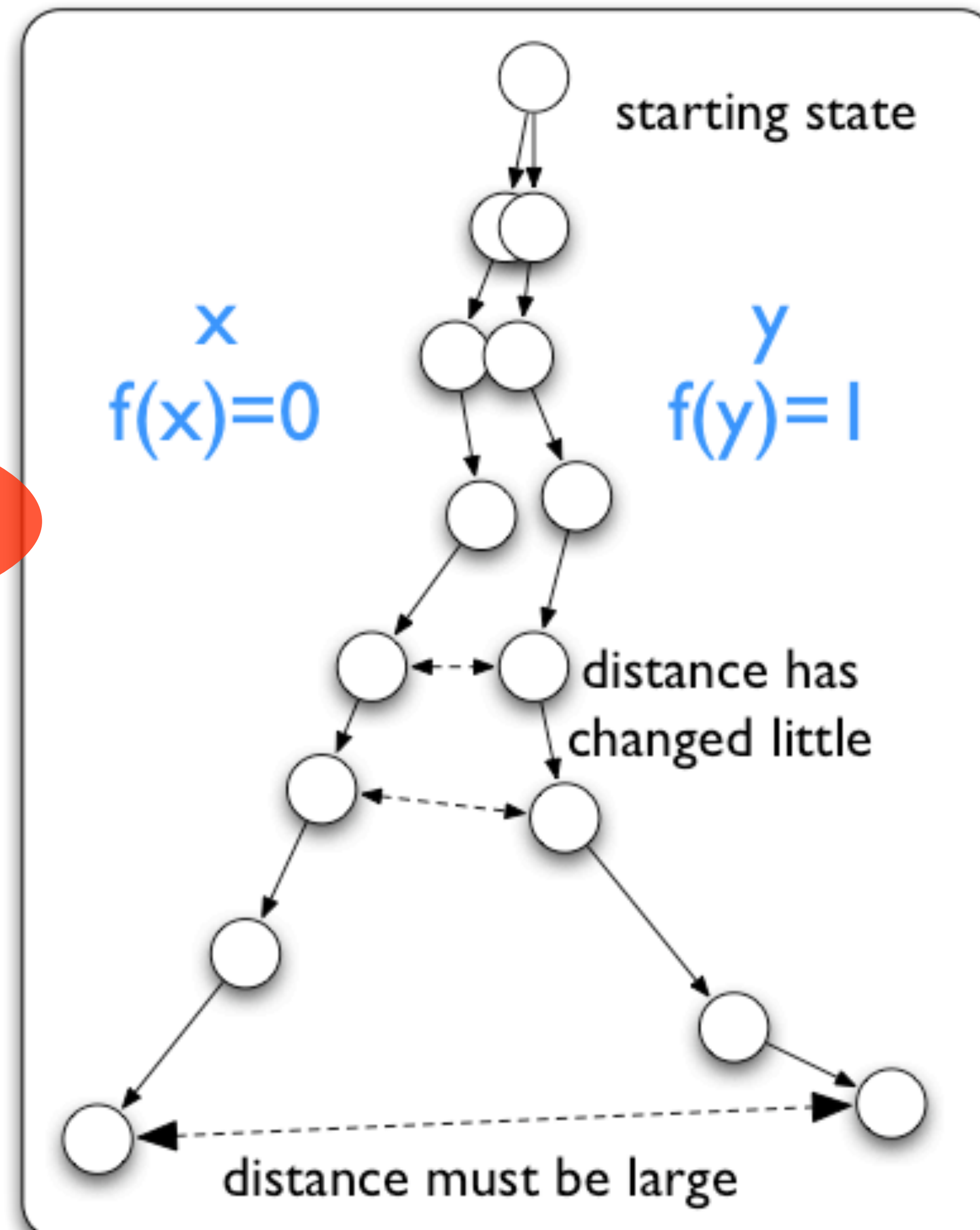
Idea:

- computation starts in a fixed state $|\varphi_x^0\rangle = |\varphi\rangle$ independent of input x
- one query can only change $\langle \varphi_x^t | \varphi_y^t \rangle$ by a small amount, on the average

the bound on T depends on the average

- at the end, $\langle \varphi_x^T | \varphi_y^T \rangle$ must be small for each input pair x, y with f(x)≠f(y), otherwise the algorithm cannot distinguish x and y

➡ T must be large



History of the adversary method

History of the adversary method

- [Bennett, Bernstein, Brassard & Vazirani '94]
  hybrid method

History of the adversary method

- [Bennett, Bernstein, Brassard & Vazirani '94]
  hybrid method
- [Ambainis '00] adversary method

History of the adversary method

- [Bennett, Bernstein, Brassard & Vazirani '94] hybrid method
- [Ambainis '00] adversary method
- [Høyer, Neerbek & Shi '02] early weighted method

History of the adversary method

- [Bennett, Bernstein, Brassard & Vazirani '94]
  hybrid method
- [Ambainis '00] adversary method
- [Høyer, Neerbek & Shi '02]
  early weighted method
- [Barnum, Saks & Szegedy '03]
  spectral method

  [Ambainis '03]
  weighted adversary method

History of the adversary method

- [Bennett, Bernstein, Brassard & Vazirani '94]
  hybrid method
- [Ambainis '00] adversary method
- [Høyer, Neerbek & Shi '02]
  early weighted method
- [Barnum, Saks & Szegedy '03]
  spectral method

  [Ambainis '03]
  weighted adversary method

History of the adversary method

- [Bennett, Bernstein, Brassard & Vazirani '94]
  hybrid method
- [Ambainis '00] adversary method
- [Høyer, Neerbek & Shi '02]
  early weighted method
- [Barnum, Saks & Szegedy '03]
  spectral method

  [Ambainis '03]
  weighted adversary method
- [Høyer, Lee & S. '07]
  negative weights

Spectral method

Spectral method

- Define a *progress function* in time t:

$$W^t = \langle \Gamma, \rho_I^t \rangle$$

Spectral method

- Define a *progress function* in time t:

$$W^t = \langle \Gamma, \rho_I^t \rangle$$

  - ρ_I^t is reduced density matrix of the input register at time t

Spectral method

weighted average of the scalar products

- Define a *progress function* in time t:

$$W^t = \langle \Gamma, \rho_I^t \rangle$$

  - ρ_I^t is reduced density matrix of the input register at time t
  - Γ is the *adversary matrix* for f:
    Hermitian and $\Gamma_{x,y} = 0$ when f(x)=f(y)

Spectral method

- Define a *progress function* in time t:

$$W^t = \langle \Gamma, \rho_I^t \rangle$$

  - ρ_I^t is reduced density matrix of the input register at time t
  - Γ is the *adversary matrix* for f:
    Hermitian and $\Gamma_{x,y} = 0$ when $f(x)=f(y)$

- Run the computation on certain input superposition

Spectral method

- Define a *progress function* in time t:

$$W^t = \langle \Gamma, \rho_I^t \rangle$$

  - $\rho_I{}^t$ is reduced density matrix of the input register at time t
  - Γ is the *adversary matrix* for f:
    Hermitian and $\Gamma_{x,y} = 0$ when $f(x)=f(y)$

therefore we call it
additive adversary

- Run the computation on certain input superposition
- Upper-bound the difference $W^{t+1}-W^t$

Spectral method

- Define a *progress function* in time t:

$$W^t = \langle \Gamma, \rho_I^t \rangle$$

  - ρ_I^t is reduced density matrix of the input register at time t
  - Γ is the *adversary matrix* for f:
    Hermitian and $\Gamma_{x,y} = 0$ when $f(x)=f(y)$
- Run the computation on certain input superposition
- Upper-bound the difference $W^{t+1}-W^t$

➡ Leads to the bound

$$\mathrm{Adv}_\epsilon(f) = \left(\frac{1}{2} - \sqrt{\epsilon(1-\epsilon)}\right) \max_{\Gamma} \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$$

Spectral method

- Define a *progress function* in time t:

$$W^t = \langle \Gamma, \rho_I^t \rangle$$

  - ρ_I^t is reduced density matrix of the input register at time t
  - Γ is the *adversary matrix* for f:
    Hermitian and $\Gamma_{x,y} = 0$ when $f(x)=f(y)$
- Run the computation on certain input superposition
- Upper-bound the difference $W^{t+1}-W^t$

➡ Leads to the bound

spectral norm

$$\mathrm{Adv}_\epsilon(f) = \left(\frac{1}{2} - \sqrt{\epsilon(1-\epsilon)}\right) \max_\Gamma \frac{\|\Gamma\|}{\max_i \|\Gamma_i\|}$$

sub-matrix of Γ with zeroes when $x_i=y_i$

Pros and cons of additive adversary

Pros and cons of additive adversary

- **Pros:**
  - universal method: works for all functions
  - often gives optimal bounds (e.g., search, sorting, graph problems)
  - Γ, δ are intuitive: hard distribution on input pairs and inputs
  - easy to compute
  - composes optimally with respect to function composition

Pros and cons of additive adversary

- **Pros:**
  - universal method: works for all functions
  - often gives optimal bounds (e.g., search, sorting, graph problems)
  - Γ, δ are intuitive: hard distribution on input pairs and inputs
  - easy to compute
  - composes optimally with respect to function composition
- **Cons:**
  - gives trivial bound for low success probability
  - no direct product theorem

Pros and cons of additive adversary

we overcome the cons

- **Pros:**
  - universal method: works for all functions
  - often gives optimal bounds (e.g., search, sorting, graph problems)
  - Γ, δ are intuitive: hard distribution on input pairs and inputs
  - easy to compute
  - composes optimally with respect to function composition
- **Cons:**
  - gives trivial bound for low success probability
  - no direct product theorem

Pros and cons of additive adversary

- **Pros:**
  - universal method: works for all functions
  - often gives optimal bounds (e.g., search, sorting, graph problems)
  - Γ, δ are intuitive: hard distribution on input pairs and inputs
  - easy to compute
  - composes optimally with respect to function composition

- **Cons:**
  - gives trivial bound for low success probability
  - no direct product theorem

and lose these pros

Pros and cons of additive adversary

- **Pros:**
  - universal method:
    works for all functions
  - often gives optimal bounds (e.g., search, sorting, graph problems)
  - Γ, δ are intuitive:
    hard distribution on input pairs and inputs
  - easy to compute
  - composes optimally with respect to function composition
- **Cons:**
  - gives trivial bound for low success probability
  - no direct product theorem

Origin of our method

Origin of our method

- Problem: search k ones in an n-bit input.

Origin of our method

- Problem: search k ones in an n-bit input.
- [Ambainis '05] new method based on analysis of eigenspaces of the reduced density matrix of the input register
    - $\Omega(\sqrt{kn})$ queries are needed even for success $2^{-O(k)}$
    - reproving the result of [Klauck, S. & de Wolf '04] based on the polynomial method.

Origin of our method

- Problem: search k ones in an n-bit input.
- [Ambainis '05] new method based on analysis of eigenspaces of the reduced density matrix of the input register
  - $\Omega(\sqrt{kn})$ queries are needed even for success $2^{-O(k)}$
  - reproving the result of [Klauck, S. & de Wolf '04] based on the polynomial method.
- Pros:
  - tight bound not relying on polynomial approximation theory

Origin of our method

- Problem: search k ones in an n-bit input.
- [Ambainis '05] new method based on analysis of eigenspaces of the reduced density matrix of the input register
  - $\Omega(\sqrt{kn})$ queries are needed even for success $2^{-O(k)}$
  - reproving the result of [Klauck, S. & de Wolf '04] based on the polynomial method.
- Pros:
  - tight bound not relying on polynomial approximation theory
- Cons:
  - tailored to one specific problem
  - technical, complicated, non-modular proof without much intuition

Origin of our method

Origin of our method

- [Ambainis '05] new method based on analysis of eigenspaces of the reduced density matrix of the input register

Origin of our method

- [Ambainis '05] new method based on analysis of eigenspaces of the reduced density matrix of the input register
- We improve his method as follows:
  - put it into the well-studied adversary framework
  - generalize it to all functions
  - provide additional intuition, modularize the proof, and separate the quantum and combinatorial part

Origin of our method

- [Ambainis '05] new method based on analysis of eigenspaces of the reduced density matrix of the input register
- We improve his method as follows:
  - put it into the well-studied adversary framework
  - generalize it to all functions
  - provide additional intuition, modularize the proof, and separate the quantum and combinatorial part
- However, the underlying combinatorial analysis stays the same and we cannot omit any single detail

New type of adversary

New type of adversary

- Differences:
  - *adversary matrix* Γ has different semantics then before
  - We upper-bound the ratio W^{t+1}/W^{t}, not difference

New type of adversary

- Differences:
  - *adversary matrix* Γ has different semantics than before
  - We upper-bound the ratio W^{t+1}/W^{t}, not difference

now, guess the name of our method

Multiplicative adversary

- Differences:
  - *adversary matrix* Γ has different semantics then before
  - We upper-bound the ratio W^{t+1}/W^{t}, not difference

Multiplicative adversary

- Differences:
  - *adversary matrix* Γ has different semantics then before
  - We upper-bound the ratio W^{t+1}/W^t, not difference
- The bound looks similar, however, it requires common block-diagonalization of Γ and the input oracle O_i, and therefore is extremely hard to compute

Multiplicative adversary

- Differences:
  - *adversary matrix* Γ has different semantics then before
  - We upper-bound the ratio W^{t+1}/W^{t}, not difference
- The bound looks similar, however, it requires common block-diagonalization of Γ and the input oracle O_i, and therefore is extremely hard to compute

additive: $$\|\Gamma\| \cdot \min_i \frac{1}{\|\Gamma_i\|}$$

mutliplicative: $$\log(\|\Gamma\|) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^k)}{\|\Gamma_i^k\|}$$

Multiplicative adversary

- Differences:
  - *adversary matrix* Γ has different semantics then before
  - We upper-bound the ratio W^{t+1}/W^t, not difference
- The bound looks similar, however, it requires common block-diagonalization of Γ and the input oracle O_i, and therefore is extremely hard to compute

sub-matrix of Γ with zeroes when $x_i=y_i$

additive: $\|\Gamma\| \cdot \min_i \frac{1}{\|\Gamma_i\|}$

mutliplicative: $\log(\|\Gamma\|) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^k)}{\|\Gamma_i^k\|}$

Multiplicative adversary

- Differences:
  - *adversary matrix* Γ has different semantics then before
  - We upper-bound the ratio W^{t+1}/W^t, not difference
- The bound looks similar, however, it requires common block-diagonalization of Γ and the input oracle O_i, and therefore is extremely hard to compute

Γ^k is the k-th block on the diagonal

additive: $\|\Gamma\| \cdot \min_i \frac{1}{\|\Gamma_i\|}$

mutliplicative: $\log(\|\Gamma\|) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^k)}{\|\Gamma_i^k\|}$

Multiplicative adversary

- Differences:
  - *adversary matrix* Γ has different semantics then before
  - We upper-bound the ratio W^{t+1}/W^t, not difference
- The bound looks similar, however, it requires common block-diagonalization of Γ and the input oracle O_i, and therefore is extremely hard to compute

$\lambda_{min}(M)$ is the smallest eigenvalue of M

additive: $$\|\Gamma\| \cdot \min_i \frac{1}{\|\Gamma_i\|}$$

mutliplicative: $$\log(\|\Gamma\|) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^k)}{\|\Gamma_i^k\|}$$

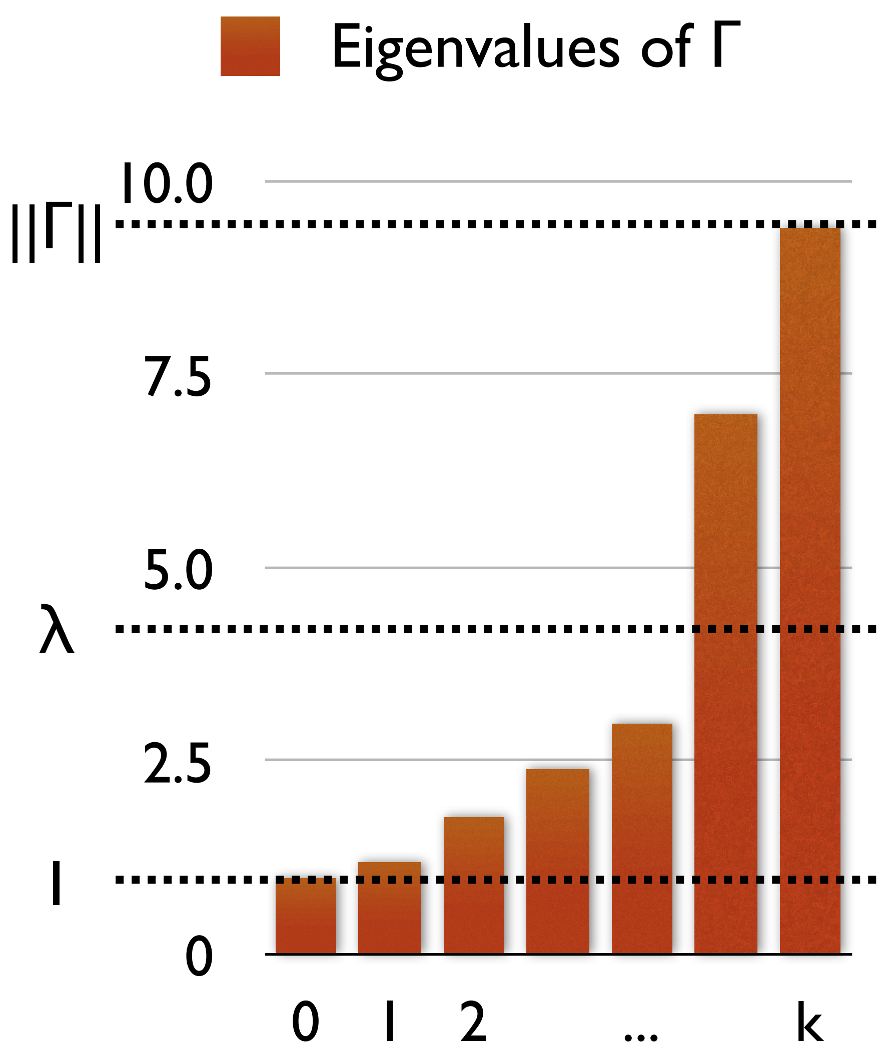
Multiplicative adversary matrix

Multiplicative adversary matrix

- Consider a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$, a positive definite matrix Γ with minimal eigenvalue 1, and $1 < \lambda \leq ||\Gamma||$:

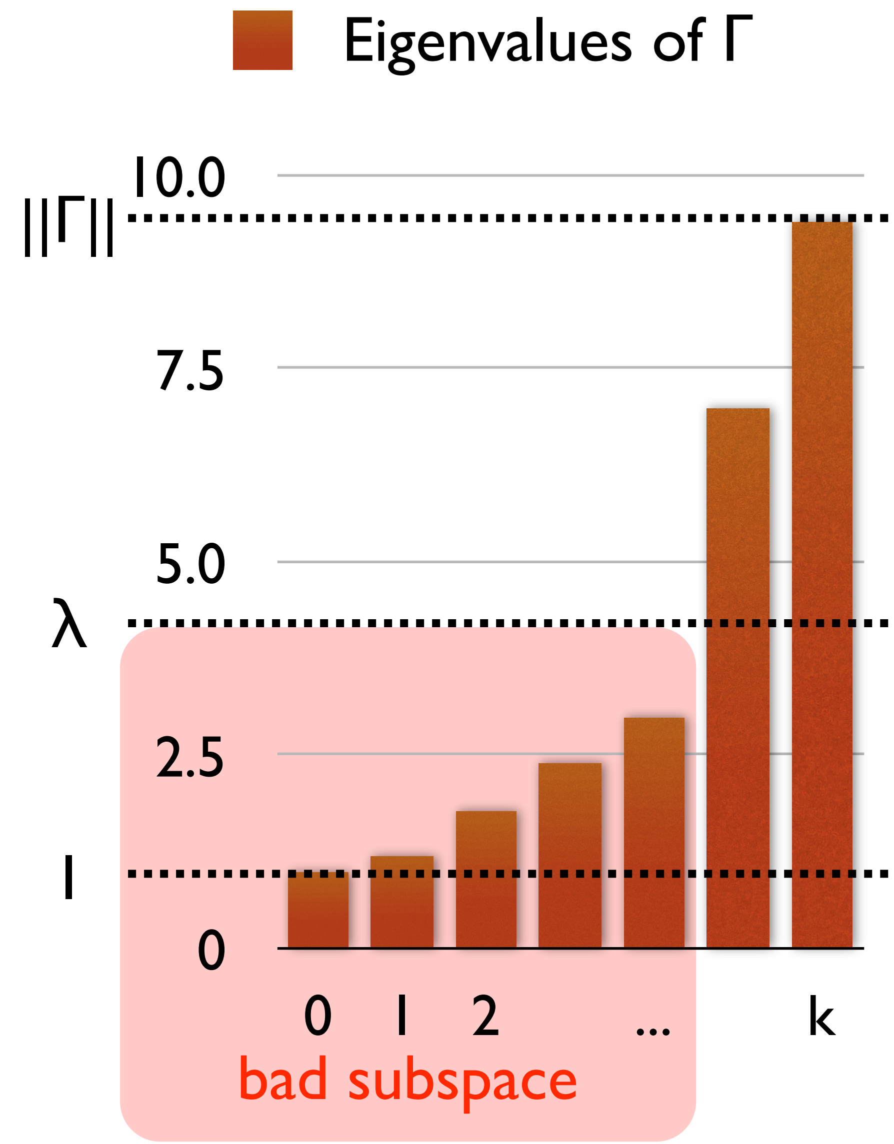
Multiplicative adversary matrix

- Consider a function f: $\{0,1\}^n \rightarrow \{0,1\}^m$, a positive definite matrix Γ with minimal eigenvalue 1, and $1 < \lambda \leq \|\Gamma\|$:



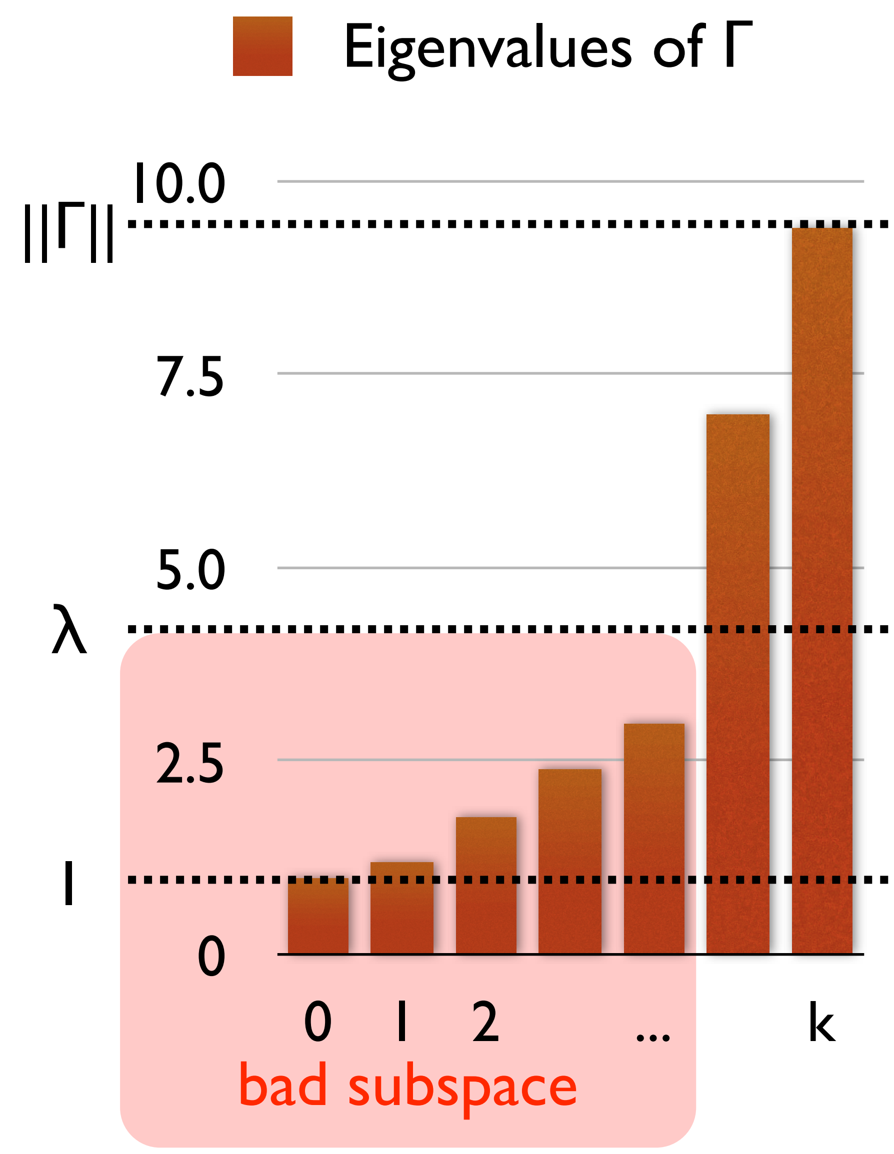
Multiplicative adversary matrix

- Consider a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$, a positive definite matrix Γ with minimal eigenvalue 1, and $1 < \lambda \leq \|\Gamma\|$:
  - $\Pi_{\mathbf{bad}}$ is a projector onto the bad subspace, which is the direct sum of all eigenspaces corresponding to eigenvalues smaller than λ



Multiplicative adversary matrix

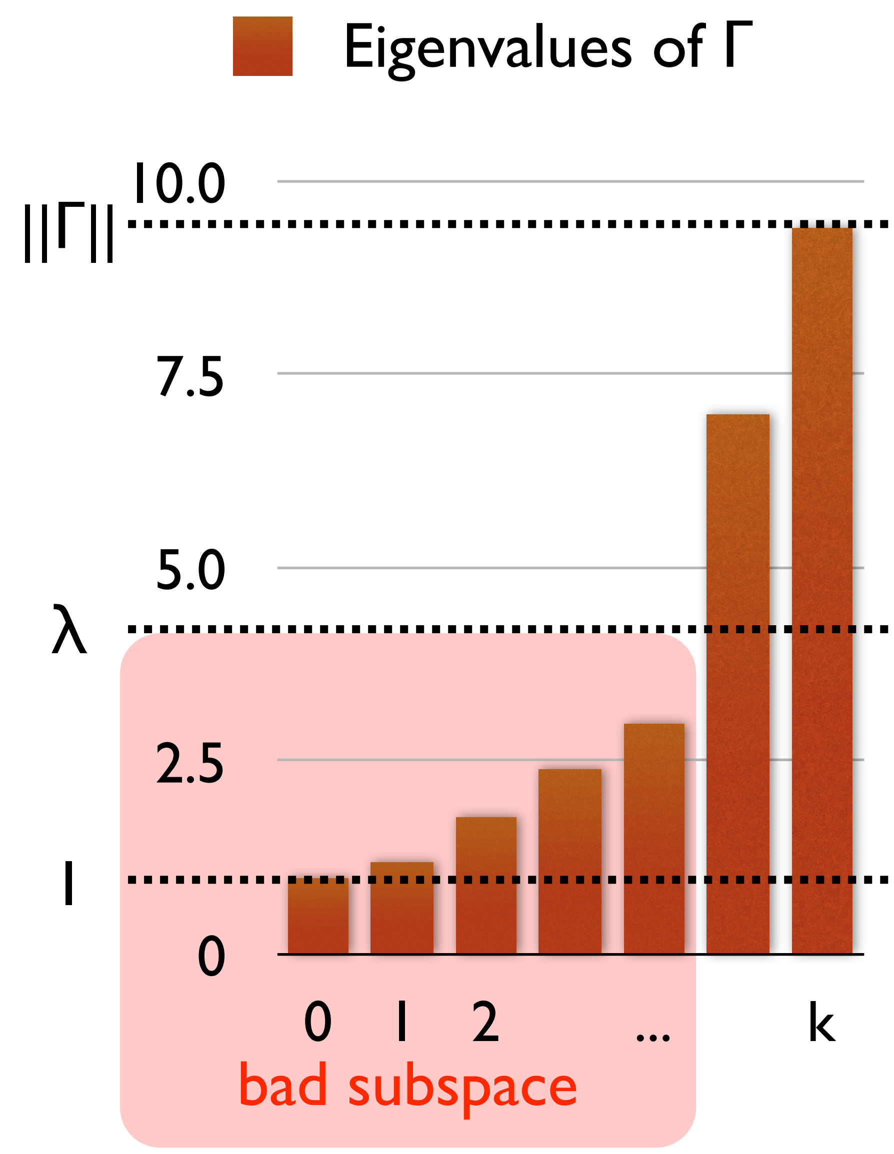
- Consider a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$, a positive definite matrix Γ with minimal eigenvalue 1, and $1 < \lambda \leq \|\Gamma\|$:
  - $\Pi_{\mathbf{bad}}$ is a projector onto the bad subspace, which is the direct sum of all eigenspaces corresponding to eigenvalues smaller than λ
  - $\mathbf{F_z}$ is a diagonal projector onto inputs evaluating to z



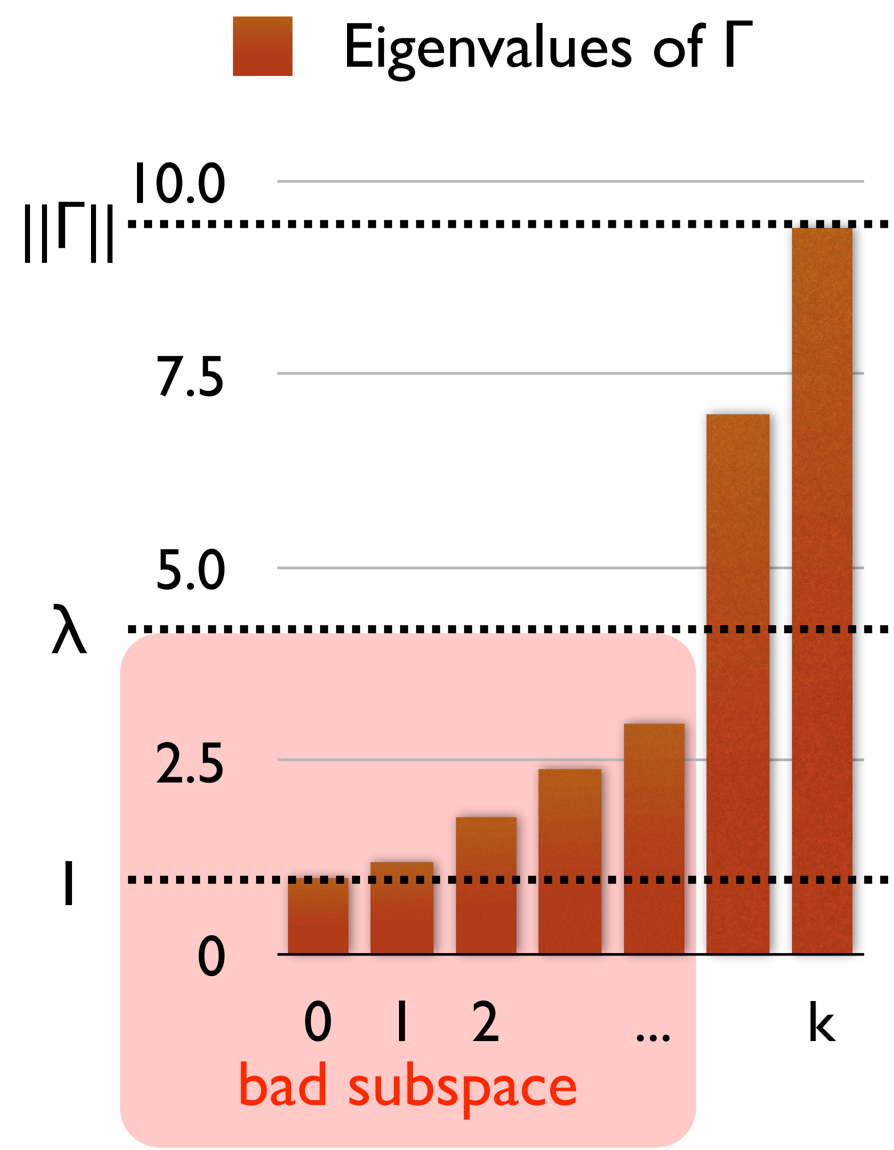
Multiplicative adversary matrix

- Consider a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$, a positive definite matrix Γ with minimal eigenvalue 1, and $1 < \lambda \leq ||\Gamma||$:
  - $\Pi_{\mathbf{bad}}$ is a projector onto the bad subspace, which is the direct sum of all eigenspaces corresponding to eigenvalues smaller than λ
  - $\mathbf{F_z}$ is a diagonal projector onto inputs evaluating to z
- (Γ,λ) is a multiplicative adversary for success probability η iff

for every $z \in \{0,1\}^m$, $||F_z \Pi_{bad}|| \leq \eta$



Multiplicative adversary matrix

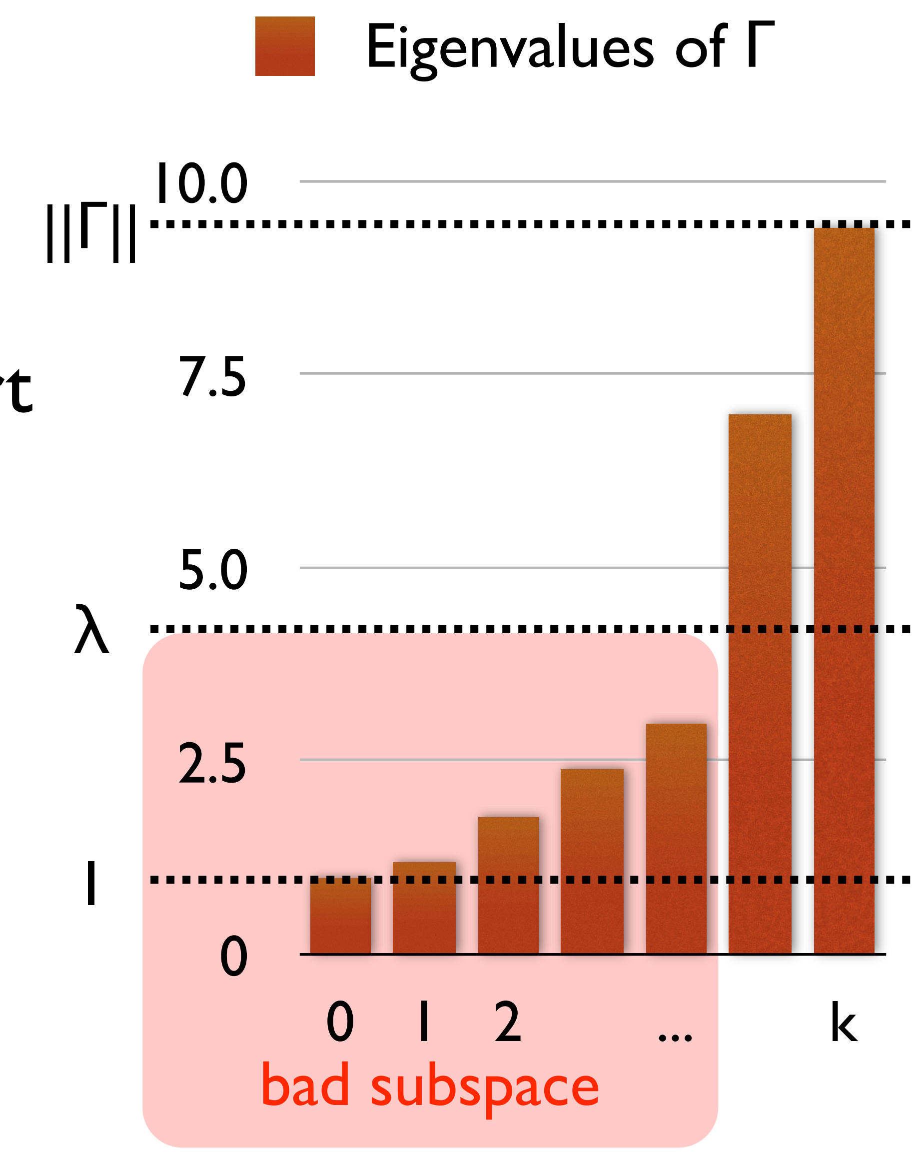


for every $z \in \{0,1\}^m$, $\|F_z \Pi_{bad}\| \leq \eta$

Multiplicative adversary matrix

for every $z \in \{0,1\}^m$, $||F_z \Pi_{bad}|| \leq \eta$

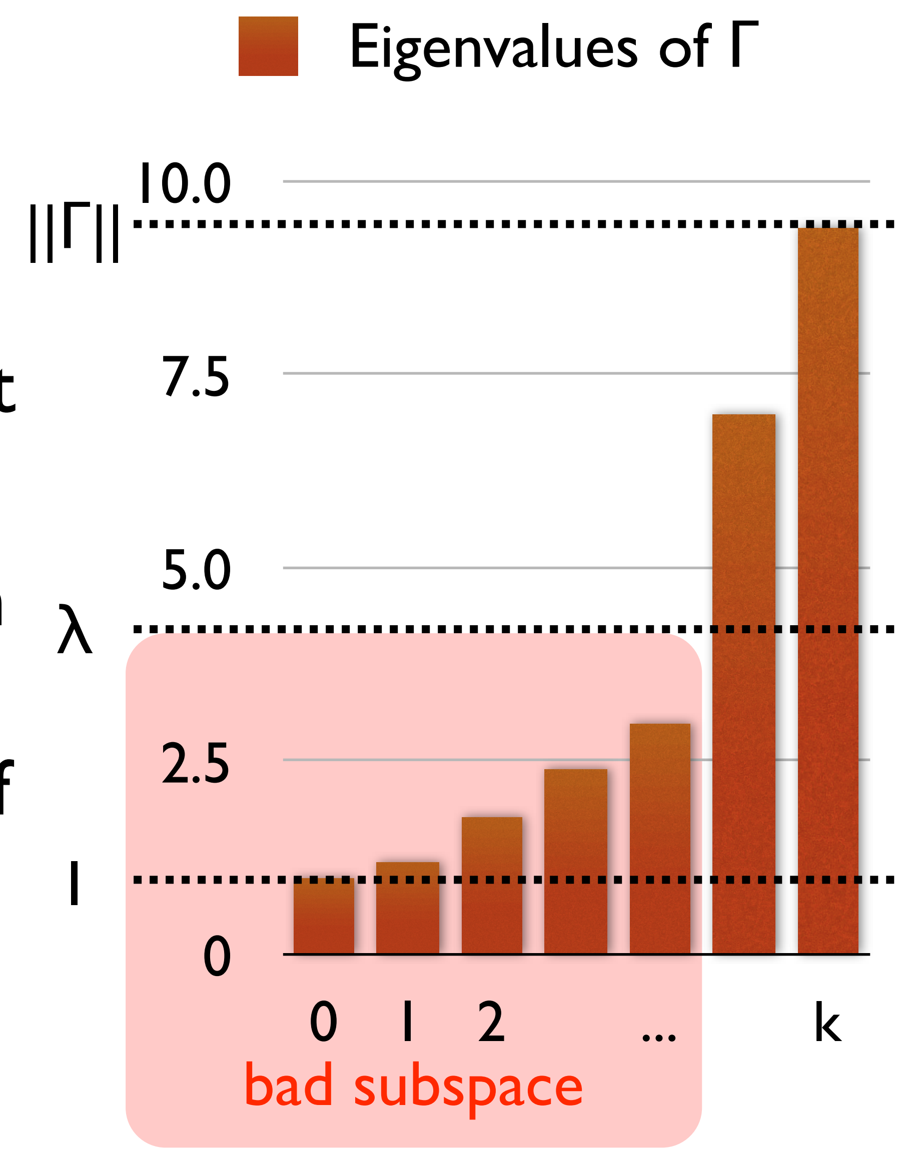
- It says that each vector (= superposition of inputs) from the bad subspace has short projection onto **each** F_z



Multiplicative adversary matrix

for every $z \in \{0,1\}^m$, $\|F_z \Pi_{bad}\| \leq \eta$

- It says that each vector (= superposition of inputs) from the bad subspace has short projection onto **each** F_z
- If the final state of the input register lies in the bad subspace, then the algorithm has success probability at most η regardless of the outcome it outputs. Typically, η is the trivial success probability of a random choice.



Evolution of the progress function

Evolution of the progress function

- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)

Evolution of the progress function

- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)
- We run A on input δ with $\Gamma\delta=\delta$. Then:

1. $W^0=1$
2. each $W^{t+1}/W^t \leq \max_i ||O_i\Gamma O_i \Gamma^{-1}||$
3. $W^T \geq \lambda \zeta^2/16$

Evolution of the progress function

- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)

trivial

- We run A on input δ with $\Gamma\delta=\delta$. Then:

  1. $W^0=1$
  2. each $W^{t+1}/W^t \leq \max_i \|O_i \Gamma O_i \Gamma^{-1}\|$
  3. $W^T \geq \lambda\, \zeta^2/16$

- **Proof:**

Evolution of the progress function

- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)
- We run A on input δ with $\Gamma\delta=\delta$. Then:
  1. $W^0=1$
  2. each $W^{t+1}/W^t \leq \max_i \|O_i \Gamma O_i \Gamma^{-1}\|$
  3. $W^T \geq \lambda \zeta^2/16$
- **Proof:**

very simple:

W^t is average of scalar products of $|\varphi_x^t\rangle$

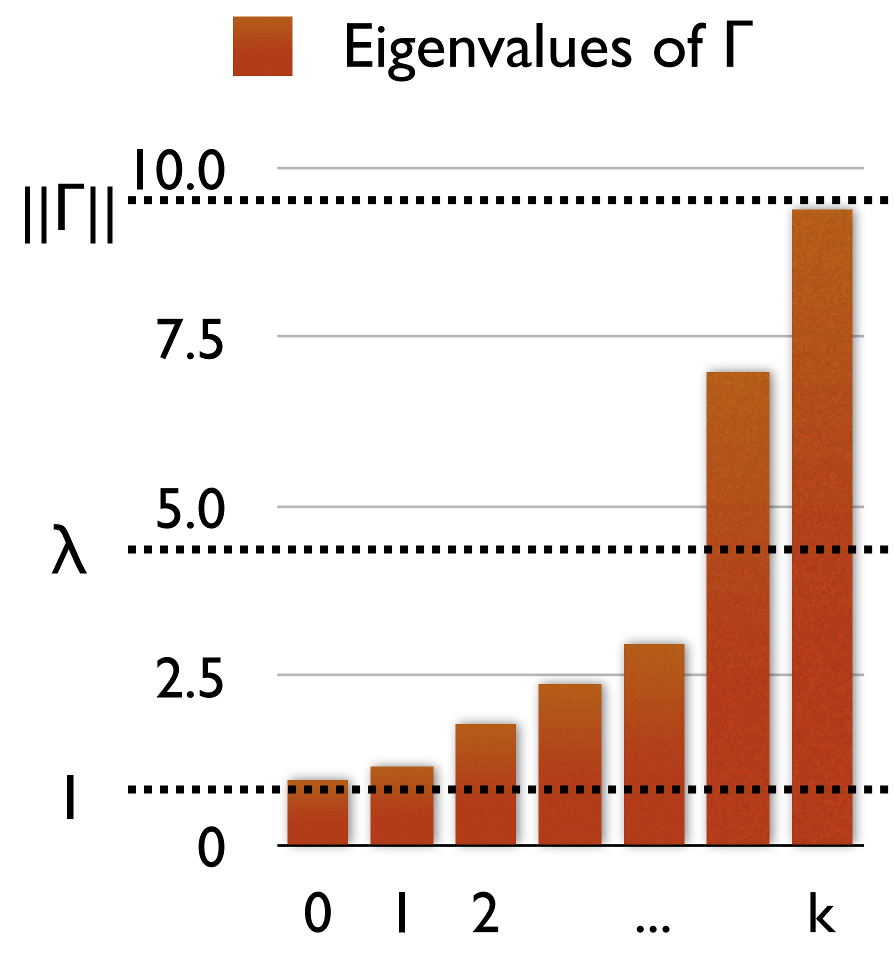
W^{t+1} is average of scalar products of $U_{t+1}O|\varphi_x^t\rangle$

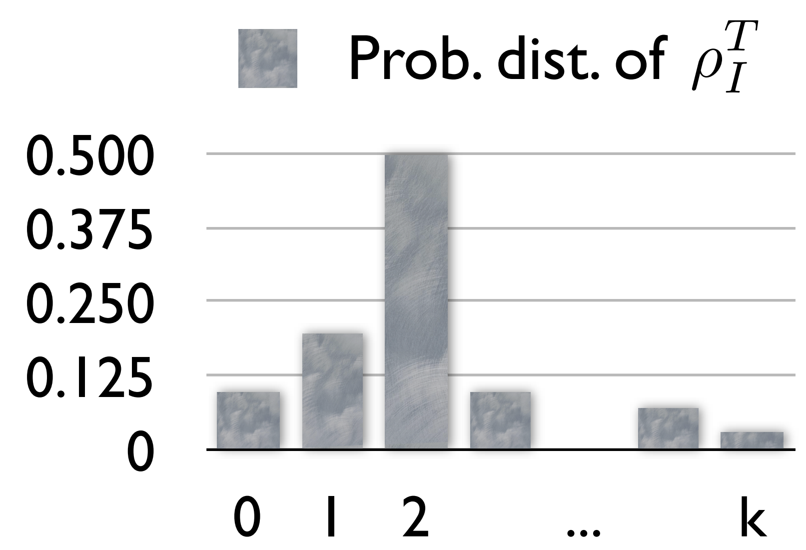
The unitaries cancel and the oracle calls can be absorbed into Γ, forming $O_i\Gamma O_i$, where

$$O_i : |x\rangle \rightarrow (-1)^{x_i}|x\rangle$$

Evolution of the progress function

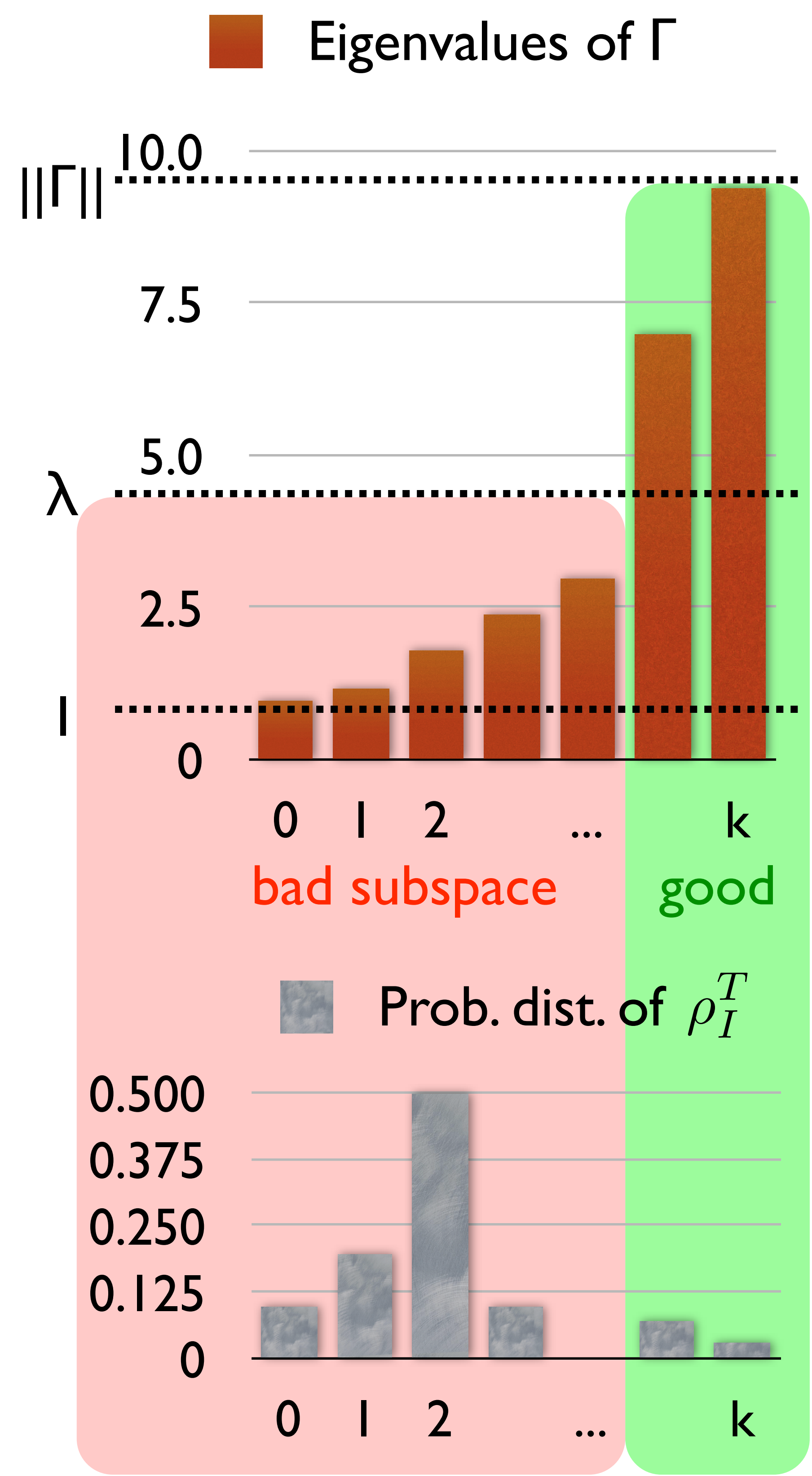
- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)
- We run A on input δ with $\Gamma\delta=\delta$. Then:
  1. $W^0=1$
  2. each $W^{t+1}/W^t \leq \max_i \|O_i \Gamma O_i \Gamma^{-1}\|$
  3. $W^T \geq \lambda \zeta^2/16$
- **Proof:**





Evolution of the progress function

- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)
- We run A on input δ with $\Gamma\delta=\delta$. Then:
  1. $W^0=1$
  2. each $W^{t+1}/W^t \leq \max_i ||O_i\Gamma O_i\,\Gamma^{-1}||$
  3. $W^T \geq \lambda\,\zeta^2/16$
- **Proof:**



Evolution of the progress function

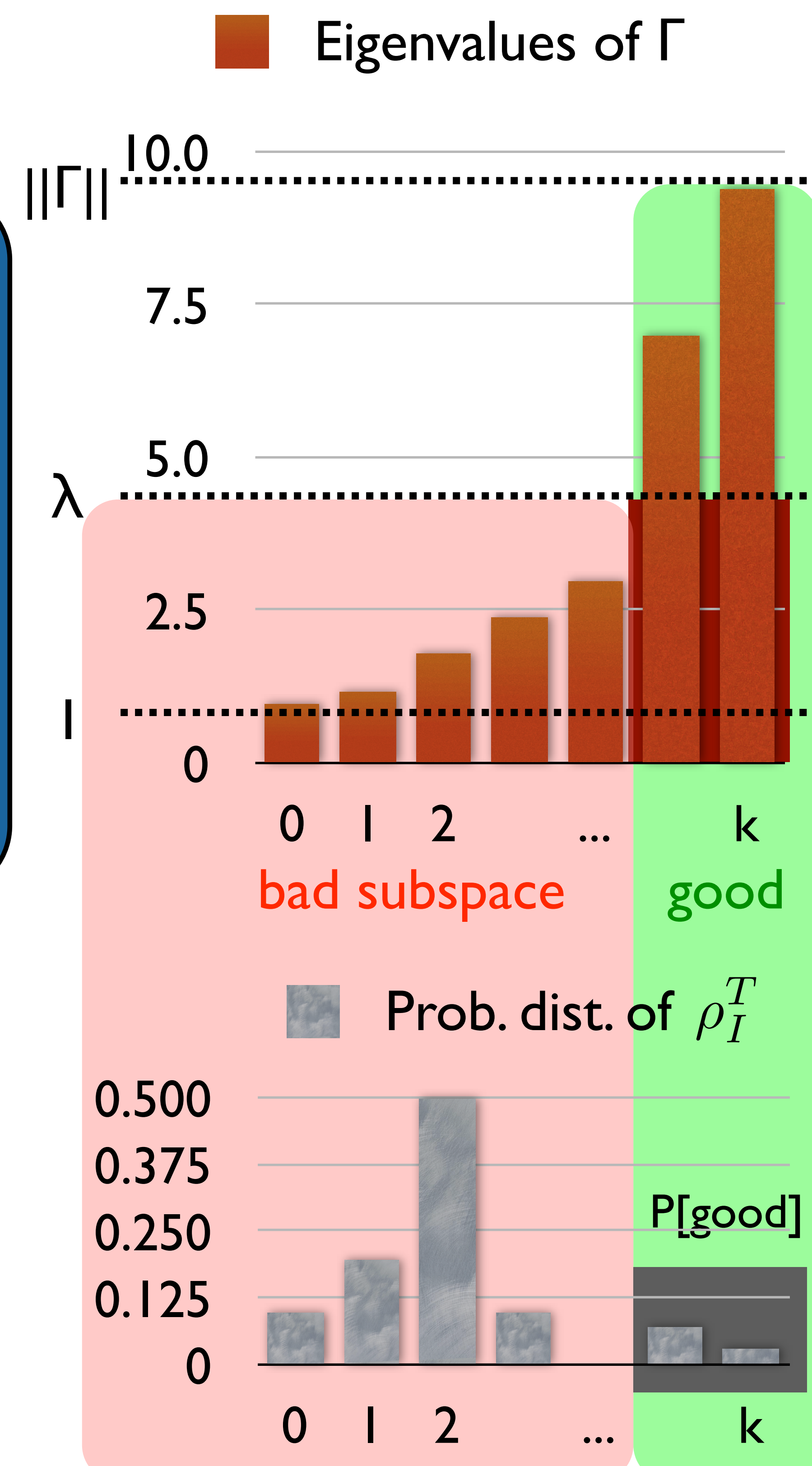
- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)
- We run A on input σ with $\Gamma\sigma=\sigma$. Then:
  1. $W^0=1$
  2. each $W^{t+1}/W^t \leq \max_i \|O_i \Gamma O_i \Gamma^{-1}\|$
  3. $W^T \geq \lambda\, \zeta^2/16$
- **Proof:**

Lower-bound area under curve

$$\langle \Gamma, \rho_I^T \rangle \geq \lambda \cdot P[\text{good}]$$

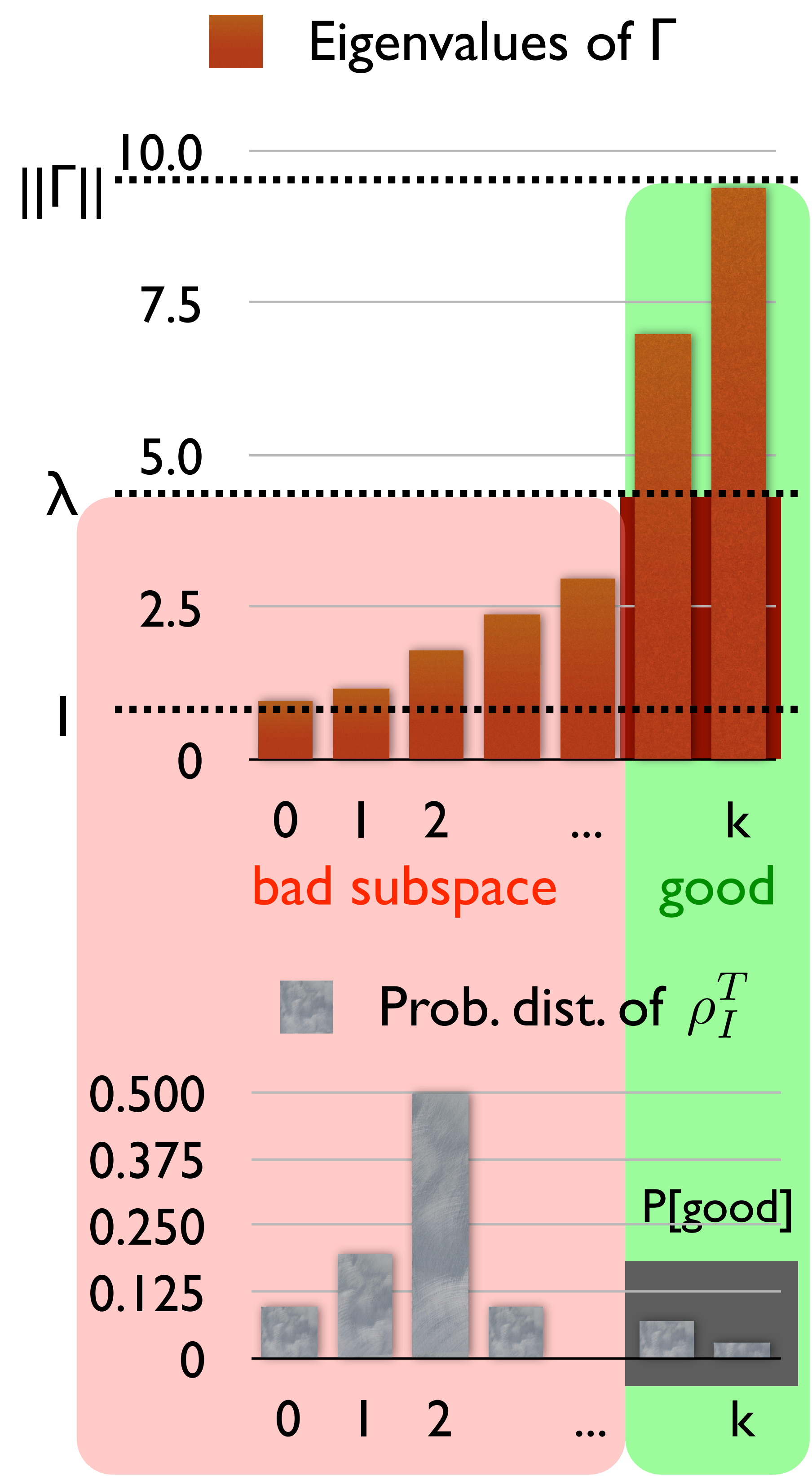
In the bad subspace, the success probability is at most η, in the good subspace it is at most 1. By **[Bernstein & Vazirani '93]**, A can succeed w.p. at most

$$\eta + 4\sqrt{P[\text{good}]}$$



Evolution of the progress function

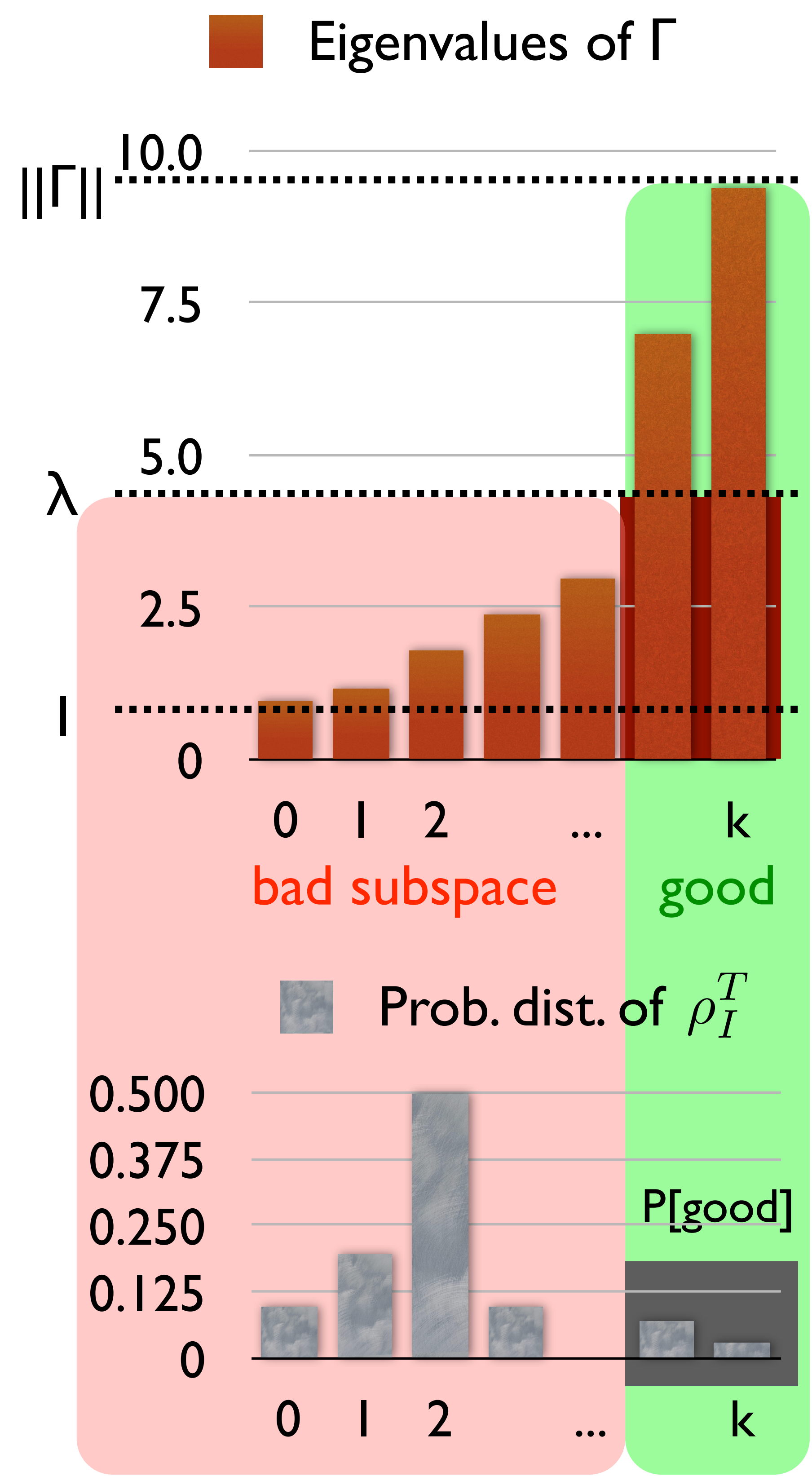
- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)
- We run A on input δ with $\Gamma\delta=\delta$. Then:
  1. $W^0=1$
  2. each $W^{t+1}/W^t \leq \max_i ||O_i \Gamma O_i \Gamma^{-1}||$
  3. $W^T \geq \lambda \zeta^2/16$
- **Proof:** **q.e.d.**



Evolution of the progress function

- Consider algorithm A running in time T, computing function f with success probability at least $\eta+\zeta$, and multiplicative adversary (Γ,λ)
- We run A on input δ with $\Gamma\delta=\delta$. Then:
  1. $W^0=1$
  2. each $W^{t+1}/W^t \le \max_i \|O_i\Gamma O_i\Gamma^{-1}\|$
  3. $W^T \ge \lambda\zeta^2/16$
- **Proof:** **q.e.d.**
- We get lower bound $T \ge \mathrm{MAdv}_{\eta,\zeta}(f)$ with

$$\mathrm{MAdv}_{\eta,\zeta}(f) = \max_{(\Gamma,\lambda)} \frac{\log(\lambda\zeta^2/16)}{\log(\max_i \|O_i\Gamma O_i\Gamma^{-1}\|)}$$



Block-diagonalization of Γ and O_i

Block-diagonalization of Γ and O_i

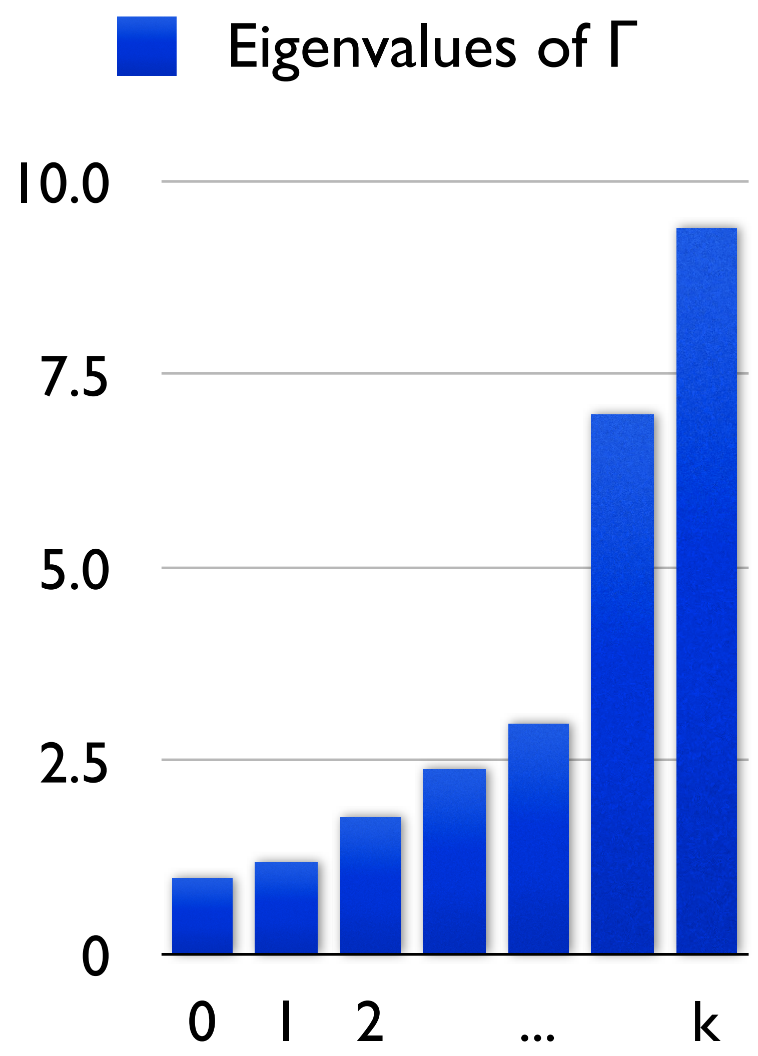
- How to efficiently upper-bound $||O_i \Gamma O_i \cdot \Gamma^{-1}||$?

Block-diagonalization of Γ and O_i

- How to efficiently upper-bound $||O_i \Gamma O_i \cdot \Gamma^{-1}||$?
- The eigenspaces of the conjugated $O_i \Gamma O_i$ overlap different eigenspaces of Γ, and we want them to cancel as much as possible so that the norm above is small

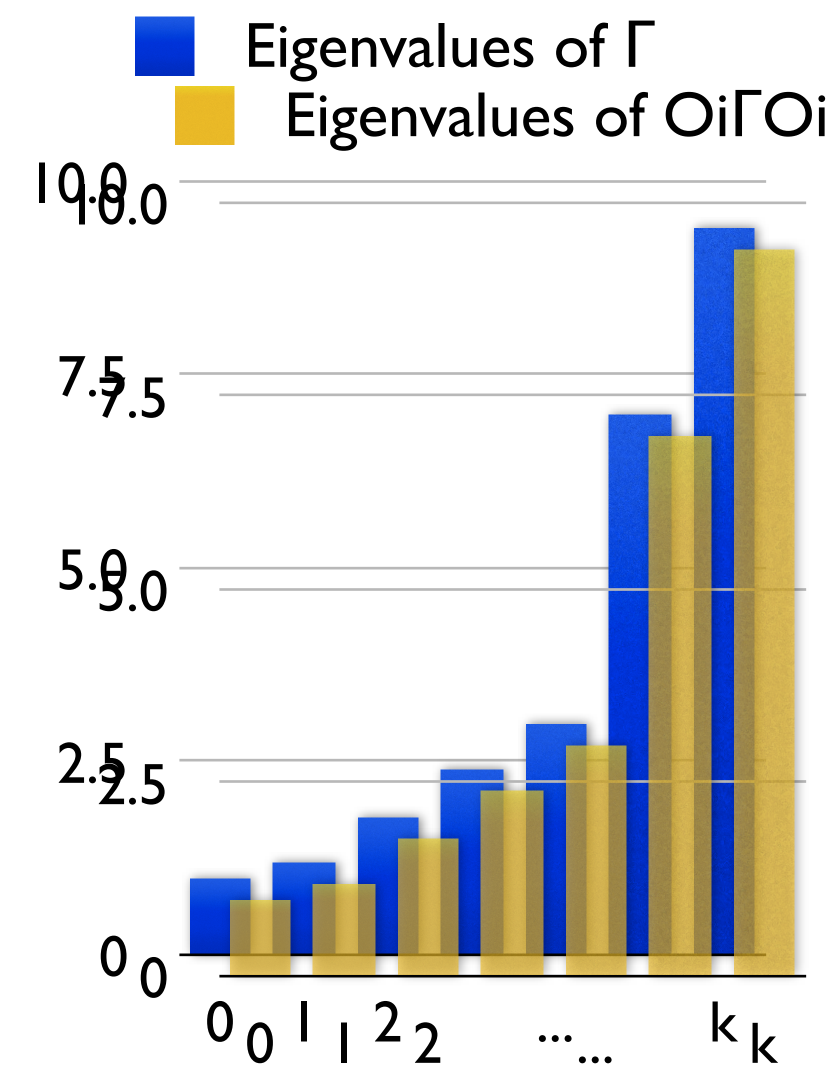
Block-diagonalization of Γ and O_i

- How to efficiently upper-bound $||O_i \Gamma O_i \cdot \Gamma^{-1}||$?
- The eigenspaces of the conjugated $O_i \Gamma O_i$ overlap different eigenspaces of Γ, and we want them to cancel as much as possible so that the norm above is small



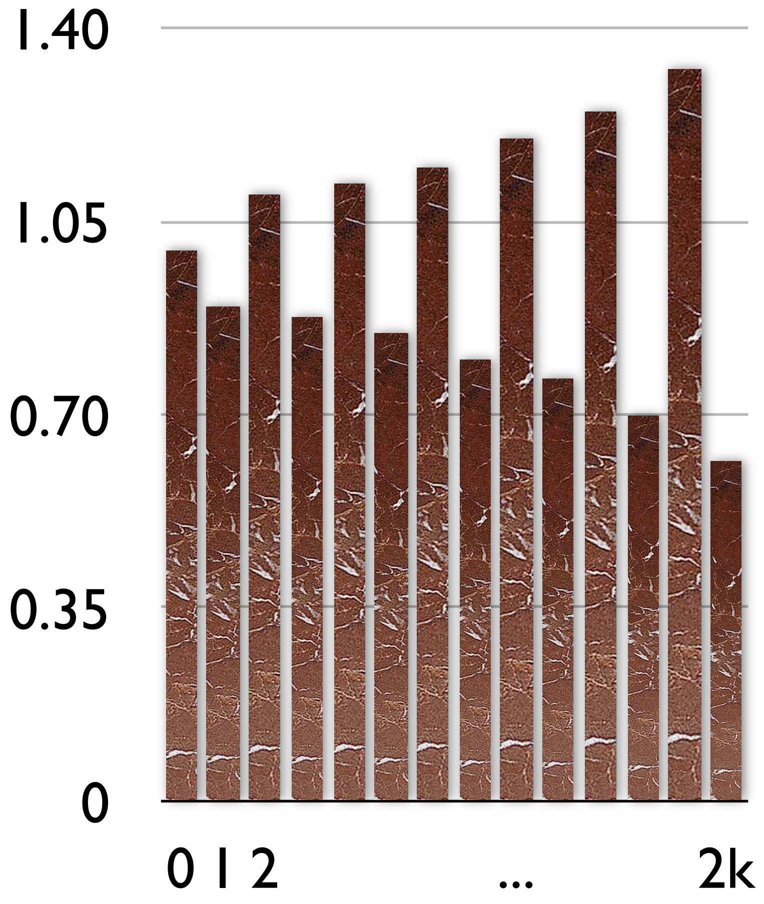
Block-diagonalization of Γ and O_i

- How to efficiently upper-bound $||O_i \Gamma O_i \cdot \Gamma^{-1}||$?
- The eigenspaces of the conjugated $O_i \Gamma O_i$ overlap different eigenspaces of Γ, and we want them to cancel as much as possible so that the norm above is small



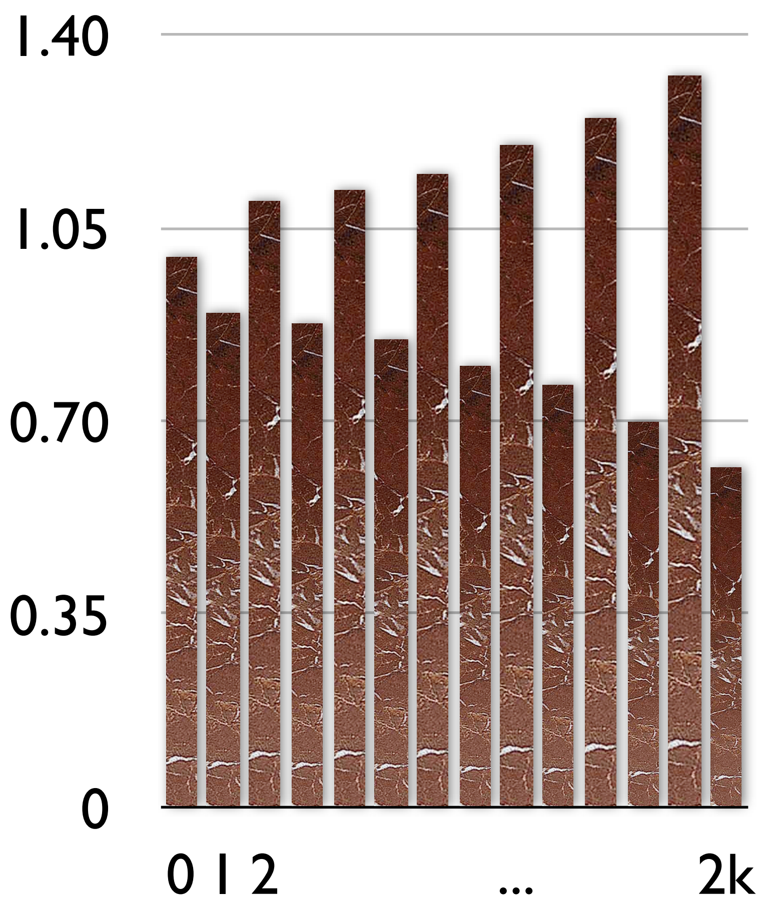
Block-diagonalization of Γ and O_i

- How to efficiently upper-bound $||O_i \Gamma O_i \cdot \Gamma^{-1}||$?
- The eigenspaces of the conjugated $O_i \Gamma O_i$ overlap different eigenspaces of Γ, and we want them to cancel as much as possible so that the norm above is small
  - like here...



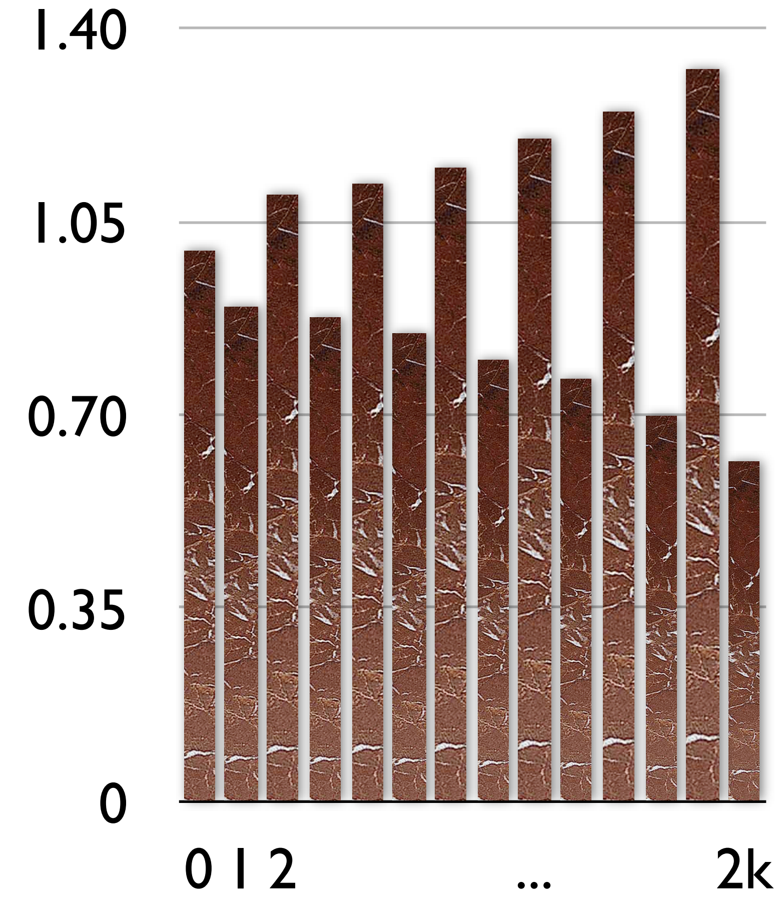
Block-diagonalization of Γ and O_i

- How to efficiently upper-bound $||O_i\Gamma O_i \cdot \Gamma^{-1}||$?
- The eigenspaces of the conjugated $O_i\Gamma O_i$ overlap different eigenspaces of Γ, and we want them to cancel as much as possible so that the norm above is small
  - like here...
  - we still need the condition on the bad subspace

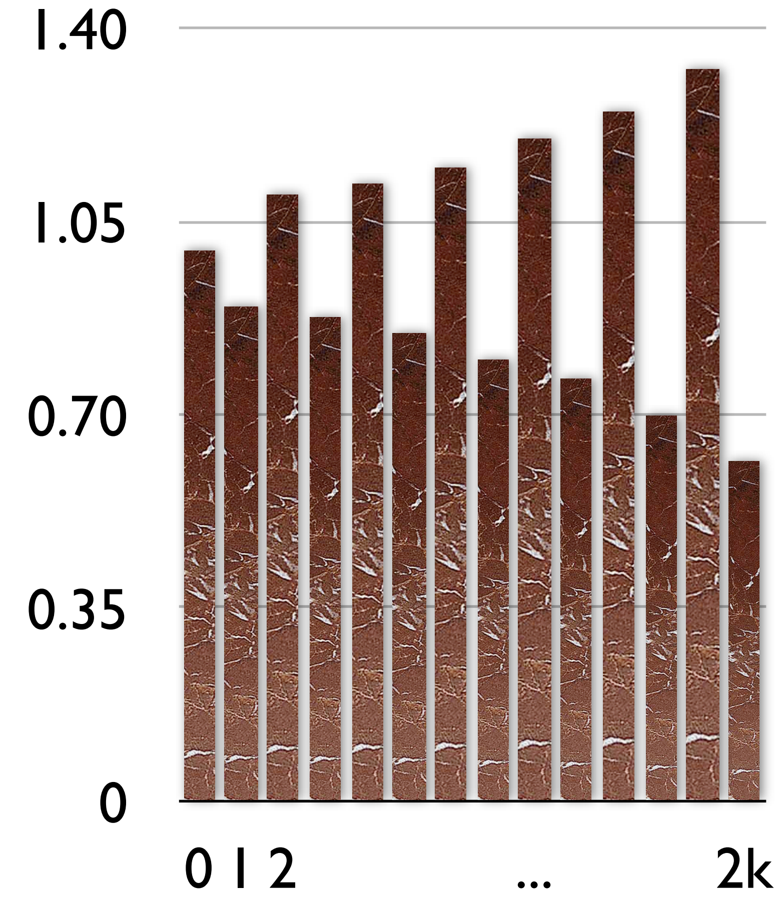


Block-diagonalization of Γ and O_i

- How to efficiently upper-bound $||O_i \Gamma O_i \cdot \Gamma^{-1}||$?
- The eigenspaces of the conjugated $O_i \Gamma O_i$ overlap different eigenspaces of Γ, and we want them to cancel as much as possible so that the norm above is small
  - like here...
  - we still need the condition on the bad subspace
- This makes the multiplicative adversary matrices hard to design

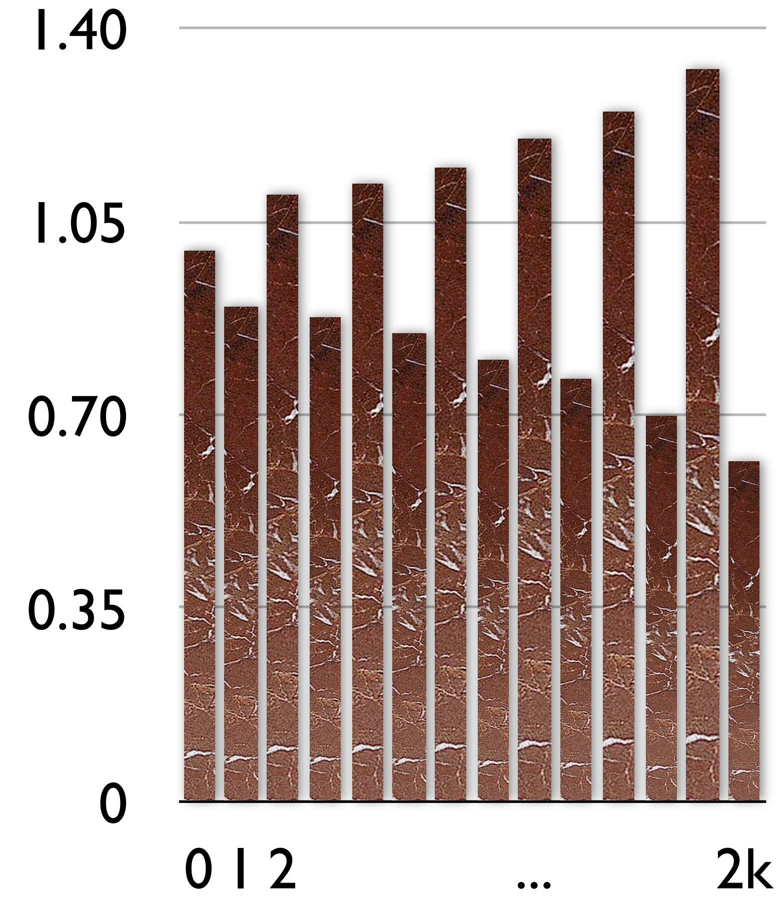


Block-diagonalization of Γ and O_i



Block-diagonalization of Γ and O_i

- By block-diagonalizing Γ and O_i together, we can bound each block separately

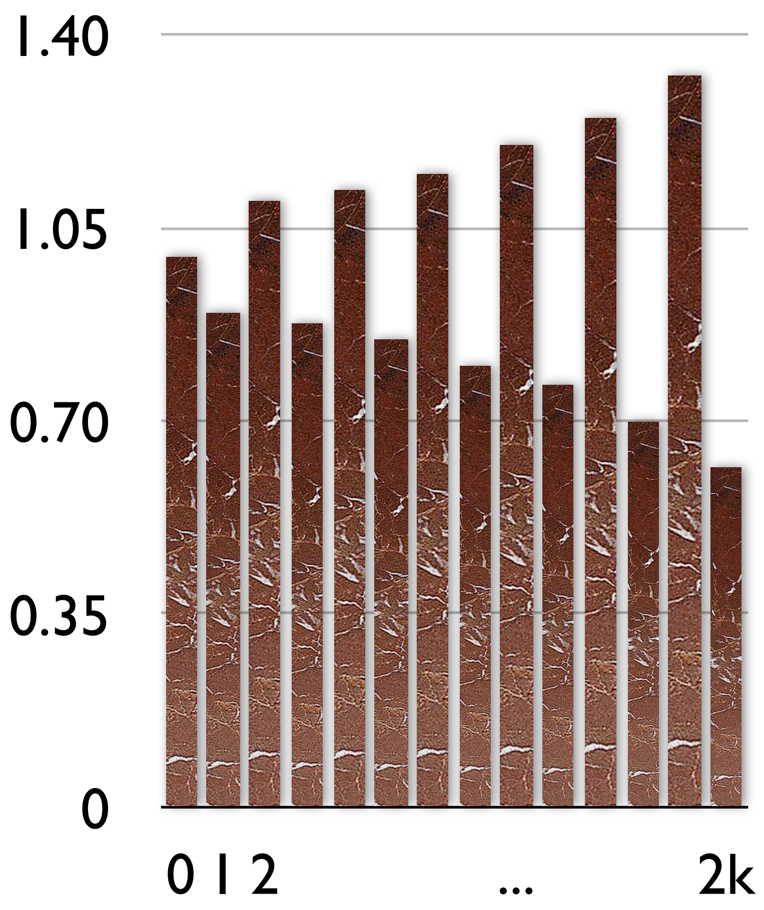


Block-diagonalization of Γ and O_i

- By block-diagonalizing Γ and O_i together, we can bound each block separately
- Since the eigenvalues in one block don't differ so much like in the whole matrix, we can use some bounds, such as

$$\lambda_{min}(M) \leq \lambda \leq ||M||,$$

and don't lose too much



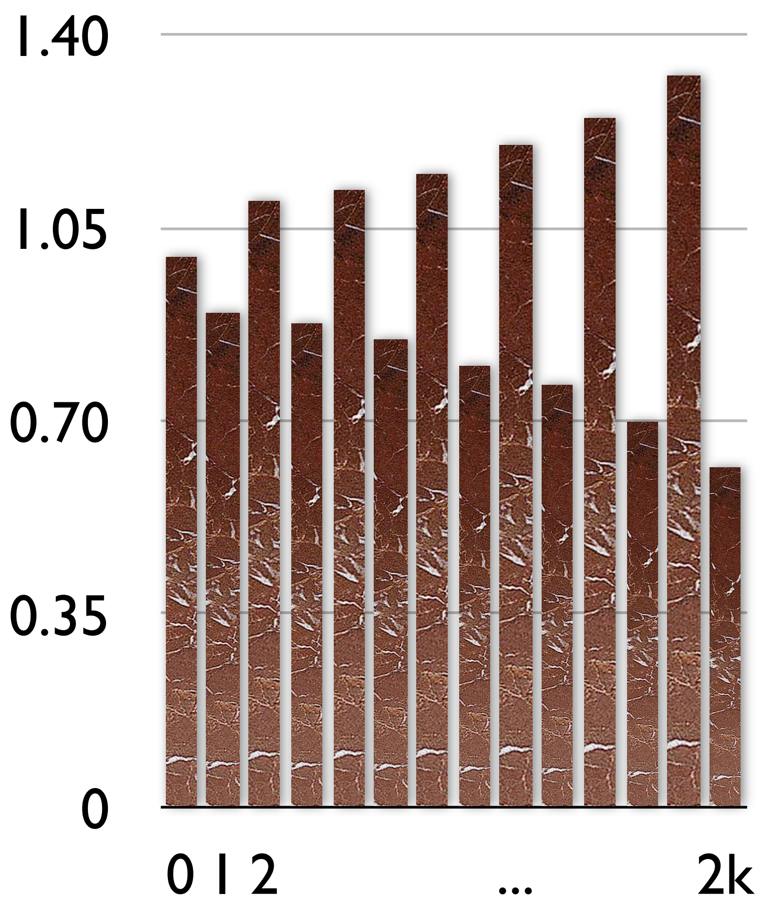
Block-diagonalization of Γ and O_i

- By block-diagonalizing Γ and O_i together, we can bound each block separately
- Since the eigenvalues in one block don't differ so much like in the whole matrix, we can use some bounds, such as

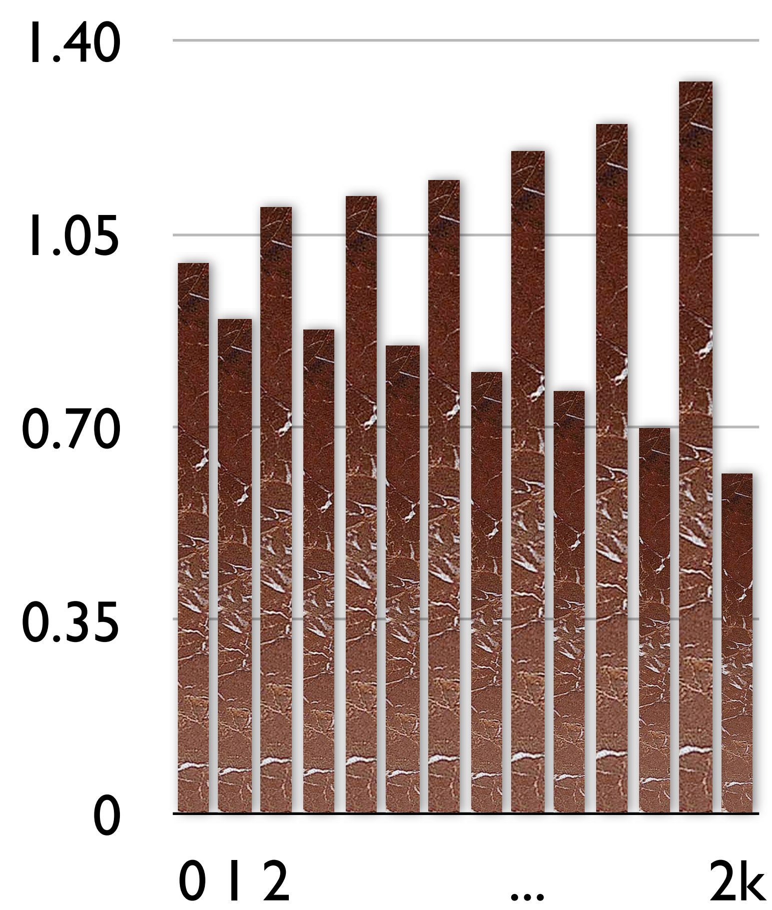
  $$\lambda_{min}(M) \leq \lambda \leq ||M||,$$

  and don't lose too much
- This gives the bound

$$\|O_i \Gamma O_i \cdot \Gamma^{-1}\| \leq 1 + 2 \max_k \frac{\|\Gamma_i^{(k)}\|}{\lambda_{\min}(\Gamma^{(k)})}$$



Block-diagonalization of Γ and O_i



- By block-diagonalizing Γ and O_i together, we can bound each block separately
- Since the eigenvalues in one block don't differ so much like in the whole matrix, we can use some bounds, such as

  $$\lambda_{min}(M) \leq \lambda \leq ||M||,$$

  and don't lose too much
- This gives the bound

$$\|O_i \Gamma O_i \cdot \Gamma^{-1}\| \leq 1 + 2 \max_k \frac{\|\Gamma_i^{(k)}\|}{\lambda_{\min}(\Gamma^{(k)})}$$

$\Gamma^{(k)}$ is the k-th block on the diagonal

Block-diagonalization of Γ and O_i

- By block-diagonalizing Γ and O_i together, we can bound each block separately
- Since the eigenvalues in one block don't differ so much like in the whole matrix, we can use some bounds, such as

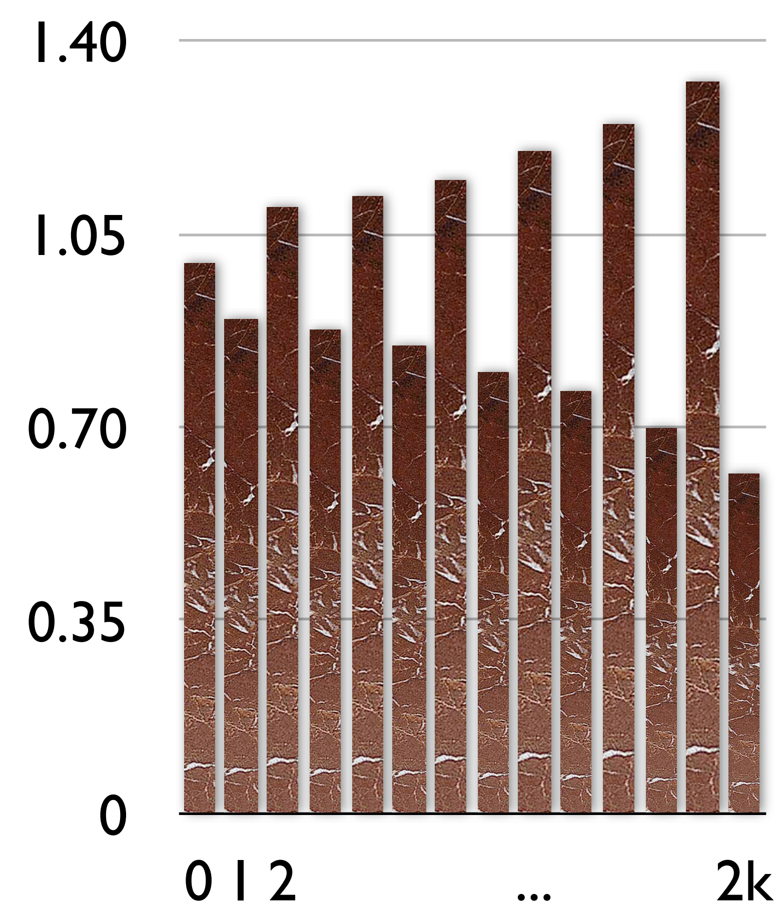
$$\lambda_{min}(M) \leq \lambda \leq ||M||,$$

and don't lose so much

- This gives the bound

sub-matrix of $\Gamma^{(k)}$ with zeroes when $x_i = y_i$

$$\|O_i \Gamma O_i \cdot \Gamma^{-1}\| \leq 1 + 2 \max_k \frac{\|\Gamma_i^{(k)}\|}{\lambda_{\min}(\Gamma^{(k)})}$$



Block-diagonalization of Γ and O_i

$$\mathrm{MAdv}_{\eta,\zeta}(f) \geq \max_{\Gamma,\lambda} \log(\tfrac{1}{16}\zeta^2\lambda) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|}$$

Block-diagonalization of Γ and O_i

- The final multiplicative adversary bound is

$$\mathrm{MAdv}_{\eta,\zeta}(f) \geq \max_{\Gamma,\lambda} \log\left(\tfrac{1}{16}\zeta^2\lambda\right) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|}$$

Block-diagonalization of Γ and O_i

you pick the success probability η of a random choice, and additional success ζ

- The final multiplicative adversary bound is

$$\mathrm{MAdv}_{\eta,\zeta}(f) \geq \max_{\Gamma,\lambda} \log\left(\tfrac{1}{16}\zeta^2\lambda\right) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|}$$

Block-diagonalization of Γ and O_i

maximize over all multiplicative adversaries

- The final multiplicative adversary bound is

$$\mathrm{MAdv}_{\eta,\zeta}(f) \geq \max_{\Gamma,\lambda} \log(\tfrac{1}{16}\zeta^2\lambda) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|}$$

Block-diagonalization of Γ and O_i

λ is proportional to ||Γ|| and it has to cancel ζ^2

- The final multiplicative adversary bound is

$$\mathrm{MAdv}_{\eta,\zeta}(f) \geq \max_{\Gamma,\lambda} \log\left(\tfrac{1}{16}\zeta^2\lambda\right) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|}$$

Block-diagonalization of Γ and O$_i$

minimize over input bits i=1,...,n
and blocks on the diagonal

- The final multiplicative adversary bound is

$$\mathrm{MAdv}_{\eta,\zeta}(f) \geq \max_{\Gamma,\lambda} \log\left(\tfrac{1}{16}\zeta^2\lambda\right) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|}$$

Block-diagonalization of Γ and O_i

- The final multiplicative adversary bound is

$$\mathrm{MAdv}_{\eta,\zeta}(f) \geq \max_{\Gamma,\lambda} \log\left(\tfrac{1}{16}\zeta^2\lambda\right) \cdot \min_{i,k} \frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|}$$

- You don't have to use the *finest* block-diagonalization.

  Any is good, including using the whole space as one block, but then the obtained lower bound need not be very strong.

Example: Lower bound for search

Example: Lower bound for search

- Given an n-bit string with exactly one 1. **Task:** find it.

Example: Lower bound for search

- Given an n-bit string with exactly one 1. **Task:** find it.

  $\mathrm{MAdv}_{1/n,\zeta}(\mathrm{Search}_n) = \Omega(\zeta^2\sqrt{n})$

Example: Lower bound for search

- Given an n-bit string with exactly one 1. **Task:** find it.

  $\mathrm{MAdv}_{1/n,\zeta}(\mathrm{Search}_n) = \Omega(\zeta^2\sqrt{n})$

- Define v=(1,...,1) of length n and v_i=(1,...,1, 1-n, 1,...,1), normalized to length 1. Note that $v \perp v_i$.

Example: Lower bound for search

- Given an n-bit string with exactly one 1. **Task:** find it.

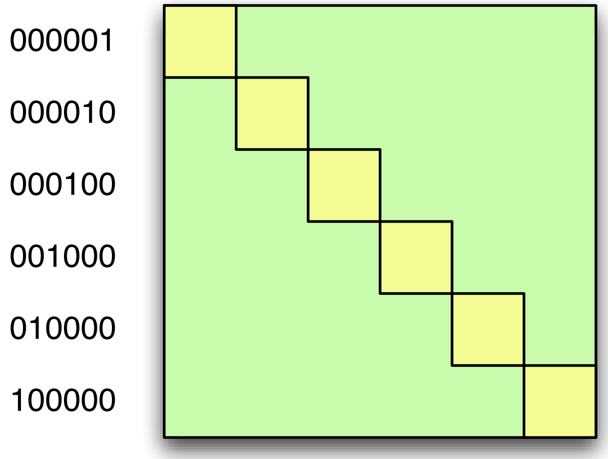
  $\mathrm{MAdv}_{1/n,\zeta}(\mathrm{Search}_n) = \Omega(\zeta^2\sqrt{n})$

- Define $v=(1,...,1)$ of length n and $v_i=(1,...,1,\ 1-n,\ 1,...,1)$, normalized to length 1. Note that $v \perp v_i$.

- Let $\Gamma = (1-q)|v\rangle\langle v| + qI$

  $\Gamma v = v$ and $\Gamma v_i = q\, v_i$, i.e. v and v_i are eigenvectors.

  Let $\lambda = ||\Gamma|| = q = 32/\zeta^2$.



Example: Lower bound for search

- Given an n-bit string with exactly one 1. **Task:** find it.

  $\mathrm{MAdv}_{1/n,\zeta}(\mathrm{Search}_n) = \Omega(\zeta^2\sqrt{n})$

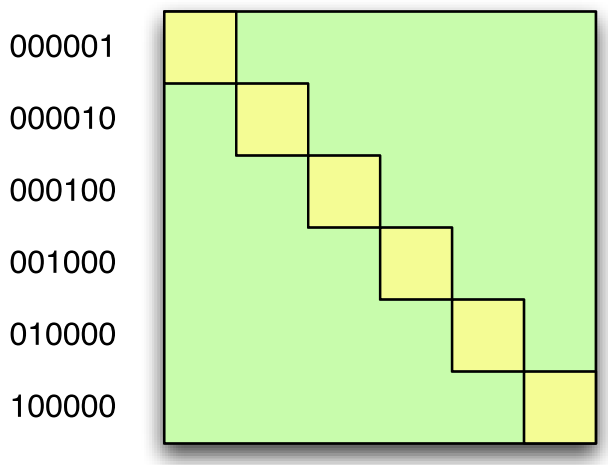
- Define $v=(1,\ldots,1)$ of length n and $v_i=(1,\ldots,1,\ 1-n,\ 1,\ldots,1)$, normalized to length 1. Note that $v \perp v_i$.

- Let $\Gamma = (1-q)|v\rangle\langle v| + qI$

  $\Gamma v = v$ and $\Gamma v_i = q\, v_i$, i.e.
  v and v_i are eigenvectors.

  Let $\lambda = ||\Gamma|| = q = 32/\zeta^2$.

- The success probability in the bad subspace (containing v) is $\eta = 1/n$.



Example: Lower bound for search

- Given an n-bit string with exactly one 1. **Task:** find it.

  $\mathrm{MAdv}_{1/n,\zeta}(\mathrm{Search}_n) = \Omega(\zeta^2\sqrt{n})$

- Define $v=(1,...,1)$ of length n and $v_i=(1,...,1, 1-n, 1,...,1)$, normalized to length 1. Note that $v \perp v_i$.

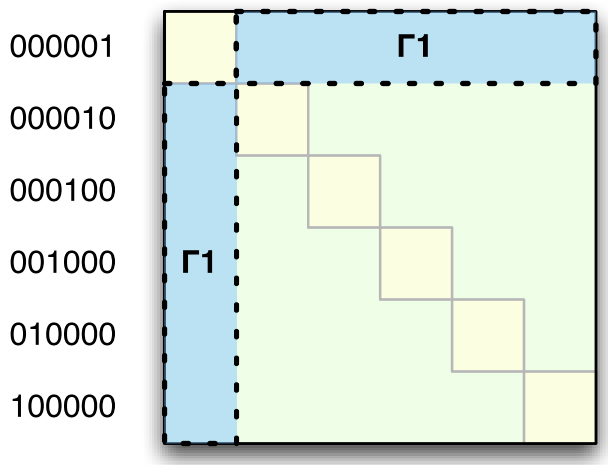
- Let $\Gamma = (1-q)|v\rangle\langle v| + qI$

  $\Gamma v = v$ and $\Gamma v_i = q\, v_i$, i.e. v and v_i are eigenvectors.

  Let $\lambda=\|\Gamma\|= q = 32/\zeta^2$.

- The success probability in the bad subspace (containing v) is $\eta=1/n$.

- Use just one block. Then $\lambda_{min}(\Gamma) = 1$ and $\|\Gamma_i\|<q/\sqrt{n}$.



Example: Lower bound for search

- Given an n-bit string with exactly one 1. **Task:** find it.

  $\mathrm{MAdv}_{1/n,\zeta}(\mathrm{Search}_n) = \Omega(\zeta^2\sqrt{n})$

- Define $v=(1,\ldots,1)$ of length n and $v_i=(1,\ldots,1, 1-n, 1,\ldots,1)$, normalized to length 1. Note that $v \perp v_i$.

- Let $\Gamma = (1-q)|v\rangle\langle v| + qI$
  $\Gamma v = v$ and $\Gamma v_i = q\, v_i$, i.e. v and v_i are eigenvectors.

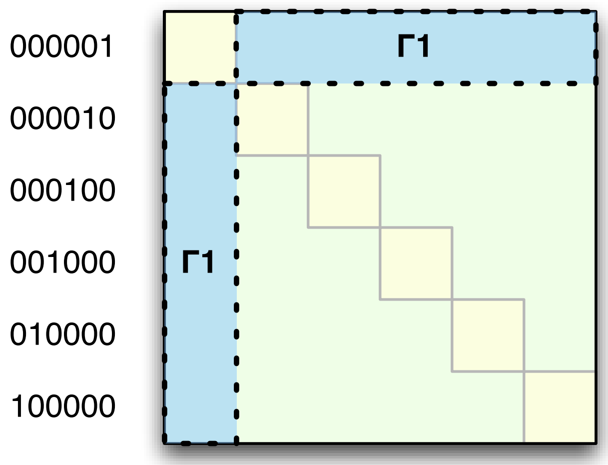
  Let $\lambda=\|\Gamma\|= q = 32/\zeta^2$.

- The success probability in the bad subspace (containing v) is $\eta=1/n$.

- Use just one block. Then $\lambda_{\min}(\Gamma) = 1$ and $\|\Gamma_i\|<q/\sqrt{n}$.

- The final bound is

$$\log(\tfrac{1}{16}\zeta^2\lambda)\cdot\min_{i,k}\frac{\lambda_{\min}(\Gamma^{(k)})}{2\|\Gamma_i^{(k)}\|} > \frac{\log 2}{64}\zeta^2\sqrt{n}$$



Lower bound for k-search

Lower bound for k-search

- Given an n-bit string with k ones. **Task:** find them.

Lower bound for k-search

- Given an n-bit string with k ones. **Task:** find them.
- $\mathrm{MAdv}_{\exp(-O(k)),\exp(-O(k))}(\mathrm{Search}_{k,n}) = \Omega(\sqrt{(kn)})$

Lower bound for k-search

- Given an n-bit string with k ones. **Task:** find them.
- $\mathrm{MAdv}_{\exp(-O(k)),\exp(-O(k))}(\mathrm{Search}_{k,n}) = \Omega(\sqrt{(kn)})$
  - The multiplicative adversary matrix Γ is a *combinatorial matrix*, whose entries $\Gamma_{x,y}$ only depends on $|x \cap y|$.

Lower bound for k-search

- Given an n-bit string with k ones. **Task:** find them.
- $\mathrm{MAdv}_{\exp(-O(k)),\exp(-O(k))}(\mathrm{Search}_{k,n}) = \Omega(\sqrt{(kn)})$
  - The multiplicative adversary matrix Γ is a *combinatorial matrix*, whose entries $\Gamma_{x,y}$ only depends on $|x \cap y|$.
  - The k+1 eigenspaces can be indexed by "knowledge", i.e. how many ones has the algorithm already found, with eigenvectors being superpositions of all strings consistent with some pattern of ones.

Lower bound for k-search

- Given an n-bit string with k ones. **Task:** find them.
- $\mathrm{MAdv}_{\exp(-O(k)),\exp(-O(k))}(\mathrm{Search}_{k,n}) = \Omega(\sqrt{(kn)})$
    - The multiplicative adversary matrix Γ is a *combinatorial matrix*, whose entries $\Gamma_{x,y}$ only depends on $|x \cap y|$.
    - The k+1 eigenspaces can be indexed by "knowledge", i.e. how many ones has the algorithm already found, with eigenvectors being superpositions of all strings consistent with some pattern of ones.
    - Tedious combinatorial calculation done by [Ambainis '05] and we can reuse it

Lower bound for k-search

- Given an n-bit string with k ones. **Task:** find them.
- $MAdv_{\exp(-O(k)),\exp(-O(k))}(Search_{k,n}) = \Omega(\sqrt{(kn)})$
  - The multiplicative adversary matrix Γ is a *combinatorial matrix*, whose entries $\Gamma_{x,y}$ only depends on $|x \cap y|$.
  - The k+1 eigenspaces can be indexed by "knowledge", i.e. how many ones has the algorithm already found, with eigenvectors being superpositions of all strings consistent with some pattern of ones.
  - Tedious combinatorial calculation done by [Ambainis '05] and we can reuse it
- One can use $\Gamma \approx \Delta^{-k}$, where Δ is the **additive** adversary matrix (much simpler). Don't know any other example where this holds.

Open: element distinctness

- Given n number. **Task:** are they distinct?
- The quantum query complexity is known to be $\theta(n^{2/3})$ **[Ambainis '04, Aaronson & Shi '04]**, where the lower bound is proved using the polynomial method.
- Having an adversary bound of either type would make the bound composable and give bounds for other functions.
- Can one use the structure of the *automorphism group* of the function to design the structure of the eigenspaces?

Direct product theorem

- The multiplicative adversary bound satisfies an unconditional *strong direct product theorem*:

$$\mathrm{MAdv}_{\eta^{\Omega(k)},\zeta^{\Omega(k)}}(f^{(k)}) = \Omega(k \cdot \mathrm{MAdv}_{\eta,\zeta}(f))$$

- **Proof:** take the tensor power $\Gamma^{\otimes k}$ and $\lambda^{k/10}$. Both η and ζ go down exponentially.
- For Search and the OR function our calculations are simple, hence we get a new and elementary proof of the *time-space tradeoffs* for matrix-vector multiplication and sorting from **[Klauck, Š. & de Wolf '04]**.
- Maybe our method is so hard to use precisely because it gives a free SDPT, which is usually very hard to prove.

Summary

- New variant of the adversary bound
- Suitable for exponentially small success probabilities
- Satisfies strong direct product theorem