Hashing, Contention, and Cell-Probe Proofs

Yitong Yin
Advisor: James Aspnes

Department of Computer Science
Yale University

June 28, 2009
Hashing, Contention, and Cell-Probe Proofs

Yitong Yin

Mission: Searching

The Art of Computer Programming

Volume 3
Sorting and Searching
Second Edition

Donald E. Knuth
Hashing, Contention, and Cell-Probe Proofs

Yitong Yin

Preliminary
Data structure problems
Cell-probe model

New challenges
Cell-probe-proofs
Nondeterministic complexity
Locally checkable data structures

Low-contention data structures
Upper bounds
A Lower bound

Ranged hash functions
Lower bounds for the monotone case
Non-monotone case

Summary
Searching in the 21st Century

What does searching become?
Searching in the 21st Century

What does searching become?
Outline

Preliminary
  Data structure problems
  Cell-probe model

New challenges

Cell-probe-proofs
  Nondeterministic complexity
  Locally checkable data structures

Low-contention data structures
  Upper bounds
  A Lower bound

Ranged hash functions
  Lower bounds for the monotone case
  Non-monotone case

Summary
Outline

Preliminary
  Data structure problems
  Cell-probe model

New challenges

Cell-probe-proofs
  Nondeterministic complexity
  Locally checkable data structures

Low-contention data structures
  Upper bounds
  A Lower bound

Ranged hash functions
  Lower bounds for the monotone case
  Non-monotone case

Summary
The problems of searching

Data structure problems.
The problems of searching

Data structure problems.
Definition (Elias and Flower, 1975)

A data structure problem is a function \( f : X \times Y \rightarrow Z \). For each query \( x \in X \) and each data set \( y \in Y \), \( f(x, y) \) defines the answer to the query \( x \) on data \( y \).

Decision version

\( f : X \times Y \rightarrow \{0, 1\} \).
Examples: “basic” searching problems

Membership query
Each query \( x \) is an element of universe \([N]\), and each data set is a set \( S \in \binom{[N]}{n} \). The query answers the question of “Is \( x \in S \)?”.

\[
f(x, S) := \begin{cases} 
1 & x \in S \\
0 & \text{o.w.}
\end{cases}
\]

Predecessor search
\[
f(x, S) := \max\{z \in S \mid z \leq x\}.
\]
Examples: “advanced” searching problems

Nearest neighbor search (Hamming space)
$X = \{0, 1\}^d$ and $Y = \binom{X}{n}$, for every $(x, y) \in X \times Y$, $f(x, y)$ is the point in $y$ which is closest to $x$ (Hamming distance).

Partial match
$X = \{0, 1, \ast\}^d$ and $Y = \binom{\{0, 1\}^d}{n}$, for every $(x, y) \in X \times Y$, $f(x, y) := 1$ iff $x$ matches some $z \in y$.

- “Curse of dimensionality”: if $d = \omega(\log n)$, the above two problems are hard.
Examples: more general problems

Polynomial evaluation

$X$ is a finite field, and $Y$ is the set of all $d$-degree polynomial over $X$. For every $x \in X$ and $y \in Y$,

$$f(x, y) := y(x).$$

Random problem

For every $(x, y) \in X \times Y$,

$$f(x, y) := \text{random value}.$$
Outline

Preliminary
Data structure problems
Cell-probe model

New challenges

Cell-probe-proofs
Nondeterministic complexity
Locally checkable data structures

Low-contention data structures
Upper bounds
A Lower bound

Ranged hash functions
Lower bounds for the monotone case
Non-monotone case

Summary
Cell-probe model [Yao, 1981]

Cell-probing scheme:
- The data set $y$ is encoded to a **table of cells**.
- For each query $x$, the result $f(x, y)$ is decided by **adaptive probes**.

![Data diagram](image)
Cell-probe model [Yao, 1981]

Cell-probing scheme:

- The data set \( y \) is encoded to a table of cells.
- For each query \( x \), the result \( f(x, y) \) is decided by adaptive probes.
Cell-probe model [Yao, 1981]

Cell-probing scheme:
- The data set $y$ is encoded to a **table of cells**.
- For each query $x$, the result $f(x, y)$ is decided by adaptive probes.

<table>
<thead>
<tr>
<th>12</th>
<th>19</th>
<th>23</th>
<th>24</th>
<th>28</th>
<th>37</th>
<th>42</th>
<th>45</th>
<th>53</th>
<th>60</th>
<th>67</th>
<th>68</th>
<th>79</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>
Cell-probe model [Yao, 1981]

Cell-probing scheme:

- The data set $y$ is encoded to a table of cells.
- For each query $x$, the result $f(x, y)$ is decided by adaptive probes.

<table>
<thead>
<tr>
<th>12</th>
<th>19</th>
<th>23</th>
<th>24</th>
<th>28</th>
<th>37</th>
<th>42</th>
<th>45</th>
<th>53</th>
<th>60</th>
<th>67</th>
<th>68</th>
<th>79</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

query 21

Probing
Two views of cell-probe model

- An encoding scheme and a decision tree.

- A communication protocol between an adaptive player and an oblivious player.

\[ f(x,y) \]

- An encoding scheme and a decision tree.

- A communication protocol between an adaptive player and an oblivious player.
Space/time tradeoff

For any $f$, there are two naïve solutions:

- **Bit vector**: store $f(x, y)$ for every $x$.
- **Read the raw data $y$ every time.**
New Challenges

- **Locality**: the information for answering searches is locally accessible.
- **Contention**: the same resource is requested by multiple processors at the same time.
- **Churn**: the set of participants changes.

Our approach: Lower bounds. (What cannot be done?)
“Love Thy Neighbor.”
Local verifications

The data structure is implemented by a distributed system.

An important property of distributed systems is locality.
Local verifications

- In a distributed hash table (DHT), answering a query consists of two actions: routing, and **local verification**.

```
Routing-Verification Paradigm
loop
    if f(x, y) can be decided locally from P then
        return f(x, y);
    end if
    P ← next(P, x);
end loop
```

- Generalizing the scheme, we get a **routing-verification paradigm** for any problem $f$.

**Question**
What cannot be verified locally?
In a distributed hash table (DHT), answering a query consists of two actions: routing, and local verification.

**Routing-Verification Paradigm**

```
loop
  if f(x, y) can be decided locally from P then
    return f(x, y);
  end if
  P ← next(P, x);
end loop
```

Generalizing the scheme, we get a routing-verification paradigm for any problem $f$.

**Question**
What cannot be verified locally?
Outline

Preliminary
- Data structure problems
- Cell-probe model

New challenges

Cell-probe-proofs
- Nondeterministic complexity
- Locally checkable data structures

Low-contention data structures
- Upper bounds
- A Lower bound

Ranged hash functions
- Lower bounds for the monotone case
- Non-monotone case

Summary
Nondeterminism and verification

- **Nondeterministic cell-probes**: information to decide the query result.
- **Example**: membership query over a sorted table.

Query $x = 7$. Is $x \in S$?
Nondeterminism and verification

- **Nondeterministic cell-probes**: information to decide the query result.
- **Example**: membership query over a sorted table.

Query $x = 7$. Prove that $x \in S$. 
Nondeterminism and verification

- **Nondeterministic cell-probes:** information to decide the query result.
- **Example:** membership query over a sorted table.

Query $x = 8$. Prove that $x \notin S$. 
Nondeterminism and verification

- **Nondeterministic cell-probes**: information to decide the query result.
- **Example**: membership query over a sorted table.

Query $x = 8$. Cannot prove anything.
Cell-probe proofs.

The prover and the verifier communicate via the table.

Completeness
\( \forall (x, y), \exists \text{ a proof } P, \text{ verify}(x, P) = f(x, y) \)

Soundness
\( \forall (x, y), \forall \text{ proof } P, \text{ verify}(x, P) = \begin{cases} f(x, y) \\ \bot \end{cases} \)

\((s, b, t)\)-CPP: s cells, each of b bits, t-cell proof for \( \forall (x, y) \).
Cell-probe proofs.

The prover and the verifier communicate via the table.

\[ \forall (x, y), \exists \text{ a proof } P, \text{ verify}(x, P) = f(x, y) \]

**Completeness**

**Soundness**

\[ \forall (x, y), \forall \text{ proof } P, \text{ verify}(x, P) = \begin{cases} f(x, y) \\ \bot \end{cases} \]

\((s, b, t)\text{-CPP}: \) s cells, each of b bits, t-cell proof for \(\forall (x, y)\).
Cell-probe proofs.

The prover and the verifier communicate via the table.

Completeness
∀(x, y),
∃ a proof P,
\textbf{verify}(x, P) = f(x, y)

Soundness
∀(x, y),
∀ proof P,
\textbf{verify}(x, P) = \begin{cases} f(x, y) \\ \bot \end{cases}

\textbf{(s, b, t)-CPP}: s cells, each of b bits, t-cell proof for ∀(x, y).
Cell-probe proofs.

The prover and the verifier communicate via the table.

\[ \forall (x, y), \exists \text{ a proof } P, \text{ verify}(x, P) = f(x, y) \]

Completeness

\[ \forall (x, y), \forall \text{ proof } P, \text{ verify}(x, P) = \begin{cases} f(x, y) \\ \bot \end{cases} \]

Soundness

\((s, b, t)\)-CPP: \(s\) cells, each of \(b\) bits, \(t\)-cell proof for \(\forall (x, y)\).
A combinatorial characterization of CPP

Definition
\( \mathcal{F} \) is an \( s \times k \)-partition of set \( Y \), if \( \mathcal{F} = \bigcup_{i=1}^{s} G_i \) where each \( G_i \) is a partition of \( Y \) and \( |G_i| \leq k \).

Example
A \( 3 \times 2 \)-partition of \( \{1, 2, 3, 4\} \).

\[ \mathcal{F} = \{ \{1, 2\}, \{3, 4\}, \{1, 2, 4\}, \{3\}, \{1, 2, 3, 4\} \} \]

Theorem (Characterization)
There exists an \((s, b, t)\)-CPP for the data structure problem \( f : X \times Y \rightarrow Z \) if and only if there exists an \( s \times 2^b \)-partition \( \mathcal{F} \) of \( Y \) such that for every \( (x, y) \in X \times Y \), there exists an \( A \in \mathcal{F} \) such that \( y \in A \) and \( |f(x, A)| = 1 \).
Nondeterministic cell-probe complexity

For uniformly random \( f : \{0, 1\}^m \times \{0, 1\}^n \rightarrow \{0, 1\} \):

**Theorem**

*If the space complexity is:*
  - \( 2^m - 1 \) cells;
  - 1 bit in each cell;

*then the nondeterministic time complexity is \( \Omega(\frac{n}{m}) \).*
Nondeterministic cell-probe complexity

For the nearest neighbor search problem or the partial match problem with \( n \) points in \( d \)-dimensional Hamming space where \( d = \omega(\log n) \):

**Theorem**

*If the space complexity is:*

- \( \text{Poly}(n) \) cells;
- \( O(n^{1-C}) \) bits in each cell for some \( C > 0 \);

*then the nondeterministic time complexity is \( \Omega(\log(\frac{d}{\log n})) \).*

**Remark**

The current deterministic lower bound for the problem is \( \Omega(\frac{d}{\log n}) \).
Nondeterministic cell-probe complexity

For the $d$-degree polynomial evaluation problem defined over the finite field $GF(2^k)$:

**Theorem**

*If the space complexity is:*

- $s$ cells;
- $k$ bits in each cell;

*then the nondeterministic time complexity is $\min(d, \frac{k - \log d}{\log s})$.***

**Remark**

This nondeterministic lower bound matches the asymptotically tight deterministic lower bound $\min(d + 1, \frac{k - \log d}{\log s})$.***
Outline

Preliminary

Data structure problems
Cell-probe model

New challenges

Cell-probe-proofs

Nondeterministic complexity
Locally checkable data structures

Low-contention data structures

Upper bounds
A Lower bound

Ranged hash functions

Lower bounds for the monotone case
Non-monotone case

Summary
Locally checkable data structure

- The data set $y$ is encoded to:
  - a set of cells;
  - and a network.
- For every query $x$, there exists a cell $P$, such that $f(x, y)$ can be decided locally from $P$.
- $(s, b, \delta)$-locally checkable data structure: $s$ cells, each of $b$ bits, with maximum degree $\delta$. 

- preprocessing
- query: $x$
- verification
- $f(x, y)$

Data structure problems
Cell-probe model
New challenges
Cell-probe-proofs
Nondeterministic complexity
Locally checkable data structures
Low-contention data structures
Upper bounds
A Lower bound
Ranged hash functions
Lower bounds for the monotone case
Non-monotone case
Summary
Locally checkable data structure as cell-probe proofs

A **locally checkable data structure** is a data structure with local cell-probe proofs.

**Lemma**

\[(s, b, \delta)\text{-locally checkable data structure} \implies (s, \delta(b + \log s), 1)\text{-CPP}.

**Proof**

- The local decision maker is the verifier of the CPP.
- Each cell contains its entire neighborhood.
Lower bounds for locally checkable data structures

For the following problems:

- high-dimensional nearest neighbor search;
- high-dimensional partial match;
- polynomial evaluation;

let $n = \log |Y|$ be the size of the raw data, and $\epsilon = o(1)$:

**Theorem**

*There does not exist a $(\text{Poly}(n), n^\epsilon, n^\epsilon)$-locally checkable data structure for these problems.*
“Spread the table and contention will cease.”
A spellchecker example

- A spell-checker queries in a dictionary.
- “Multi-core!”
- Memory contention!
A spellchecker example

- A spell-checker queries in a dictionary.
- “Multi-core!”
- Memory contention!
A spellchecker example

- A spell-checker queries in a dictionary.
- “Multi-core!”
- Memory contention!
Most data structures fail to address contention: Binary search, BST, hashing, indexing, ...:(

Contention depends on query distribution.

Information of query distribution can be used for data structure construction.
Balanced cell-probing scheme

- **$x$: query; $y$: data; $q$: query distribution.**
  - The table is constructed from $y$ and $q$.
  - $x$ is generated by $q$.
  - The cell-probing algorithm does not know $q$.
  - At any step, the **contention** of a cell is defined as the probability that it is probed.
Balanced cell-probing scheme

Definition

\((s, b, t, \phi)\)-balanced cell-probing scheme:

- \(s\) cells, each of \(b\) bits;
- worst-case time complexity \(t\);
- the maximum contention in each step is at most \(\phi\).

Ideally, \(\phi = \frac{1}{s}\).
Outline

Preliminary
   Data structure problems
   Cell-probe model

New challenges

Cell-probe-proofs
   Nondeterministic complexity
   Locally checkable data structures

Low-contention data structures
   Upper bounds
   A Lower bound

Ranged hash functions
   Lower bounds for the monotone case
   Non-monotone case

Summary
Uniform membership query

Theorem
For the membership query,

- if the query distribution is uniform over both positive and negative queries,
- there exists an \( O(n), O(\log n), O(1), O(\frac{1}{n}) \)-balanced cell-probing scheme.

Remark

- The construction takes linear time.
- Based on Fredman, Komlos, and Szemeredi, 1984. (FKS perfect hashing)
- Based on Dietzfelbinger and auf der Heide, 1990. (de-amortized dynamic perfect hashing)
A general reduction

Theorem

\((s, b, t)\)-cell-probing scheme

\[ \Rightarrow (O\left(\frac{st \cdot \log s}{\log \log s}\right), b, O\left(\frac{t \cdot \log s}{\log \log s}\right), O\left(\frac{1}{s}\right)) \]-balanced cell-probing scheme.

Proof

The core of the reduction is a low-contention data structure for the prefix-sum problem.

Definition

The prefix-sum problem \( f : X \times Y \rightarrow Z \):

- \( X := \{1, 2, \ldots, s\} \).
- \( Y := \{y \in [s + 1]^s \mid \sum_{i=1}^{s} y_i = s\} \).
- \( f(x, y) := \sum_{i<x} y_i \).
- Query distribution \( q(x) := \frac{y_x}{s} \).
A general reduction

Lemma

There exists an $(O(\frac{s \cdot \log s}{\log \log s}), b, O(\frac{\log s}{\log \log s}), O(\frac{1}{s}))$-balanced cell-probing scheme for the prefix-sum problem.

The reduction

With prefix-sum, each original cell-probe is simulated by $O(\frac{\log s}{\log \log s})$ balanced cell-probes on a new table of size $O(\frac{s \log s}{\log \log s})$. Repeat this for every one of the $t$ cell-probes.
Application

Corollary

There exists an
\( O\left( \frac{n \cdot \log n}{\log \log n} \right) , O(\log n) , O\left( \frac{\log n}{\log \log n} \right) , O\left( \frac{1}{n} \right) \)-balanced cell-probing scheme for the membership query.

Back to the spell-checker

\( M \)-word document; sufficiently many cores; zero tolerance of memory contention.

- Before: sequential queries,
  \( O(M) \) total time, \( O(n) \) space.

- After: parallel queries,
  \( O\left( \frac{M}{\sqrt{n}} \right) \) total time, \( O\left( \frac{n \log n}{\log \log n} \right) \) space.

\( n \) is the number of English words.
Outline

Preliminary
  Data structure problems
  Cell-probe model

New challenges

Cell-probe-proofs
  Nondeterministic complexity
  Locally checkable data structures

Low-contention data structures
  Upper bounds
  A Lower bound

Ranged hash functions
  Lower bounds for the monotone case
  Non-monotone case

Summary
An interesting class of algorithms

Independent probes

A randomized cell-probing algorithm, such that:

- given a query and a table,
  the probing sequence $I_1, I_2, \ldots, I_t$ are independent.

Remark

- Cell-probes are still adaptive.
- All our upper bounds have the property.
- All deterministic cell-probing schemes have the property.
- Randomization is used only for balancing the contention.
An $\Omega(\log \log n)$ lower bound

For the cell-probing algorithms defined in the last slide:

**Theorem**

*If there exists an $(s, \text{Polylog}(n), t, \frac{\text{Polylog}(n)}{s})$-balanced cell-probing scheme for the membership query, then $t = \Omega(\log \log n)$.*

**Remark**

- The theorem is a time-contention tradeoff.
- If the contention is within poly-logarithmic to the optimal, the time complexity is $\Omega(\log \log n)$.
- The tradeoff works for unbounded space.

**Generality**

The tradeoff holds for a general class of problems.
Proof to the lower bound

- Run $n$ instances in parallel.
- Alice for the algorithm, with inputs $x_1, x_2, \ldots, x_n$; Bob for the table, with inputs $y$ and $q$.
- In each round, Alice sends “balanced” parallel probes $P_1, P_2, \ldots, P_n$.
- Alice gets $n$ bits information in $t$ rounds.
Proof to the lower bound

- Parallel probes are simulated by a joint distribution $S$ of $\{P_1, P_2, \ldots, P_n\}$.
- The marginal distributions are the same as before.
- $|\{P_1, P_2, \ldots, P_n\}|$, the total number of cells probed by Alice in a round, is minimized by $S$. 
Proof to the lower bound

A recursion is built:

- Alice knows little about $q$
- $P_i$ is oblivious
- Alice gets little information
- $|\{P_i\}|$ is small

The total information is $\geq n$ bits.
“Churn!”
Idea of hashing

- A hash function $h : [N] \rightarrow [m]$; $[N]$ for items; $[m]$ for buckets.
- For an $S \subset [N]$, the load of bucket $i$:
  \[ \ell(i, S) := |\{ j \in S \mid h(j) = i \}|. \]
- The idea of hashing:
  Balance the unknown $S$ with a consistent $h$.
- A fundamental flaw: for any $h$, $\exists$ a bad $S$.
- Randomization: the OPT is the balls-into-bins bound.

\[
\max_{S \in \binom{[N]}{n}} \max_{i \in [m]} \ell(i, S) = \begin{cases} 
\Theta \left( \frac{\log n}{\log \log n} \right) & n = \Theta(m) \\
\Theta \left( \frac{n}{m} \right) & n = \Omega(m \log m) 
\end{cases}
\]

The “price of unknown data”!
Churn

A hash function is implemented by a distributed system.

- Buckets: machines in a system.
- Buckets may become available or unavailable.
- Churn!
Ranged hash functions


- Basis for many DHTs.
- Consistent hashing $\in$ ranged hash functions.
Definition

A ranged hash function $h : 2^{|M|} \times [n] \rightarrow [M]$, 
- for each set $S \subseteq [M]$ of available buckets, 
- $h_S$ specifies a mapping from $[n]$ to $S$.

Remark

- $n$ fixed items are mapped to $M$ unreliable buckets. 
- State: the set $S$ of available buckets. 
- The load of bucket $i$ at state $S$:

$$\ell(i, S) := |h_S(i)^{-1}|.$$ 

- The “price of churn”:

$$\ell_h := \max_{S \in \binom{[M]}{m}} \max_{i \in S} \ell(i, S).$$
Performance measures

Two performance measures for a ranged hash function $h$:

- **Worst-case maximum load:**
  \[ \ell_h = \max_{S \in \binom{[M]}{m}} \max_{i \in S} \ell(i, S). \]

- **Reassignment costs:**
  $h_S, h_T \in S^n$, the reassignment costs from $S$ to $T$:
  \[ h_S \Delta h_T. \]

- **Consistent hashing:** $O\left(\frac{n}{m} \log m\right)$ maximum load with minimal reassignment costs.
Outline

Preliminary
  Data structure problems
  Cell-probe model

New challenges

Cell-probe-proofs
  Nondeterministic complexity
  Locally checkable data structures

Low-contention data structures
  Upper bounds
  A Lower bound

Ranged hash functions
  Lower bounds for the monotone case
  Non-monotone case

Summary
Monotonicity

Definition

$h$ is monotone if for all $S \subset T \subset [M]$, $h_T(i) \in S$ implies that $h_S(i) = h_T(i)$.

Being monotone:
Items are reassigned only if necessary.

Lemma (Karger et al.)

$h$ is monotone iff each item is always assigned to the first available bucket in its preference list.

- Consistent hashing is monotone.
Load balance vs. monotonicity

**Theorem**

If a randomized ranged hash function \( h : 2^{[M]} \times [n] \rightarrow [M] \) is monotone, then w.h.p., the maximum load for the worst-case \( S \in \binom{[M]}{m} \) is

\[
\ell_h = \begin{cases} 
\Omega \left( \sqrt{\frac{n}{m} \log m} \right) & n = o(m \log m) \\
\Omega \left( \frac{n}{m} \right) & n = \Omega(m \log m)
\end{cases}
\]

The lower bound is tight.

**Perturbed cube**

All items and buckets are uniformly embedded into a hypercube with a random perturbation.
Load balance vs. expansion

Generalize consistent hashing:

Hashing via metric embedding

- All items \([n]\) and all buckets \([M]\) are embedded into a metric space.
- Each item is assigned to the nearest available bucket.

Expansion rate

The expansion rate or the KR-dimension of a metric embedding is \(d\) if the volume of any ball expands at most \(2^d\) times by doubling the radius.
Load balance vs. expansion

Theorem

*If the expansion rate of the embedding is $d$, it holds w.h.p. that the maximum load is*

$$\ell_h = \Omega \left( 4^{-d} \cdot \frac{n}{m} \log m \right).$$

Corollary

*If the metric embedding is growth-restricted, i.e. with $O(1)$ expansion rate, the maximum load is*

$$\ell_h = \Omega \left( \frac{n}{m} \log m \right).$$

- Worse than balls-into-bins!
Outline

Preliminary
  Data structure problems
  Cell-probe model

New challenges

Cell-probe-proofs
  Nondeterministic complexity
  Locally checkable data structures

Low-contention data structures
  Upper bounds
  A Lower bound

Ranged hash functions
  Lower bounds for the monotone case
  Non-monotone case

Summary
Being non-monotone

- $n$ items are hashed to $m$ available buckets from $[M]$.
- **BinHash:**
  1. Binning: $[n]$ items are packed into $\frac{m}{2}$ bins $[\frac{m}{2}]$. Item $i$ is put into bin $j$ if $j$ is the maximal suffix of $i$ (in binary).
  2. Hashing: assign bins to buckets by *uniform hashing*.
- BinHash achieves both asymptotically optimal maximum load and reassignment costs.
- BinHash works for *weighted* items.
Summary

- Most data structure problems are hard nondeterministically.
- For problems such as high-dimensional nearest neighbor search, partial match, and polynomial evaluation, the answers cannot be verified locally.
- Low contention can be traded for a price of time and space, but even with unbounded space, there is always a price of time to pay.
- Although dimensionality ruins the locality of verification, it actually helps load balancing.
- In a metric space with bounded expansion rate, which is the case for Internet, the unreliability of machines costs more than the uncertainty of data.
Publication list

Hashing, Contention, and Cell-Probe Proofs

Yitong Yin

Preliminary
- Data structure problems
- Cell-probe model

New challenges
- Cell-probe-proofs
- Nondeterministic complexity
- Locally checkable data structures

Low-contention data structures
- Upper bounds
- A Lower bound

Ranged hash functions
- Lower bounds for the monotone case
- Non-monotone case

Summary