# Instance Compression and Succint PCP's for NP

Sivaramakrishnan.N.R.

March 31, 2012

# Outline


## Basics of Parameterized Complexity

- What are we looking for?
- Effecient Computation
- The Vertex Cover Problem

## Instance Compression

- Introduction
- Definitions
- W-Reductions
- Infeasibility of Deterministic Compression
- Probabilistic Compression

## Succint PCPs

# What are we looking for ?

- **Classical Complexity Theory** - the running time as a function $T(n)$ of the input size $n$

# What are we looking for ?

- **Classical Complexity Theory** - the running time as a function $T(n)$ of the input size $n$
- Instead as a function $T(n, k)$ of the input size $n$ and a parameter $k$.

# What are we looking for ?

- **Classical Complexity Theory** - the running time as a function $T(n)$ of the input size $n$
- Instead as a function $T(n, k)$ of the input size $n$ and a parameter $k$.

## Definition

A parameterization of a decision problem is a function that assigns a parameter $k$ to each input instance $x$.
Any parametric problem is a subset of
$\{< x, 1^k > | x \in \{0,1\}^*, k \in \mathbb{N}\}$.

# The class FPT

- We now look at an idea of efficient computation in the parameterized world.

# The class FPT

- We now look at an idea of efficient computation in the parameterized world.

- Definition
  A parameterized problem is **fixed parameter tractable(FPT)** if there is a $f(k)n^c$ time algorithm for some constant $c$.

# The class FPT

- We now look at an idea of efficient computation in the parameterized world.

- Definition

  A parameterized problem is **fixed parameter tractable(FPT)** if there is a $f(k)n^c$ time algorithm for some constant $c$.

  - This indicates that we are looking for 'efficient' algorithms when the size of $k$ is small.

# The vertex Cover Problem

1. Every edge $\{u, v\}$, $u$ *or* $v$ has to be in the cover.

# The vertex Cover Problem

1. Every edge $\{u, v\}$, $u$ *or* $v$ has to be in the cover.
2. Branch on $u$ or $v (u, v \in E)$, delete the respective vertex with its incident edges and proceed recursively.

# The vertex Cover Problem

1. Every edge $\{u, v\}$, $u$ *or* $v$ has to be in the cover.
2. Branch on $u$ or $v(u, v \in E)$, delete the respective vertex with its incident edges and proceed recursively.
3. The depth of the search tree is atmost $k$ and has atmost $2^k$ nodes, and processing time at each node being atmost quadratic yields a running time of $2^k.n^2$

# The vertex Cover Problem

1. Every edge $\{u, v\}$, $u$ *or* $v$ has to be in the cover.
2. Branch on $u$ or $v(u, v \in E)$, delete the respective vertex with its incident edges and proceed recursively.
3. The depth of the search tree is atmost $k$ and has atmost $2^k$ nodes, and processing time at each node being atmost quadratic yields a running time of $2^k.n^2$

# Preprocessing the input

Now we shall do some preprocessing.

# Preprocessing the input

Now we shall do some preprocessing.

1. Any vertex $v$ with degree $k+1$ or more, has atleast $k+1$ edges with one end point being $v$.

# Preprocessing the input

Now we shall do some preprocessing.

1. Any vertex $v$ with degree $k+1$ or more, has atleast $k+1$ edges with one end point being $v$.
2. To cover all these edges, we either include $v$ in the cover or all its neighbours in the cover.

# Preprocessing the input

Now we shall do some preprocessing.

1. Any vertex $v$ with degree $k+1$ or more, has atleast $k+1$ edges with one end point being $v$.
2. To cover all these edges, we either include $v$ in the cover or all its neighbours in the cover.
3. When we include all it neighbours, the size of the cover exceeds $k$ and hence $v$ has to be included in the cover.

# Preprocessing the input

Now we shall do some preprocessing.

1. Any vertex $v$ with degree $k+1$ or more, has atleast $k+1$ edges with one end point being $v$.
2. To cover all these edges, we either include $v$ in the cover or all its neighbours in the cover.
3. When we include all it neighbours, the size of the cover exceeds $k$ and hence $v$ has to be included in the cover.
4. Remove $v$ and its incident edges from the graph and look for a cover of size $k-1$ in $G-v$.

# Preprocessing the input

Now we shall do some preprocessing.

1. Any vertex $v$ with degree $k+1$ or more, has atleast $k+1$ edges with one end point being $v$.
2. To cover all these edges, we either include $v$ in the cover or all its neighbours in the cover.
3. When we include all it neighbours, the size of the cover exceeds $k$ and hence $v$ has to be included in the cover.
4. Remove $v$ and its incident edges from the graph and look for a cover of size $k-1$ in $G-v$.
5. After doing the preprocessing, the resulting graph $G'$ has atmost $k^2$ edges and $k^2+k$ vertices.

# Preprocessing the input

Now we shall do some preprocessing.

1. Any vertex $v$ with degree $k+1$ or more, has atleast $k+1$ edges with one end point being $v$.
2. To cover all these edges, we either include $v$ in the cover or all its neighbours in the cover.
3. When we include all it neighbours, the size of the cover exceeds $k$ and hence $v$ has to be included in the cover.
4. Remove $v$ and its incident edges from the graph and look for a cover of size $k-1$ in $G-v$.
5. After doing the preprocessing, the resulting graph $G'$ has atmost $k^2$ edges and $k^2+k$ vertices.

# Instance Compression

- The notion of instance compressibility for $NP$ problems - related to parameterized complexity(Kernalization).

# Instance Compression

- The notion of instance compressibility for $NP$ problems - related to parameterized complexity(Kernalization).
- Consider a language $L$ and we wish to test the membership of $x \in L$.

# Instance Compression

- The notion of instance compressibility for $NP$ problems - related to parameterized complexity(Kernalization).
- Consider a language $L$ and we wish to test the membership of $x \in L$.
- Try to find a function $f$ such that $f(x)$ is an instance of $L$ and $|f(x)| < |x|$.

# Instance Compression

- The notion of instance compressibility for $NP$ problems - related to parameterized complexity(Kernalization).
- Consider a language $L$ and we wish to test the membership of $x \in L$.
- Try to find a function $f$ such that $f(x)$ is an instance of $L$ and $|f(x)| < |x|$.
- Applying it iteratively until the instance size is reduced to a constant.

# Instance Compression

- The notion of instance compressibility for $NP$ problems - related to parameterized complexity(Kernalization).
- Consider a language $L$ and we wish to test the membership of $x \in L$.
- Try to find a function $f$ such that $f(x)$ is an instance of $L$ and $|f(x)| < |x|$.
- Applying it iteratively until the instance size is reduced to a constant.

**Questions:**

- For which $NP-$Complete $L$ is there a polynomial-time computable compression function $f$ such that $|f(x)| < |x|$ for all $x$?

**Questions:**

- For which $NP-$Complete $L$ is there a polynomial-time computable compression function $f$ such that $|f(x)| < |x|$ for all $x$?
  It is unlikely, say to a length sub-polynomial in $|x|$.

**Questions:**

- For which $NP-$Complete $L$ is there a polynomial-time computable compression function $f$ such that $|f(x)| < |x|$ for all $x$?
  It is unlikely, say to a length sub-polynomial in $|x|$.
- For which $NP-$Complete $L$ is there a polynomial-time compression function $f$ which compresses to size polynomial in the witness size ?

**Questions:**

- For which $NP-$Complete $L$ is there a polynomial-time computable compression function $f$ such that $|f(x)| < |x|$ for all $x$?
  It is unlikely, say to a length sub-polynomial in $|x|$.
- For which $NP-$Complete $L$ is there a polynomial-time compression function $f$ which compresses to size polynomial in the witness size ?
  A conceptual view of "partial solvability". A more reasonable question to ask.

# Definitions

## Definition

Let $L$ be a parametric problem and $A \subseteq \{0,1\}^*$. $L$ is said to be compressibel within $A$ if there is a polynomial $p$ and a polynomial-time computable function $f$, such that for each $x \in \{0,1\}^*$ and $n \in \mathbb{N}$, $|f(< x, 1^n >)| \leq p(n)$ and $< x, 1^n > \in L$ *iff* $f(< x, 1^n >) \in A$. $L$ is compressible if there is some $A$ for which $L$ is compressible within $A$. $L$ is self-compressible if $L$ is compressible within $L$.

### Definition (Non-uniform Compression)

A parametric problem $L$ is said to be compressible with advice $s$, if the compression function is computable in deterministic polynomial time when given access to an advice string of size $s(|x|, n)$. $L$ is non-uniformly compressible if $s$ is polynomially bounded in $m$ and $n$.

We shall now define some parametric problems in .

## Definition

$SAT = \{< \phi, 1^n > | \phi$ *is a satisfiable formula*,
and n is atleast the number of variables in $\phi\}$

We shall now define some parametric problems in .

## Definition

$SAT = \{< \phi, 1^n > | \phi$ *is a satisfiable formula*,
and n is atleast the number of variables in $\phi\}$

## Definition

$VC = \{< G, 1^{klog(m)} > | G$ *has a vertex cover of size atmost* $k\}$

We shall now define some parametric problems in .

**Definition**

$SAT = \{< \phi, 1^n > | \phi$ *is a satisfiable formula*,
and n is atleast the number of variables in $\phi\}$

**Definition**

$VC = \{< G, 1^{klog(m)} > | G$ *has a vertex cover of size atmost* $k\}$

**Definition**

$OR - SAT = \{< \phi_i, 1^n >$
$|$*At least one* $\phi_i$ *is satisfiable, and each* $\phi_i$ *has size atmost* $n\}$.

# W-Reductions

Given parametric problems $L_1$ and $L_2$, $L_1$ $W-$reduces to $L_2$(denoted $L_1 \leq_W L_2$) if there is a polynomial-time computable function $f$ and polynomials $p_1$ and $p_2$ such that:

1. $f(< x, 1^{n_1} >)$ is of the form $< y, n_2 >$ where $|y| \leq p_1(n_1 + |x|)$ and $n_2 \leq p_2(n_1)$.
2. $f(< x, 1^{n_1} >) \in L_2$ *iff* $< x, 1^{n_1} > \in L_1$.

# W-Reductions

Given parametric problems $L_1$ and $L_2$, $L_1$ $W-$reduces to $L_2$(denoted $L_1 \leq_W L_2$) if there is a polynomial-time computable function $f$ and polynomials $p_1$ and $p_2$ such that:

1. $f(< x, 1^{n_1} >)$ is of the form $< y, n_2 >$ where $|y| \leq p_1(n_1 + |x|)$ and $n_2 \leq p_2(n_1)$.
2. $f(< x, 1^{n_1} >) \in L_2$ *iff* $< x, 1^{n_1} > \in L_1$.

Why such a definition ?

**Propositions:**

1. If $L_1 \leq_W L_2$ and $L_2$ is compressible, then $L_1$ is compressible.

**Propositions:**

1. If $L_1 \leq_W L_2$ and $L_2$ is compressible, then $L_1$ is compressible.
2. $VC$ is self-compressible.

**Propositions:**

1. If $L_1 \leq_W L_2$ and $L_2$ is compressible, then $L_1$ is compressible.
2. $VC$ is self-compressible.
3. $OR - SAT \leq_W Clique \leq_W SAT$.

# Infeasibility of Deterministic Compression

**Theorem**

*If $OR - SAT$ is compressible, then $coNP \subseteq NP/Poly$, and hence PH collapses.*

# Infeasibility of Deterministic Compression

**Theorem**

*If $OR - SAT$ is compressible, then $coNP \subseteq NP/Poly$, and hence PH collapses.*

**Proof:**

- Size of the formula - $m$. Size of sub-formulae - atmost $n$.

# Infeasibility of Deterministic Compression

**Theorem**

*If $OR - SAT$ is compressible, then $coNP \subseteq NP/Poly$, and hence PH collapses.*

**Proof:**

- Size of the formula - $m$. Size of sub-formulae - atmost $n$.
- By the hypothesis there exist $A$ and $f$ computable in $poly(m)$such that
  1. $|f(\phi, 1^n)| \leq O(poly(n, log(m)))$,
  2. $\phi$ is satisfiable *iff* $f(\phi, 1^n) \in A$.

# Infeasibility of Deterministic Compression

**Theorem**

*If $OR - SAT$ is compressible, then $coNP \subseteq NP/Poly$, and hence PH collapses.*

**Proof:**

- Size of the formula - $m$. Size of sub-formulae - atmost $n$.
- By the hypothesis there exist $A$ and $f$ computable in $poly(m)$such that
  1. $|f(\phi, 1^n)| \leq O(poly(n, log(m)))$,
  2. $\phi$ is satisfiable *iff* $f(\phi, 1^n) \in A$.
- Size of compressed instance $k = (n + log(m))^c$.

- $S$ be the set of unsatisfiable formula of size atmost $n$.

- $S$ be the set of unsatisfiable formula of size atmost $n$.
- $T$ be the set of strings in $\bar{A}$ of length atmost $k$.

- $S$ be the set of unsatisfiable formula of size atmost $n$.
- $T$ be the set of strings in $\bar{A}$ of length atmost $k$.
- $f$ induces a map $g : S^{\frac{m}{n}} \rightarrow T$.

- $S$ be the set of unsatisfiable formula of size atmost $n$.
- $T$ be the set of strings in $\bar{A}$ of length atmost $k$.
- $f$ induces a map $g : S^{\frac{m}{n}} \rightarrow T$.
- Find a $poly(n)$ size set $C \subseteq T$, such that any formula in $S$ is contained in atleast one tuple that maps to a string in $C$ under $g$.

- $S$ be the set of unsatisfiable formula of size atmost $n$.
- $T$ be the set of strings in $\bar{A}$ of length atmost $k$.
- $f$ induces a map $g : S^{\frac{m}{n}} \rightarrow T$.
- Find a $poly(n)$ size set $C \subseteq T$, such that any formula in $S$ is contained in atleast one tuple that maps to a string in $C$ under $g$.
- If such a $C$ exist, then to decide $\phi \in S\bar{A}T$,
  1. Guess a tuple of $\frac{m}{n}$ formulae of size atmost $n$ with $\phi$ belonging to it.
  2. Check if it maps to a string in $C$.

- $S$ be the set of unsatisfiable formula of size atmost $n$.
- $T$ be the set of strings in $\bar{A}$ of length atmost $k$.
- $f$ induces a map $g : S^{\frac{m}{n}} \rightarrow T$.
- Find a $poly(n)$ size set $C \subseteq T$, such that any formula in $S$ is contained in atleast one tuple that maps to a string in $C$ under $g$.
- If such a $C$ exist, then to decide $\phi \in S\bar{A}T$,
  1. Guess a tuple of $\frac{m}{n}$ formulae of size atmost $n$ with $\phi$ belonging to it.
  2. Check if it maps to a string in $C$.
- If $m = poly(n)$ we get $coNP \subseteq NP/Poly$.

- $S$ be the set of unsatisfiable formula of size atmost $n$.
- $T$ be the set of strings in $\bar{A}$ of length atmost $k$.
- $f$ induces a map $g : S^{\frac{m}{n}} \rightarrow T$.
- Find a $poly(n)$ size set $C \subseteq T$, such that any formula in $S$ is contained in atleast one tuple that maps to a string in $C$ under $g$.
- If such a $C$ exist, then to decide $\phi \in S\bar{A}T$,
  1. Guess a tuple of $\frac{m}{n}$ formulae of size atmost $n$ with $\phi$ belonging to it.
  2. Check if it maps to a string in $C$.
- If $m = poly(n)$ we get $coNP \subseteq NP/Poly$.

- Strings in $C$ picked in greedy fashion.

- Strings in $C$ picked in greedy fashion.
- To prove that the procedure terminates after picking polynomially many strings.

- Strings in $C$ picked in greedy fashion.
- To prove that the procedure terminates after picking polynomially many strings.
- An iterative algorithm.

- Strings in $C$ picked in greedy fashion.
- To prove that the procedure terminates after picking polynomially many strings.
- An iterative algorithm.
- At the $(i+1)^{th}$ iteration,
  1. $S_i$ is the set of strings in $S$ that are yet to be covered,
  2. $C_{i+1}$ is the set of strings picked at or before stage $i+1$.

- Strings in $C$ picked in greedy fashion.
- To prove that the procedure terminates after picking polynomially many strings.
- An iterative algorithm.
- At the $(i+1)^{th}$ iteration,
  1. $S_i$ is the set of strings in $S$ that are yet to be covered,
  2. $C_{i+1}$ is the set of strings picked at or before stage $i+1$.
- $X = S^{\frac{m}{n}}$

- Strings in $C$ picked in greedy fashion.
- To prove that the procedure terminates after picking polynomially many strings.
- An iterative algorithm.
- At the $(i+1)^{th}$ iteration,
  1. $S_i$ is the set of strings in $S$ that are yet to be covered,
  2. $C_{i+1}$ is the set of strings picked at or before stage $i+1$.
- $X = S^{\frac{m}{n}}$
- $X_i \subseteq X$ be the set of tuples that do not belong to the pre-image set of $S_i$.

- Strings in $C$ picked in greedy fashion.
- To prove that the procedure terminates after picking polynomially many strings.
- An iterative algorithm.
- At the $(i+1)^{th}$ iteration,
  1. $S_i$ is the set of strings in $S$ that are yet to be covered,
  2. $C_{i+1}$ is the set of strings picked at or before stage $i+1$.
- $X = S^{\frac{m}{n}}$
- $X_i \subseteq X$ be the set of tuples that do not belong to the pre-image set of $S_i$.
- At the $i^{th}$ iteration, we pickk a string in $T$ with the maximum number of pre-images in $X_{i-1}$ and add it to $C_{i-1}$.

We now choose an appropriate $m$ and prove that the size of $S_i$ decreases decreases by atleast a constant factor in each stage.

We now choose an appropriate $m$ and prove that the size of $S_i$ decreases decreases by atleast a constant factor in each stage.

1. $|X_{i-1} - X_i| \geq \frac{|X_{i-1}|}{2^k}$. (Pigeonhole Principle)

We now choose an appropriate $m$ and prove that the size of $S_i$ decreases decreases by atleast a constant factor in each stage.

1. $|X_{i-1} - X_i| \geq \frac{|X_{i-1}|}{2^k}$. (Pigeonhole Principle)
2. $|S_{i-1} - S_i| \geq \frac{|X_{i-1}|^{\frac{n}{m}}}{2^{\frac{kn}{m}}}$

We now choose an appropriate $m$ and prove that the size of $S_i$ decreases decreases by atleast a constant factor in each stage.

1. $|X_{i-1} - X_i| \geq \frac{|X_{i-1}|}{2^k}$. (Pigeonhole Principle)
2. $|S_{i-1} - S_i| \geq \frac{|X_{i-1}|^{\frac{n}{m}}}{2^{\frac{kn}{m}}}$
3. $|X_{i-1}|^{\frac{n}{m}} \geq |S_{i-1}|$

We now choose an appropriate $m$ and prove that the size of $S_i$ decreases decreases by atleast a constant factor in each stage.

1. $|X_{i-1} - X_i| \geq \frac{|X_{i-1}|}{2^k}$. (Pigeonhole Principle)
2. $|S_{i-1} - S_i| \geq \frac{|X_{i-1}|^{\frac{n}{m}}}{2^{\frac{kn}{m}}}$
3. $|X_{i-1}|^{\frac{n}{m}} \geq |S_{i-1}|$

Hence we get $|S_{i-1} - S_i| \geq \frac{|S_{i-1}|}{2^{\frac{kn}{m}}}$. Since $k = (n + log(m))^c$, we can pick a constant $c' > c$ large enough such that $kn < m$ when $m = n^{c'}$. For this choice, we get $|S_i| \leq \frac{|S_{i-1}|}{2}$. Hence the proof.

# Probabilistic Compression

## Definition

Let $L$ be a parametric problem and $A \subseteq \{0,1\}^*$. $L$ is said to be probabilistically compressible with error $\epsilon(n)$ within $A$ if there is a probabilistic polynomial-time computable function $f$ such that for each $x \in \{0,1\}^*$ and $n \in \mathbb{N}$, with probability atleasy $1 - \epsilon(|x|)$ we have :

# Probabilistic Compression

## Definition

Let $L$ be a parametric problem and $A \subseteq \{0,1\}^*$. $L$ is said to be probabilistically compressible with error $\epsilon(n)$ within $A$ if there is a probabilistic polynomial-time computable function $f$ such that for each $x \in \{0,1\}^*$ and $n \in \mathbb{N}$, with probability atleasy $1 - \epsilon(|x|)$ we have :

1. $|f(< x, 1^n >)| \leq poly(n)$

# Probabilistic Compression

## Definition

Let $L$ be a parametric problem and $A \subseteq \{0,1\}^*$. $L$ is said to be probabilistically compressible with error $\epsilon(n)$ within $A$ if there is a probabilistic polynomial-time computable function $f$ such that for each $x \in \{0,1\}^*$ and $n \in \mathbb{N}$, with probability atleasy $1 - \epsilon(|x|)$ we have :

1. $|f(< x, 1^n >)| \leq poly(n)$
2. $|f(< x, 1^n >)| \in A$ *iff* $x \in L$.

# Probabilistic Compression

## Definition

Let $L$ be a parametric problem and $A \subseteq \{0,1\}^*$. $L$ is said to be probabilistically compressible with error $\epsilon(n)$ within $A$ if there is a probabilistic polynomial-time computable function $f$ such that for each $x \in \{0,1\}^*$ and $n \in \mathbb{N}$, with probability atleasy $1 - \epsilon(|x|)$ we have :

1. $|f(< x, 1^n >)| \leq poly(n)$
2. $|f(< x, 1^n >)| \in A$ *iff* $x \in L$.

We say that a probabilistic compression function has randomness complexity $R$ if it uses atmost $R$ random bits.

**Theorem**

*If $OR - SAT$ is compressible with error ¡ $2^{-m}$, where $m$ is the instance size, then $coNP \subseteq NP/Poly$ and hence PH collapses.*

## Theorem

*If $OR - SAT$ is compressible with error ¡ $2^{-m}$, where $m$ is the instance size, then $coNP \subseteq NP/Poly$ and hence PH collapses.*

## Proof.

The key observation is that compression with error $< 2^{-m}$ implies non-uniform compression. We know that there exist a random string $r$ of size atmost $poly(m)$ and the probabilistic machine yields the correct compressed instance for each instance of length $m$. This string $r$ can be a part of the advice along with the set $C$ defined in the previous proof. Hence we get $coNP \subseteq NP/Poly$. □

# Succint PCPs

- The PCP Theorem:
  - Any $NP-$Complete languate has a polynomial-size(on the unput size) proofs .
  - Verified probabilistically.
  - Constant number of queries.

# Succint PCPs

- The PCP Theorem:
  - Any $NP-$Complete languate has a polynomial-size(on the unput size) proofs .
  - Verified probabilistically.
  - Constant number of queries.
- **Question:** Can the proof can be made polynomial in the size of the *witness*.

## Definition

Let $L$ be a parametric problem. $L$ is said to have a succint PCP with completeness $c$, soundness $s$, proof size $S$ and query complexity $q$ if there is a probabilistic polynomial-time oracle machine $V$ such that the following holds for any instance $< x, 1^n >$:

## Definition

Let $L$ be a parametric problem. $L$ is said to have a succint PCP with completeness $c$, soundness $s$, proof size $S$ and query complexity $q$ if there is a probabilistic polynomial-time oracle machine $V$ such that the following holds for any instance $< x, 1^n >$:

1. If $< x, 1^n > \in L$, then there is a proof $y$ of size $S(n)$ such that on input $< x, 1^n >$, $V$ makes atmost $q$ queries to $y$ and accepts with probability atleast $c$.

## Definition

Let $L$ be a parametric problem. $L$ is said to have a succint PCP with completeness $c$, soundness $s$, proof size $S$ and query complexity $q$ if there is a probabilistic polynomial-time oracle machine $V$ such that the following holds for any instance $< x, 1^n >$:

1. If $< x, 1^n > \in L$, then there is a proof $y$ of size $S(n)$ such that on input $< x, 1^n >$, $V$ makes atmost $q$ queries to $y$ and accepts with probability atleast $c$.
2. If $< x, 1^n > \notin L$, then for any string $y$ of size $S(n)$, on input $< x, 1^n >$, $V$ makes atmost $q$ queries to $y$ and accepts with probability atmost $s$.

## Definition

Let $L$ be a parametric problem. $L$ is said to have a succint PCP with completeness $c$, soundness $s$, proof size $S$ and query complexity $q$ if there is a probabilistic polynomial-time oracle machine $V$ such that the following holds for any instance $< x, 1^n >$:

1. If $< x, 1^n > \in L$, then there is a proof $y$ of size $S(n)$ such that on input $< x, 1^n >$, $V$ makes atmost $q$ queries to $y$ and accepts with probability atleast $c$.
2. If $< x, 1^n > \notin L$, then for any string $y$ of size $S(n)$, on input $< x, 1^n >$, $V$ makes atmost $q$ queries to $y$ and accepts with probability atmost $s$.

$L$ is said to have a succint $PCP$ if it has a succint $PCP$ with completeness 1, and soundness $\frac{1}{2}$, proof size $poly(n)$ and constant query complexity.

**Theorem**

*If SAT has a succint PCP, then SAT is self-compressible with error less than* $2^{-m}$.

## Theorem

*If SAT has a succint PCP, then SAT is self-compressible with error less than* $2^{-m}$.

## Proof.

1. Given an input formula of size $m$ with $n$ variables, we use the hypothesis to find an equivalent formula of size $O(n^r)$.

## Theorem

*If SAT has a succint PCP, then SAT is self-compressible with error less than* $2^{-m}$.

## Proof.

1. Given an input formula of size $m$ with $n$ variables, we use the hypothesis to find an equivalent formula of size $O(n^r)$.
2. The variables of the formula correspond to the proof bits.

## Theorem

*If SAT has a succint PCP, then SAT is self-compressible with error less than $2^{-m}$.*

## Proof.

1. Given an input formula of size $m$ with $n$ variables, we use the hypothesis to find an equivalent formula of size $O(n^r)$.
2. The variables of the formula correspond to the proof bits.
3. Each clause corresponds to a computation path that encodes whether the proof bits read on that path cause the verifier to accept or not. Each clause can be expressed as CNF of size $q.2^q$.

## Theorem

*If SAT has a succint PCP, then SAT is self-compressible with error less than* $2^{-m}$.

## Proof.

1. Given an input formula of size $m$ with $n$ variables, we use the hypothesis to find an equivalent formula of size $O(n^r)$.
2. The variables of the formula correspond to the proof bits.
3. Each clause corresponds to a computation path that encodes whether the proof bits read on that path cause the verifier to accept or not. Each clause can be expressed as CNF of size $q.2^q$.
4. The randomness complexity $R = O(poly(m))$.

## Theorem

*If SAT has a succint PCP, then SAT is self-compressible with error less than* $2^{-m}$.

## Proof.

1. Given an input formula of size $m$ with $n$ variables, we use the hypothesis to find an equivalent formula of size $O(n^r)$.
2. The variables of the formula correspond to the proof bits.
3. Each clause corresponds to a computation path that encodes whether the proof bits read on that path cause the verifier to accept or not. Each clause can be expressed as CNF of size $q.2^q$.
4. The randomness complexity $R = O(poly(m))$.

1. Description of the self-compression: Sample independently $m^2$ random strings $r_1, ..., r_{m^2}$ each of length $R$.

1. Description of the self-compression: Sample independently $m^2$ random strings $r_1, ..., r_{m^2}$ each of length $R$.
2. For each $r_i$, compute the function $f_{r_i}$ on $q$ bits that corresponds to the 'computation' path. This can be explicitly computed in $poly(m)$ time.

1. Description of the self-compression: Sample independently $m^2$ random strings $r_1, ..., r_{m^2}$ each of length $R$.
2. For each $r_i$, compute the function $f_{r_i}$ on $q$ bits that corresponds to the 'computation' path. This can be explicitly computed in $poly(m)$ time.
3. The number of such functions is atmost $n^{Cq}2^{2^q}$.

1. Description of the self-compression: Sample independently $m^2$ random strings $r_1, ..., r_{m^2}$ each of length $R$.
2. For each $r_i$, compute the function $f_{r_i}$ on $q$ bits that corresponds to the 'computation' path. This can be explicitly computed in $poly(m)$ time.
3. The number of such functions is atmost $n^{Cq}2^{2^q}$.
4. Output the conjunction of $f_i's$.

1. Description of the self-compression: Sample independently $m^2$ random strings $r_1, ..., r_{m^2}$ each of length $R$.
2. For each $r_i$, compute the function $f_{r_i}$ on $q$ bits that corresponds to the 'computation' path. This can be explicitly computed in $poly(m)$ time.
3. The number of such functions is atmost $n^{Cq}2^{2^q}$.
4. Output the conjunction of $f_i's$.

The self-compressed formula has size that is polynomial in $n$. The compressed formula is satisfiable with probability atmost $\frac{1}{m^2}$ when the input formula is unsatisfiable. Hence the proof.