Sorted Neighborhood Methods

2.7.2013
Felix Naumann
Duplicate Detection
Number of comparisons: All pairs

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

400 comparisons
Reflexivity of Similarity

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

380 comparisons
Symmetry of Similarity

190 comparisons
Blocking by ZIP

32 comparisons
Overview

- The Original
- Unique sorting keys
- Adaptive SNM
  - Part 1
  - Part 2
- Sorted Blocks
- Domain-independent SNM
The Sorted Neighborhood Method

- **Input:**
  - Table with N tuples
  - Similarity measure

- **Output:**
  - Classes (clusters) of equivalent tuples (duplicates)

- **Problem:** Many tuples
  - Comparing each tuple-pair is inefficient
  - Large table may not fit in main memory (scalability)

---


Sorted Neighborhood
[Hernandez Stolfo 1998]

- **Idea**
  - Sort tuples so that similar tuples are close to each other.
  - Only compare tuples within a small neighborhood (window).

1. Generate key
   - E.g.: SSN+“first 3 letters of name“ + ...
   - Effectiveness strongly depends on choice of key
   - Key is only virtual and not unique (”sorting key“)

2. Sort by key
   - Similar tuples end up close to each other.

3. Slide window over sorted tuples
   - Compare all pairs of tuples within window.

- **Problems**
  - Choice of key
  - Choice of window size

- **Complexity:** At least 3 passes over data
  - Sorting!
### SNM – Example

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Year</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Mask of Zorro</td>
<td>1998</td>
<td>Adventure</td>
</tr>
<tr>
<td>18</td>
<td>Addams Family</td>
<td>1991</td>
<td>Comedy</td>
</tr>
<tr>
<td>25</td>
<td>Rush Hour</td>
<td>1998</td>
<td>Comedy</td>
</tr>
<tr>
<td>31</td>
<td>Matrix</td>
<td>1999</td>
<td>Sci-Fi</td>
</tr>
<tr>
<td>52</td>
<td>Return of Dschafar</td>
<td>1994</td>
<td>Children</td>
</tr>
<tr>
<td>113</td>
<td>Adams Family</td>
<td>1991</td>
<td>Comedie</td>
</tr>
<tr>
<td>207</td>
<td>Return of Djaffar</td>
<td>1995</td>
<td>Children</td>
</tr>
</tbody>
</table>

#### Key

<table>
<thead>
<tr>
<th>ID</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>MSKAD98</td>
</tr>
<tr>
<td>18</td>
<td>DDMCO91</td>
</tr>
<tr>
<td>25</td>
<td>RSHCO98</td>
</tr>
<tr>
<td>31</td>
<td>MTRSC99</td>
</tr>
<tr>
<td>52</td>
<td>RTRCH94</td>
</tr>
<tr>
<td>113</td>
<td>DMSCO91</td>
</tr>
<tr>
<td>207</td>
<td>RTRCH95</td>
</tr>
</tbody>
</table>

#### Create Key

1. **Create key**

#### Sort

2. **Sort**

#### Merge

3. **Merge**

- `classify(18,113) \rightarrow` duplicates
- `classify(52,207) \rightarrow` duplicates

---

**Felix Naumann | Data Profiling and Data Cleansing | Summer 2013**
SNM by ZIP (window size 4)

54 comparisons
Sorted Neighborhood – Complexity

- **N**: Number of tuples
- **w**: Window size
- **Computational complexity**:
  - $O(N) + O(N \log N) + O(w \cdot N) = O(N \log N)$
    - if $w < \log N$; $O(wN)$ else
- **IO complexity**
  - Linear in $N$
  - Three passes over table on disk
    - Create key, sort, window
  - Sorting: e.g. TPMMS
Sorted Neighborhood – Configuration

- Choice of key
  - Formulierung durch Experten
  - Aufwändig
  - Schwer vergleichbare Ergebnisse
  - Für Effektivität entscheidend

- Choice of window size
  - \( w = N : \mathcal{O}(N^2) \Rightarrow \text{max. accuracy & max. Zeit} \)
  - \( w = 2 : \mathcal{O}(N) \Rightarrow \text{min. accuracy & min. Zeit} \)

- Choice of classification method / similarity measure
  - Hernandez and Stolfo suggest „equational theory“
  - Rule set
Sorted Neighborhood – Multipass Approach

- Problem in choice of key
  - Example: \( r_1: 193456782 \) und \( r_2: 913456782 \)

- Solution 1:
  - Extend window size: \( w \rightarrow N \)

- Solution 2:
  - Multiple passes with different keys
  - Can keep \( w \) small
  - Transitive closure on results of each pass
Suggested Extensions

- Incremental SNM
  - Handle inserts
  - Trivial extension

- Parallel SNM
  - Each multi-pass in parallel
  - Parallel windows
  - See also current seminar "Large Scale Duplicate Detection"
    - Final presentations: July 10, 9:15 – 12:30 in ???
Overview

- The Original
- Unique sorting keys
- Adaptive SNM
  - Part 1
  - Part 2
- Sorted Blocks
- Domain-independent SNM
Choice of sorting key(s)

- General problem: Sortation among same keys is random
- Idea:
  - Create inverted index on sorting key
  - Slide (smaller) window over index
    - $w=1 \Rightarrow$ traditional blocking

Example

<table>
<thead>
<tr>
<th>Window positions</th>
<th>BKVs (Surname)</th>
<th>Identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Millar</td>
<td>R6</td>
</tr>
<tr>
<td>2</td>
<td>Miller</td>
<td>R2, R8</td>
</tr>
<tr>
<td>3</td>
<td>Myler</td>
<td>R4</td>
</tr>
<tr>
<td>4</td>
<td>Peters</td>
<td>R3</td>
</tr>
<tr>
<td>5</td>
<td>Smith</td>
<td>R1</td>
</tr>
<tr>
<td>6</td>
<td>Smyth</td>
<td>R5, R7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Window range</th>
<th>Candidate record pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 3</td>
<td>(R6,R2), (R6,R8), (R6,R4), (R2,R8), (R2,R4), (R8,R4)</td>
</tr>
<tr>
<td>2 – 4</td>
<td>(R2,R8), (R2,R4), (R2,R3), (R8,R4), (R8,R3), (R4,R3)</td>
</tr>
<tr>
<td>3 – 5</td>
<td>(R4,R3), (R4,R1), (R3,R1)</td>
</tr>
<tr>
<td>4 – 6</td>
<td>(R3,R1), (R3,R5), (R3,R7), (R1,R5), (R1,R7), (R5,R7)</td>
</tr>
</tbody>
</table>
Further ideas for key

- **Q-Grams**

<table>
<thead>
<tr>
<th>Identifiers</th>
<th>BKVs (Surname)</th>
<th>Bigram sub-lists</th>
<th>Index key values</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Smith</td>
<td>[sm, mi, it, th], [mi, it, th], [sm, it, th], [sm, mi, th], [sm, mi, it]</td>
<td>smmiitth, miitth, smith, smmith, smmiit</td>
</tr>
<tr>
<td>R2</td>
<td>Smithy</td>
<td>[sm, mi, it, th, hy], [mi, it, th, hy], [sm, it, th, hy], [sm, mi, th, hy], [sm, mi, it, hy], [sm, mi, it, th]</td>
<td>smmiitthhy, miitthhy, smithhy, smmithhy, smmiithhy, smmiithth</td>
</tr>
<tr>
<td>R3</td>
<td>Smithe</td>
<td>[sm, mi, it, th, he], [mi, it, th, he], [sm, it, th, he], [sm, mi, th, he], [sm, mi, it, he], [sm, mi, it, th]</td>
<td>smmiitthhe, miitthhe, smithhe, smmithhe, smmiithe, smmiitth</td>
</tr>
</tbody>
</table>

- **Suffix array (up to certain length)**
- **Soundex and other phonetic codes**
- **Canopy clustering**
  - Use cheap clustering approach to form blocks
- And many more
Overview

- The Original
- Unique sorting keys
- Adaptive SNM
  - Part 1
  - Part 2
- Sorted Blocks
- Domain-independent SNM
One size fits all?

- **Selection of window size w**
  - Too small -> some duplicates might be missed
  - Too large -> many unnecessary comparisons

Cluster sizes for the Cora Citation Matching data set (1,879 references of research papers)
Yan et al. [16] discuss adaptivity of record linkage algorithms using the example of SNM. They use the window to build non-overlapping blocks that can contain different numbers of records. The pairwise record comparison then takes place within these blocks. The hypothesis is that the distance between a record and its successors in the sort sequence is monotonically increasing in a small neighborhood, although the sorting is done lexicographically and not by distance. They present two algorithms and compare them with the basic SNM.

Incrementally Adaptive-SNM (IA-SNM) is an algorithm that incrementally increases the window size as long as the distance of the first and the last element in the current window is smaller than a specified threshold. The increase of the window size depends on the current window size.

Accumulative Adaptive-SNM (AA-SNM) on the other hand creates windows with one overlapping record. By considering transitivity, multiple adjacent windows can then be grouped into one block, if the last record of a window is a potential duplicate of the last record in the next adjacent window. After the enlargement of the windows both algorithms have a retrenchment phase, in which the window is decreased until all records within the block are potential duplicates.

We have implemented both IA-SNM and AA-SNM, and compare them to our work in our experimental evaluation. However, our experiments do not confirm that IA-SNM and AA-SNM perform better than SNM.

Reproducability


From: Oliver Wonneberg, Entlarvung der Adaptive Sorted Neighborhood Method, BTW 2009 Studierendenprogramm
Overview

- The Original
- Unique sorting keys
- Adaptive SNM
  - Part 1
  - Part 2
- Sorted Blocks
- Domain-independent SNM
Adaptation Idea

- Vary window size based on detected duplicates
  - Adaptation can increase or reduce number of comparisons
- The more duplicates of a record are found within a window, the larger the window should be
- If no duplicate of a record within its neighborhood is found, assume that there are no duplicates or the duplicates are very far away in the sorting order.
- Each tuple $t_i$ is once at the beginning of a window
  - Compare it with $w - 1$ successors
  - Current window: $W(i, i + w - 1)$
  - If no duplicate for $t_i$ is found, continue as normal
  - If a duplicate is found, increase window

Basic Duplicate Count Strategy

1. Assign sorting key to each record and sort the records
2. Create window with initial window size \( w \)
3. Compare first record with all other records in the window
4. Increase window size while
   \[
   \frac{\text{detected duplicates}}{\text{comparisons}} \geq \phi
   \]
5. Slide the window (initial window size \( w \))
6. Calculate transitive closure
Duplicate Count Strategy (DCS)

- \( w \) = initial window size
- Increase window while \( \frac{\text{detected duplicates}}{\text{comparisons}} \geq \phi \)

Example:

- \( w = 4 \)
- \( \phi = 0.30 \)
Duplicate Count Strategy (DCS)

- $w =$ initial window size
- Increase window while \( \frac{\text{detected duplicates}}{\text{comparisons}} \geq \phi \)

**Sort order**

**Example:**
- $w = 4$
- $\phi = 0.30$
- $d/c = 0.33$
Duplicate Count Strategy (DCS)

- \( w \) = initial window size
- Increase window while \( \frac{\text{detected duplicates}}{\text{comparisons}} \geq \phi \)

Example:
- \( w = 4 \)
- \( \phi = 0.30 \)
- \( d/c = 0.25 \)
Duplicate Count Strategy (DCS)

- \( w \) = initial window size
- Increase window while \( \frac{\text{detected duplicates}}{\text{comparisons}} \geq \phi \)

<table>
<thead>
<tr>
<th>Sort order</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₁ r₂ r₃ r₄ r₅</td>
<td>( w = 4 ) ( \phi = 0.30 ) ( d/c = 0.25 )</td>
</tr>
<tr>
<td>r₁ r₂ r₃ r₄ r₅</td>
<td>( \phi = 0.30 ) ( d/c = 0.00 )</td>
</tr>
</tbody>
</table>
Enhancement of \textit{DCS}: \textit{DCS++}

- Idea 1: In addition to \textit{DCS}, for each detected duplicate the next \(w-1\) records of that duplicate are added to the window.

- Idea 2: Windows for duplicates are skipped to save comparisons
  - In example: Skip window for \(r_3\).
  - Use the transitive closure to find additional duplicates.
  - Will not miss any, because window for \(r_1\) covers all comparisons \(r_3\) would have made.
  - Assumes perfect similarity measure... Can be relaxed.
Sort order

Example:
\[ w = 4 \]
\[ \phi = 0.30 \]
Example:
\[ w = 4 \]
\[ \phi = 0.30 \]
**Example:**

\[ w = 4 \]
\[ \phi = 0.30 \]
\[ d/c = 0.20 \]

\[ \phi = 0.30 \]
\[ d/c = 0.00 \]

\( r_3 \) is duplicate of \( r_1 \)

Calculation of the transitive closure will find additional duplicates of \( r_3 \)
Skipping windows bears the risk to miss duplicates

Example: \( w=4, \phi=1/2 \)
- For \( w_1 \): \( d/c = 1/3 > \phi \)
- Thus: Window is not increased, but \( w_4 \) is left out.

Example: \( w=4, \phi=1/3 \)
**DCS++ Evaluation**

- Skipping windows bears the risk to miss duplicates

- With \( \phi \leq \frac{1}{w-1} \) no duplicates will be missed due to skipping windows

- With \( \phi \leq \frac{1}{w-1} \) \textit{DCS}++ is at least as efficient as \textit{SNM} with an equivalent window size \((w_{SNM} = w_{DCS++})\)
  - Worst case: same number of comparisons
  - Best case: \textit{DCS}++ saves \(w-2\) comparisons per duplicate
  - Proof: Next slides
Differences in comparisons

- Regard window $W_{i,}$ with $d$ detected duplicates
- Comparisons within $W(i,j)$: $c = j - i$
- Additional comparisons compared to SNM: $a = j - i - (w - 1)$
- Saved comparisons for skipped windows: $s = d(w - 1)$
- We want to show: $a - s \leq 0$

Case 1: Beginning window of $t_i$ contains no duplicate

Case 2: Beginning window of $t_i$ contains at least one duplicate
Differences in comparisons

- Additional comparisons: \( a = j - i - (w - 1) \)
- Saved comparisons: \( s = d (w - 1) \)
- Case 1: Beginning window of \( t_i \) contains no duplicate
  - No duplicates => no window increase => \( a = 0 \)
  - No duplicates => no skipped windows => \( s = 0 \)
  - \( a - s = 0 - 0 \leq 0 \)
Differences in comparisons

- Additional comparisons: $a = j - i - (w - 1)$
- Saved comparisons: $s = d (w - 1)$
- Case 2: Beginning window of $t_i$ contains at least one duplicate
  
  $a - s = j - i - (w - 1) - d (w - 1)$
  
  $= j - i - (d + 1) (w - 1)$

- Window is increased until $d/c < \phi$.
- For $\phi \leq \frac{1}{w-1}$ we need at least $c = d (w - 1) + 1$ comparisons to stop window increase

- Worst case: We find duplicate at very last comparison and increase window without any new duplicates
  
  - $c = d (w - 1) + (w - 1) (= j - i)$
  
  $a - s = j - i - (d + 1) (w - 1)$
  
  $= d (w - 1) + (w - 1) - (d + 1) (w - 1)$
  
  $= (d + 1) (w - 1) - (d + 1) (w - 1)$
  
  $= 0$
Differences in comparisons

- **Worst case**: We find duplicate at very last comparison and increase window without any new duplicates
  - \[ c = d (w - 1) + (w - 1) (= j - i) \]

- **Best case**: We find duplicate immediately after \( t_i \).
  - \[ c = d (w - 1) + 1 (= j - i) \]

- \[ a - s = j - i - (d + 1) (w - 1) \]
  - \[ = d (w - 1) + 1 - (d + 1) (w - 1) \]
  - \[ = 1 - (w - 1) \]
  - \[ = 2 - w \]

- Can save up to \( 2 - w \) per duplicate compared to SNM
Experimental Evaluation

- Perfect classifier (lookup in the gold standard)

**Algorithms**

- Sorted Neighborhood Method (*SNM*)
- Duplicate Count Strategy (*DCS / DCS++*)
- Adaptive SNM (*AA SNM / IA SNM*) (previous slides)

<table>
<thead>
<tr>
<th>Data set</th>
<th>Provenance</th>
<th># of records</th>
<th># of dupl. pairs</th>
<th>Max. cluster size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>real-world</td>
<td>1,879</td>
<td>64,578</td>
<td>238</td>
</tr>
<tr>
<td>Febrl</td>
<td>synthetic</td>
<td>300,009</td>
<td>101,153</td>
<td>10</td>
</tr>
<tr>
<td>Persons</td>
<td>synthetic</td>
<td>1,039,776</td>
<td>89,784</td>
<td>2</td>
</tr>
</tbody>
</table>
Results Cora: Comparisons

\[ \text{recall} = \frac{\text{# detected duplicates}}{\text{# real duplicates}} \]
Results Cora: Duplicate Provenance

Detected duplicates by classifying created pairs

Additional duplicates calculated by transitive closure

Duplicate detection methods compared:
- SNM – Algorithm
- SNM – Trans. Closure
- DCS++ – Algorithm
- DCS++ – Trans. Closure

Recall

0 0.95 0.96 0.97 0.98 0.99 1
Other Variants

- From Master thesis of Oliver Wonneberg
- Sorting key strategy
  - Increase window if sorting keys are similar
  - Decrease window size for dissimilar sorting keys
  - Use different sizes of increase (depending on similarity)
- Similarity strategy
  - Same as before, but based on tuple similarity
- Difficult to calibrate
Overview

- The Original
- Unique sorting keys
- Adaptive SNM
  - Part 1
  - Part 2
- Sorted Blocks
- Domain-independent SNM
**Blocking and Windowing Algorithms**

**Blocking:**

- **Sorts Neighborhood Method [HS98]:**
  - Sorting
  - Building disjoint blocks
  - Duplicate detection within blocks

**Sorted Neighborhood Method [HS98]:**

- Sorting
- Slide window over sorted tuples
- Search for duplicates within the windows
Comparing Blocking and Windowing

Window size: 3
Block size: 5

Sorted tuples

Tuples 1 & 5 are only compared using Blocking

Tuples 16 & 14 are only compared using SNM
Increasing window size to approximate Blocking
Comparing Blocking and Windowing

Overlapping blocks to approximate Windowing

Window size: 3
Block size: 5

Sorted tuples
Sorted Blocks Method

- Generalization of blocking and windowing
- Approach
  1. Sort records and build disjoint partitions
     - Sorting key might use more attributes than the partitioning predicate
  2. Perform complete comparison within partitions
  3. Overlap partitions and slide fixed size window across sorted records within overlap
  4. Calculate transitive closure
- Overlap
  - Parameter $o =$ number of records from one partition that are part of the overlap
  - Overlap size $= 2o$
  - Size of window $= o + 1$

Sorted Blocks Method

Complete comparison within partitions

1 2 3 4 5 6 7 8 9 10 11 12 13 14

Quadratic complexity
Sorted Blocks Method

Complete comparison within partitions

\[ o = 2 \]
\[ w = o + 1 = 3 \]

\[ |O_{P_1, P_2}| = 2o \]

Comparisons within overlap

Linear complexity
Sorted Blocks Method

Complete comparison within partitions

\( o = 2 \)
\( w = o + 1 = 3 \)

Comparisons within overlap

\( |O_{P_1,P_2}| = 2o \)
\( |O_{P_2,P_3}| = 2o \)
\( |O_{P_3,P_4}| = 2o \)
Sorted Blocks Configurations

1: sort records on key
2: /* initialization */
3: listComparisonRecords ← [] // List of records that are compared with the currently processed record
4: windowNr ← o+1 // Number of the window in the overlapping area
5: i ← 1
6: /* iterate over all records and search for duplicates */
7: while i ≤ records.length do
8:   if records[i] is 1st element of new partition and i > 1 then
9:     while listComparisonRecords.length > o do
10:        listComparisonRecords.remove[1]
11:     end while
12:     windowNr ← 1
13:   else if windowNr ≤ o then
14:     listComparisonRecords.remove[1]
15:     windowNr ← windowNr + 1
16:   end if
17: /* compare current record with all records in listComparisonRecords */
18:   for j = 1 to listComparisonRecords.length do
19:     compare records[i] with listComparisonRecords[j]
20:   end for
21: listComparisonRecords.append(records[i])
22: i ← i + 1
23: end while
24: calculate transitive closure

Choose o = 1 for Blocking
Choose o = w and evaluate to true for SNM
Complexity Analysis

<table>
<thead>
<tr>
<th>Method</th>
<th>Blocking</th>
<th>Windowing</th>
<th>Sorted Blocks</th>
<th>Full enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(fixed partition size)</td>
<td></td>
</tr>
<tr>
<td>Key generation</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>---</td>
</tr>
<tr>
<td>Sorting</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>---</td>
</tr>
<tr>
<td>Detection</td>
<td>$O\left(\frac{n^2}{2b}\right)$</td>
<td>$O(wn)$</td>
<td>$O\left(\frac{nm}{2}\right)$</td>
<td>$O\left(\frac{n^2}{2}\right)$</td>
</tr>
<tr>
<td>Overall</td>
<td>$O\left(n \left(\frac{n}{2b} + \log n\right)\right)$</td>
<td>$O\left(n(w + \log n)\right)$</td>
<td>$O\left(n\left(\frac{m}{2} + \log n\right)\right)$</td>
<td>$O\left(\frac{n^2}{2}\right)$</td>
</tr>
</tbody>
</table>

- $n = $ number of tuples
- $b = $ number of blocks
- $w = $ window size
- $m = $ partition size
Sorted Blocks variants

- Overall execution time for Sorted Blocks is dominated by the largest blocks
  - E.g. partitioning by city results in large partitions for Berlin, London, etc.

- Use additional parameter: max. partition size

- 2 variants with maximum partition size:
  1. Create new partition when max. partition size is reached, independently of the partition predicate
  2. Slide window when max. partition size is reached
     - Similar to the Sorted Neighborhood Method for large partitions
Experimental Evaluation

- DuDe-toolkit for experiment execution (http://tinyurl.com/dude-toolkit)
- 8 algorithms
  - Sorted Blocks – basic
  - Sorted Blocks – fixed partition size
  - Sorted Blocks – new partition when max. size is reached
  - Sorted Blocks – slide window when max. size is reached
  - Blocking
  - Sorted Neigborhood Method
    - Incrementally-adaptive SNM (IA-SNM) ¹
    - Accumulatively-adaptive SNM (AA-SNM) ¹

¹ Yan et al. (2007), Adaptive sorted neighborhood methods for efficient record linkage
Experimental Evaluation

- 3 datasets (real-world and artificial)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th>Records</th>
<th>Duplicate pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD 1</td>
<td>real-world</td>
<td>9,763</td>
<td>299</td>
</tr>
<tr>
<td>Restaurant 2</td>
<td>real-world</td>
<td>864</td>
<td>112</td>
</tr>
<tr>
<td>Address data</td>
<td>artificial</td>
<td>1,039,776</td>
<td>89,784</td>
</tr>
</tbody>
</table>

1 http://www.freedb.org
2 http://www.cs.utexas.edu/users/ml/riddle/data.html

- Varying settings for
  - overlap parameter \( o \)
  - Partition predicate
  - Max. partition size
Evaluation CD data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th>Records</th>
<th>Duplicate pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>real-world</td>
<td>9,763</td>
<td>299</td>
</tr>
<tr>
<td>Restaurant</td>
<td>real-world</td>
<td>864</td>
<td>112</td>
</tr>
<tr>
<td>Address data</td>
<td>artificial</td>
<td>1,039,776</td>
<td>89,784</td>
</tr>
</tbody>
</table>

- Sorting key: first few letters of artist, CD title, and track 01
- Partition predicate: first 1-9 letters of the sorting key
- Overlap o: 1-100
- Max. partition size: 2-1000
Evaluation CD data
Evaluation CD data
Sorted Blocks Conclusion

- Blocking and windowing are competitive approaches to reduce the number of comparisons
  - Sorted Neighborhood outperforms Blocking slightly

- Sorted Blocks is a generalization of blocking and windowing
  - Sorted Blocks outperforms Sorted Neighborhood slightly

- Experimental evaluation shows that it is superior to windowing and blocking.

- Configuration is more difficult as it has more parameters than the other 2 approaches.
Overview

- The Original
- Unique sorting keys
- Adaptive SNM
  - Part 1
  - Part 2
- Sorted Blocks
- Domain-independent SNM
Two Ideas for Domain Independence

- Domain-independent key definition
  - Define key and reverse key
  - Union-find data structure compares only representatives of each cluster
    - Relaxation of Christen-idea from before (unique sorting key)

Two Passes

- Regard each tuple as a single long string
  - Concatenate all attribute values
- 1st pass: Key = tuple
- 2nd pass: Key = reversed tuple

- “No” dependency on good key choice

- Similarity measure: Smith-Waterman
  - Suitable for long strings
Union-find Data Structure

- Interpret result as graph
  - Connected components represent duplicate clusters
  - Graph is transitive closure
- Compare next tuple only once with each connected component
  - Collection of disjoint updateable sets
  - Each set is identified by a representative
  - Initialized with $|R|$ singletons
- Union($x$, $y$)
  - Unions the sets containing tuples $x$ and $y$ to new set, deletes old sets
  - Chooses new prime representative
- Find($x$)
  - Returns unique representative of set containing $x$
- For each detected duplicate $<u,v>$:
  - If Find($u$) $\neq$ Find($v$) then Union($u$, $v$)
- Two nodes $u$ and $v$ are in same connected component $\iff$ Find($u$) $=$ Find($v$)
Union-find Data Structure

- Define **prime representative** for each detected duplicate group
- Compare records first to the representatives
  - avoiding comparisons that can be derived through transitivity.
- Similar to Swoosh idea, but records keep their identity
- If the similarity is high enough (some intermediate threshold), compare with other members of cluster
- Slight improvement: Allow multiple representatives
  - To represent large variety of tuples in cluster
Algorithm

- Priority Queue: Contains sets of tuples
  - Fixed size (≈ window size)
  - Sorted by recency of addition: Queue represents last few detected clusters
- Sort records by key (2 passes)
- For each record r
  - Test if r already part of a cluster in queue:
    - Improvement: Ignore step if first pass
    - Find(r) based on representatives
    - If successful: move cluster up in queue
    - If not successful: similarity comparison with all representatives
      - If similar:
        » Union(r,x)
        » Make r representative if not too similar
        » break
  - Else: r is new singleton cluster at top of queue
Summary

- The Original
- Unique sorting keys
- Adaptive SNM
  - Part 1
  - Part 2
- Sorted Blocks
- Domain-independent SNM