Applications of Orthogonal Arrays to Computer Science

K. Gopalakrishnan
East Carolina University
Greenville, NC, USA

AND

D. R. Stinson
University of Waterloo
Waterloo, ON, Canada

ICDM'06
Indian Institute of Science
17th Dec. 2006

Orthogonal Arrays

- An *orthogonal array* $OA_\lambda(t, k, v)$ is a $\lambda v^t \times k$ array of symbols from a v-set, such that in any t columns, every possible list of t symbols occurs in exactly λ rows.

Orthogonal Arrays

- An *orthogonal array* $OA_\lambda(t, k, v)$ is a $\lambda v^t \times k$ array of symbols from a v-set, such that in any t columns, every possible list of t symbols occurs in exactly λ rows.
- An OA is *simple* if every row is distinct.

Orthogonal Arrays

- An *orthogonal array* $OA_\lambda(t,k,v)$ is a $\lambda v^t \times k$ array of symbols from a v-set, such that in any t columns, every possible list of t symbols occurs in exactly λ rows.
- An OA is *simple* if every row is distinct.
- If $\lambda = 1$, we simply denote it by $OA(t,k,v)$.

Orthogonal Arrays

- An *orthogonal array* $OA_\lambda(t,k,v)$ is a $\lambda v^t \times k$ array of symbols from a v-set, such that in any t columns, every possible list of t symbols occurs in exactly λ rows.
- An OA is *simple* if every row is distinct.
- If $\lambda = 1$, we simply denote it by $OA(t,k,v)$.
- If $v = 2$, then these are called *binary orthogonal arrays*.

An Example

Here is a simple $OA(3,4,2)$.

1	2	3	4
0	0	0	0
1	1	0	0
1	0	1	0
0	1	1	0
1	0	0	1
0	1	0	1
0	0	1	1
1	1	1	1

An Example

Here is a simple $OA(3,4,2)$.

1	2	3	4
0	0	0	
1	1	0	
1	0	1	
0	1	1	
1	0	0	
0	1	0	
0	0	1	
1	1	1	

An Example

Here is a simple $OA(3,4,2)$.

1	2	3	4
0	0		0
1	1		0
1	0		0
0	1		0
1	0		1
0	1		1
0	0		1
1	1		1

An Example

Here is a simple $OA(3, 4, 2)$.

1	2	3	4
0		0	0
1		0	0
1		1	0
0		1	0
1		0	1
0		0	1
0		1	1
1		1	1

An Example

Here is a simple $OA(3, 4, 2)$.

1	2	3	4
	0	0	0
	1	0	0
	0	1	0
	1	1	0
	0	0	1
	1	0	1
	0	1	1
	1	1	1

An Example

Here is a simple $OA(3, 4, 2)$.

1	2	3	4
0	0	0	0
1	1	0	0
1	0	1	0
0	1	1	0
1	0	0	1
0	1	0	1
0	0	1	1
1	1	1	1

Theory of OAs

- For what parameters t, k, v and λ, do they exist?

Theory of OAs

- For what parameters t, k, v and λ, do they exist?
- How to construct them?

Theory of OAs

- For what parameters t, k, v and λ, do they exist?
- How to construct them?
- Given t, k, v, determine the smallest λ for which an $OA_\lambda(t, k, v)$ exists.

Theory of OAs

- For what parameters t, k, v and λ, do they exist?
- How to construct them?
- Given t, k, v, determine the smallest λ for which an $OA_\lambda(t, k, v)$ exists.
- For basic theory of OAs, see the recent book on Orthogonal Arrays by Hedayat, Sloane and Stuffken.

Theory of OAs

- For what parameters t, k, v and λ, do they exist?
- How to construct them?
- Given t, k, v, determine the smallest λ for which an $OA_\lambda(t, k, v)$ exists.
- For basic theory of OAs, see the recent book on Orthogonal Arrays by Hedayat, Sloane and Stuffken.
- Here we will focus on Applications of OAs to Computer Science.

Applications of OAs

- Threshold Schemes

Applications of OAs

- Threshold Schemes
- Authentication Codes

Applications of OAs

- Threshold Schemes
- Authentication Codes
- Derandomization of Algorithms

Applications of OAs

- Threshold Schemes
- Authentication Codes
- Derandomization of Algorithms
- Random Pattern Testing of VLSI Chips

Applications of OAs

- Threshold Schemes
- Authentication Codes
- Derandomization of Algorithms
- Random Pattern Testing of VLSI Chips
- Universal Hash Functions

Applications of OAs

- Threshold Schemes
- Authentication Codes
- Derandomization of Algorithms
- Random Pattern Testing of VLSI Chips
- Universal Hash Functions
- Perfect Local Randomizers

Applications of OAs

- Threshold Schemes
- Authentication Codes
- Derandomization of Algorithms
- Random Pattern Testing of VLSI Chips
- Universal Hash Functions
- Perfect Local Randomizers
- and many more.

Threshold Schemes

- A method of sharing a secret among a set of n participants so that only groups of participants of size at least t could gain access to the secret.

Threshold Schemes

- A method of sharing a secret among a set of n participants so that only groups of participants of size at least t could gain access to the secret.
- Let $\mathcal{P}$ be a set of n participants, say $P_1, P_2, \ldots P_n$.

Threshold Schemes

- A method of sharing a secret among a set of n participants so that only groups of participants of size at least t could gain access to the secret.
- Let $\mathcal{P}$ be a set of n participants, say $P_1, P_2, \ldots P_n$.
- Let $\mathcal{K}$ be the set of possible values of the secret.

The Model

- A particular secret $K \in \mathcal{K}$ is chosen by a special participant called the *dealer*, D.

The Model

- A particular secret $K \in \mathcal{K}$ is chosen by a special participant called the *dealer*, D.
- When D wants to share the secret K among the participants in $\mathcal{P}$, he gives each participant some partial information called a *share*.

The Model

- A particular secret $K \in \mathcal{K}$ is chosen by a special participant called the *dealer*, D.
- When D wants to share the secret K among the participants in $\mathcal{P}$, he gives each participant some partial information called a *share*.
- The shares should be distributed secretly. Let $\mathcal{S}$ be the set of possible values of the shares.

Perfect Threshold Schemes

- We call such a method of sharing a secret a *perfect* (t, n) *threshold scheme*, if the following two properties are satisfied.

Perfect Threshold Schemes

- We call such a method of sharing a secret a *perfect (t, n) threshold scheme*, if the following two properties are satisfied.

1. If a subset of participants $B \subseteq \mathcal{P}$ pool their shares, then they can determine the value of K provided $|B| \geq t$.

Perfect Threshold Schemes

- We call such a method of sharing a secret a *perfect (t, n) threshold scheme*, if the following two properties are satisfied.
  1. If a subset of participants $B \subseteq \mathcal{P}$ pool their shares, then they can determine the value of K provided $|B| \geq t$.
  2. On the other hand, if $|B| < t$, then they should be able to determine nothing about the value of K.

Ideal Threshold Schemes

- It is important to minimize the information (share size) a participant is expected to remember for security purposes.

Ideal Threshold Schemes

- It is important to minimize the information (share size) a participant is expected to remember for security purposes.
- It is not too difficult to see that in any perfect thresold scheme $|\mathcal{S}| \geq |\mathcal{K}|$.

Ideal Threshold Schemes

- It is important to minimize the information (share size) a participant is expected to remember for security purposes.
- It is not too difficult to see that in any perfect thresold scheme $|\mathcal{S}| \geq |\mathcal{K}|$.
- A perfect thresold scheme in which $|\mathcal{S}| = |\mathcal{K}|$, is called an *ideal threshold schemes*.

Relation to OAs

- **Theorem:** An ideal (t, n) threshold scheme with $|\mathcal{K}| = v$ exists if and only if an $OA(t, n+1, v)$ exists.

Relation to OAs

- **Theorem:** An ideal (t, n) threshold scheme with $|\mathcal{K}| = v$ exists if and only if an $OA(t, n+1, v)$ exists.
  - First observed by Keith Martin

Relation to OAs

- **Theorem:** An ideal (t, n) threshold scheme with $|\mathcal{K}| = v$ exists if and only if an $OA(t, n+1, v)$ exists.
  - First observed by Keith Martin
  - Also, independently by Dawson et. al.

Relation to OAs

- **Theorem:** An ideal (t, n) threshold scheme with $|\mathcal{K}| = v$ exists if and only if an $OA(t, n+1, v)$ exists.
  - First observed by Keith Martin
  - Also, independently by Dawson et. al.
  - Fairly simple; we shall prove half of it.

Construction

- Start with $OA(t, n+1, v)$.

Construction

- Start with $OA(t, n+1, v)$.
- First column corresponds to dealer.

Construction

- Start with $OA(t, n+1, v)$.
- First column corresponds to dealer.
- Remaining columns correspond to the participants.

Construction

- Start with $OA(t, n+1, v)$.
- First column corresponds to dealer.
- Remaining columns correspond to the participants.
- To distribute a specific key K, dealer selects a random row of OA such that K appears in the first column.

Construction

- Start with $OA(t, n+1, v)$.
- First column corresponds to dealer.
- Remaining columns correspond to the participants.
- To distribute a specific key K, dealer selects a random row of OA such that K appears in the first column.
- Dealer gives out remaining elements of the row as shares to the participants.

Validity

- When t participants pool their shares, their collective information determines a unique row of the OA. So, they can figure out K as the first element in the row.

Validity

- When t participants pool their shares, their collective information determines a unique row of the OA. So, they can figure out K as the first element in the row.
- Can a group of $t-1$ participants compute K?

Validity

- When t participants pool their shares, their collective information determines a unique row of the OA. So, they can figure out K as the first element in the row.
- Can a group of $t - 1$ participants compute K?
- Any possible value of secret along with shares of $t - 1$ participants determine a unique row of the OA.

Validity

- When t participants pool their shares, their collective information determines a unique row of the OA. So, they can figure out K as the first element in the row.
- Can a group of $t-1$ participants compute K?
- Any possible value of secret along with shares of $t-1$ participants determine a unique row of the OA.
- Hence, $t-1$ participants can get no information about the secret.

Authentication Codes - Idea

- Alice wants to communicate to Bob over a public channel.

Authentication Codes - Idea

- Alice wants to communicate to Bob over a public channel.
- Oscar, the bad guy, can introduce and/or modify messages in the channel.

Authentication Codes - Idea

- Alice wants to communicate to Bob over a public channel.
- Oscar, the bad guy, can introduce and/or modify messages in the channel.
- The purpose is to protect the *integrity* of the information (and not to provide *secrecy*).

Authentication Codes - Idea

- Alice wants to communicate to Bob over a public channel.
- Oscar, the bad guy, can introduce and/or modify messages in the channel.
- The purpose is to protect the *integrity* of the information (and not to provide *secrecy*).
- When Bob receives a message from Alice, How can he be sure that the message was really sent by Alice and is not tampered with along the way?

Authentication Codes - Definition

- An **Authentication Code** is a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{K}, \mathcal{E})$, where

Authentication Codes - Definition

- An **Authentication Code** is a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{K}, \mathcal{E})$, where

1. $\mathcal{S}$ is a finite set of *source states*. Let $|\mathcal{S}| = k$.

Authentication Codes - Definition

- An **Authentication Code** is a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{K}, \mathcal{E})$, where

  1. $\mathcal{S}$ is a finite set of *source states*. Let $|\mathcal{S}| = k$.
  2. $\mathcal{A}$ is a finite set of *authenticators*. Let $|\mathcal{A}| = l$.

Authentication Codes - Definition

- An **Authentication Code** is a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{K}, \mathcal{E})$, where

1. $\mathcal{S}$ is a finite set of *source states*. Let $|\mathcal{S}| = k$.
2. $\mathcal{A}$ is a finite set of *authenticators*. Let $|\mathcal{A}| = l$.
3. $\mathcal{K}$ is a finite set of *keys*.

Authentication Codes - Definition

- An **Authentication Code** is a four-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{K}, \mathcal{E})$, where

1. $\mathcal{S}$ is a finite set of *source states*. Let $|\mathcal{S}| = k$.
2. $\mathcal{A}$ is a finite set of *authenticators*. Let $|\mathcal{A}| = l$.
3. $\mathcal{K}$ is a finite set of *keys*.
4. For each $K \in \mathcal{K}$, there is an *authentication rule* $e_k \in \mathcal{E}$. Each $e_K : \mathcal{S} \to \mathcal{A}$.

Authentication Codes - Model

- Alice and Bob jointly choose a secret key $K \in \mathcal{K}$.

Authentication Codes - Model

- Alice and Bob jointly choose a secret key $K \in \mathcal{K}$.
- Suppose Alice wants to send $s \in \mathcal{S}$.

Authentication Codes - Model

- Alice and Bob jointly choose a secret key $K \in \mathcal{K}$.
- Suppose Alice wants to send $s \in \mathcal{S}$.
- Alice uses authentication rule e_K to produce the authenticator $a = e_K(s)$.

Authentication Codes - Model

- Alice and Bob jointly choose a secret key $K \in \mathcal{K}$.
- Suppose Alice wants to send $s \in \mathcal{S}$.
- Alice uses authentication rule e_K to produce the authenticator $a = e_K(s)$.
- Alice sends the message $m = (s, a)$ over the channel.

Authentication Codes - Model

- Alice and Bob jointly choose a secret key $K \in \mathcal{K}$.
- Suppose Alice wants to send $s \in \mathcal{S}$.
- Alice uses authentication rule e_K to produce the authenticator $a = e_K(s)$.
- Alice sends the message $m = (s, a)$ over the channel.
- When Bob receives m, he checks that $a = e_K(s)$ to authenticate.

Authentication Matrix

- An authentication code can be represented by an $|\mathcal{E}| \times |\mathcal{S}|$ *authentication matrix*, where

Authentication Matrix

- An authentication code can be represented by an $|\mathcal{E}| \times |\mathcal{S}|$ *authentication matrix*, where
  - the rows are indexed by authentication rules

Authentication Matrix

- An authentication code can be represented by an $|\mathcal{E}| \times |\mathcal{S}|$ *authentication matrix*, where
  - the rows are indexed by authentication rules
  - the columns are indexed by source states

Authentication Matrix

- An authentication code can be represented by an $|\mathcal{E}| \times |\mathcal{S}|$ *authentication matrix*, where
  - the rows are indexed by authentication rules
  - the columns are indexed by source states
  - the entry in row e and column s is $e(s)$.

Deception Probabilities

- When Oscar places a new message $m = (s, a)$ in the channel, it is called *impersonation*.

Deception Probabilities

- When Oscar places a new message $m = (s, a)$ in the channel, it is called *impersonation*.
- Let P_{d_0} be the probability of successfully impersonating.

Deception Probabilities

- When Oscar places a new message $m = (s, a)$ in the channel, it is called *impersonation*.
- Let P_{d_0} be the probability of successfully impersonating.
- When Oscar sees a message m and changes it to a message $m^{'} \neq m$, this is called *substitution*.

Deception Probabilities

- When Oscar places a new message $m = (s, a)$ in the channel, it is called *impersonation*.
- Let P_{d_0} be the probability of successfully impersonating.
- When Oscar sees a message m and changes it to a message $m' \neq m$, this is called *substitution*.
- Let P_{d_1} be the probability of successsully substituting.

Authentication Codes - Goals

- When designing a good authentication code, we want to

Authentication Codes - Goals

- When designing a good authentication code, we want to
  - Minimize P_{d_0}.

Authentication Codes - Goals

- When designing a good authentication code, we want to
  - Minimize P_{d_0}.
  - Minimize P_{d_1}.

Authentication Codes - Goals

- When designing a good authentication code, we want to
  - Minimize P_{d_0}.
  - Minimize P_{d_1}.
  - Also, minimize the number of authentication rules.

Authentication Codes - Goals

- When designing a good authentication code, we want to
  - Minimize P_{d_0}.
  - Minimize P_{d_1}.
  - Also, minimize the number of authentication rules.
- It is not too difficult to show that $P_{d_0} \geq 1/l$ and $P_{d_1} \geq 1/l$, where l is the number of authenticators.

Connection to OAs

- **Theorem:** Suppose we have an authentication code for k source states and having l authenticators, in which $P_{d_0} = P_{d_1} = 1/l$. Then

Connection to OAs

- **Theorem:** Suppose we have an authentication code for k source states and having l authenticators, in which $P_{d_0} = P_{d_1} = 1/l$. Then

1. $|\mathcal{E}| \geq l^2$, and equality occurs if and only if the authentication matrix is an orthogonal array $OA(2, k, l)$ (with $\lambda = 1$)

Connection to OAs

- **Theorem:** Suppose we have an authentication code for k source states and having l authenticators, in which $P_{d_0} = P_{d_1} = 1/l$. Then
  1. $|\mathcal{E}| \geq l^2$, and equality occurs if and only if the authentication matrix is an orthogonal array $OA(2, k, l)$ (with $\lambda = 1$)
  2. $|\mathcal{E}| \geq k(l-1)+1$, and equality occurs if and only if the authentication matrix is an $OA_\lambda(2, k, l)$ where

$$\lambda = \frac{k(l-1)+1}{l^2}.$$

A Glimpse of Proof

- Suppose, we use $OA(2, k, l)$ as the authentication matrix. Then, it is easy to see that

A Glimpse of Proof

- Suppose, we use $OA(2, k, l)$ as the authentication matrix. Then, it is easy to see that
  - Number of authentication rules is l^2.

A Glimpse of Proof

- Suppose, we use $OA(2, k, l)$ as the authentication matrix. Then, it is easy to see that
  - Number of authentication rules is l^2.
  - $P_{d_0} = 1/l$ as in each column each authenticator appears exactly l times.

A Glimpse of Proof

- Suppose, we use $OA(2, k, l)$ as the authentication matrix. Then, it is easy to see that
  - Number of authentication rules is l^2.
  - $P_{d_0} = 1/l$ as in each column each authenticator appears exactly l times.
  - $P_{d_1} = 1/l$ as any ordered pair of authenticators appears exactly once in any two selected columns.

Illustration

4 states, 3 authenticators, 9 encoding rules.

s_1	s_2	s_3	s_4
1	1	1	1
1	2	2	2
1	3	3	3
2	1	2	3
2	2	3	1
2	3	1	2
3	1	3	2
3	2	1	3
3	3	2	1

Illustration

Suppose $(s_2, 3)$ is observed by Oscar.

s_1	s_2	s_3	s_4
1	1	1	1
1	2	2	2
1	3	3	3
2	1	2	3
2	2	3	1
2	3	1	2
3	1	3	2
3	2	1	3
3	3	2	1

Illustration

Suppose Oscar wants to substitute s_4.

s_1	s_2	s_3	s_4
1	1	1	1
1	2	2	2
1	3	3	3
2	1	2	3
2	2	3	1
2	3	1	2
3	1	3	2
3	2	1	3
3	3	2	1

Probabilistic Algorithms

- A probabilistic algorithm will be using random bits during the course of its execution unlike a deterministic algorithm. There are two types.

Probabilistic Algorithms

- A probabilistic algorithm will be using random bits during the course of its execution unlike a deterministic algorithm. There are two types.

  Las Vegas algorithm may fail to give an answer with probability ϵ, but if it does give an answer, it is correct.

Probabilistic Algorithms

- A probabilistic algorithm will be using random bits during the course of its execution unlike a deterministic algorithm. There are two types.

  Las Vegas algorithm may fail to give an answer with probability ϵ, but if it does give an answer, it is correct.

  Monte Carlo algorithm always gives an answer, but the answer may be incorrect with some probability ϵ.

Generic Monte Carlo Algorithm

- Applied to decision problems

Generic Monte Carlo Algorithm

- Applied to decision problems
- Input: x

Generic Monte Carlo Algorithm

- Applied to decision problems
- Input: x
- Question: Is $x \in L$?

Generic Monte Carlo Algorithm

- Applied to decision problems
- Input: x
- Question: Is $x \in L$?
- A *sample point* $r \in \{0, 1, \ldots, n-1\}$ is chosen

Generic Monte Carlo Algorithm

- Applied to decision problems
- Input: x
- Question: Is $x \in L$?
- A *sample point* $r \in \{0, 1, \ldots, n-1\}$ is chosen
- A 0-1 valued deterministic function $f(x, r)$ is computed

Generic Monte Carlo Algorithm

- Applied to decision problems
- Input: x
- Question: Is $x \in L$?
- A *sample point* $r \in \{0, 1, \ldots, n-1\}$ is chosen
- A 0-1 valued deterministic function $f(x, r)$ is computed
- Result declared is $f(x, r)$

Generic Monte Carlo Algorithm

- Applied to decision problems
- Input: x
- Question: Is $x \in L$?
- A *sample point* $r \in \{0, 1, \ldots, n-1\}$ is chosen
- A 0-1 valued deterministic function $f(x, r)$ is computed
- Result declared is $f(x, r)$
- 0 means *No* and 1 means *Yes*.

Yes-biased Version

- If $x \notin L$ then for all $r \in \{0, 1, \ldots, n-1\}$, $f(x, r) = 0$.

Yes-biased Version

- If $x \notin L$ then for all $r \in \{0, 1, \ldots, n-1\}$, $f(x, r) = 0$.
- If $x \in L$ then the fraction of the values of r for which $f(x, r) = 1$ is at least $1 - \epsilon$.

Yes-biased Version

- If $x \notin L$ then for all $r \in \{0, 1, \ldots, n-1\}$, $f(x, r) = 0$.
- If $x \in L$ then the fraction of the values of r for which $f(x, r) = 1$ is at least $1 - \epsilon$.
- So, if the algorithm answers *Yes*, then it is correct answer.

Yes-biased Version

- If $x \notin L$ then for all $r \in \{0, 1, \ldots, n-1\}$, $f(x, r) = 0$.
- If $x \in L$ then the fraction of the values of r for which $f(x, r) = 1$ is at least $1 - \epsilon$.
- So, if the algorithm answers *Yes*, then it is correct answer.
- A *No* answer by the algorithm may be wrong.

Exposing the flaw

- If the algorithm is run k times, then the error probability is at most ϵ^k and hence can be made arbitrarily small.

Exposing the flaw

- If the algorithm is run k times, then the error probability is at most ϵ^k and hence can be made arbitrarily small.
- The analysis assumes mutual independence of the successive sample points used.

Exposing the flaw

- If the algorithm is run k times, then the error probability is at most ϵ^k and hence can be made arbitrarily small.
- The analysis assumes mutual independence of the successive sample points used.
- In reality, pseudo-random number generators are used in implementation.

Exposing the flaw

- If the algorithm is run k times, then the error probability is at most ϵ^k and hence can be made arbitrarily small.
- The analysis assumes mutual independence of the successive sample points used.
- In reality, pseudo-random number generators are used in implementation.
- So, all the sample points are completely determined by the seed.

Exposing the flaw

- If the algorithm is run k times, then the error probability is at most ϵ^k and hence can be made arbitrarily small.
- The analysis assumes mutual independence of the successive sample points used.
- In reality, pseudo-random number generators are used in implementation.
- So, all the sample points are completely determined by the seed.
- Analysis does not reflect reality.

Use of OAs

- Here is an alternative approach.

Use of OAs

- Here is an alternative approach.
- Consider an $OA(2, k, n)$.

Use of OAs

- Here is an alternative approach.
- Consider an $OA(2, k, n)$.
- This array has n^2 rows.

Use of OAs

- Here is an alternative approach.
- Consider an $OA(2, k, n)$.
- This array has n^2 rows.
- Generate $2 \log n$ true random bits.

Use of OAs

- Here is an alternative approach.
- Consider an $OA(2, k, n)$.
- This array has n^2 rows.
- Generate $2 \log n$ true random bits.
- Use them to index a specific row of the OA.

Use of OAs

- Here is an alternative approach.
- Consider an $OA(2, k, n)$.
- This array has n^2 rows.
- Generate $2 \log n$ true random bits.
- Use them to index a specific row of the OA.
- Run the Monte Carlo Algorithm (k times) using the k elements in the row selected as the sample points.

Calculation of Error Probability

- Let U denote the universe of sample points; $|U| = n$.

Calculation of Error Probability

- Let U denote the universe of sample points; $|U| = n$.
- Let $S \subseteq U$ be the set of *witnesses*; Then $|S| \geq (1 - \epsilon)n$.

Calculation of Error Probability

- Let U denote the universe of sample points; $|U| = n$.
- Let $S \subseteq U$ be the set of *witnesses*; Then $|S| \geq (1 - \epsilon)n$.
- Call a row of the OA a *bad row* if none of the elements in the row is a witness.

Calculation of Error Probability

- Let U denote the universe of sample points; $|U| = n$.
- Let $S \subseteq U$ be the set of *witnesses*; Then $|S| \geq (1 - \epsilon)n$.
- Call a row of the OA a *bad row* if none of the elements in the row is a witness.
- Then error probability is simply the probability that a randomly selected row of the OA is bad.

Calculation of Error Probability

- Let U denote the universe of sample points; $|U| = n$.
- Let $S \subseteq U$ be the set of *witnesses*; Then $|S| \geq (1-\epsilon)n$.
- Call a row of the OA a *bad row* if none of the elements in the row is a witness.
- Then error probability is simply the probability that a randomly selected row of the OA is bad.
- Can be shown to be at most $\frac{\epsilon}{1+(k-1)(1-\epsilon)}$ using combinatorial properties of OAs.

Remarks

- A similar idea was first proposed by Chor and Goldreich under the name *two-point based sampling*. They used a *specific* OA and derived bounds on error probability by using complex techniques.

Remarks

- A similar idea was first proposed by Chor and Goldreich under the name *two-point based sampling*. They used a *specific* OA and derived bounds on error probability by using complex techniques.
- The scheme presented is a generlization that

Remarks

- A similar idea was first proposed by Chor and Goldreich under the name *two-point based sampling*. They used a *specific* OA and derived bounds on error probability by using complex techniques.
- The scheme presented is a generlization that
  - works for any OA.

Remarks

- A similar idea was first proposed by Chor and Goldreich under the name *two-point based sampling*. They used a *specific* OA and derived bounds on error probability by using complex techniques.
- The scheme presented is a generlization that
  - works for any OA.
  - yields better error probability.

Remarks

- A similar idea was first proposed by Chor and Goldreich under the name *two-point based sampling*. They used a *specific* OA and derived bounds on error probability by using complex techniques.
- The scheme presented is a generlization that
  - works for any OA.
  - yields better error probability.
  - is analyzable by elementary techniques.

A comparison

Name	# ran. bits	Error Prob.
Original Scheme	$k \log n$	ϵ^k
Two-Point Scheme	$2 \log n$	$\frac{\epsilon}{(1-\epsilon)k}$
OA Scheme	$2 \log n$	$\frac{\epsilon}{1+(k-1)(1-\epsilon)}$

Extensions

- Dukes and Ling have extended the result to OAs of strength t.

Extensions

- Dukes and Ling have extended the result to OAs of strength t.
- If we use $OA(2, k, 2^n)$, the sample points would be n-bit binary vectors and so the scheme can be used for *random pattern built-in self testing of VLSI chips*.

Extensions

- Dukes and Ling have extended the result to OAs of strength t.
- If we use $OA(2, k, 2^n)$, the sample points would be n-bit binary vectors and so the scheme can be used for *random pattern built-in self testing of VLSI chips*.
- In some situations, OAs help to completely eliminate random bits used in randomized algorithms so that the resulting algorithm is a deterministic one. This process is called (total) *derandomization*.

Concluding Remarks

- There are several more applications of OAs to CS.

Concluding Remarks

- There are several more applications of OAs to CS.
- In view of their ubiquity, OAs are now getting recognized as fundamental combinatorial structures (arguably on par with Graphs)

Concluding Remarks

- There are several more applications of OAs to CS.
- In view of their ubiquity, OAs are now getting recognized as fundamental combinatorial structures (arguably on par with Graphs)
- The relationship between combinatorics and computer science is a mutually beneficial symbiotic one.