ALGORITHM TO OBTAIN TOTAL ORDER FROM PARTIAL ORDERS FOR SOCIAL NETWORKS

Chandrika Satyavolu
Advisor: Dr. Chris Pollett
Committee: Dr. David Taylor, Dr. Robert Chun
December 17th, 2008
Outline

• Problem
• Existing Solutions
• Algorithm
• Working example
• Getting back accurate total order
• Experiments to analyze space complexity
• Experiments to analyze fault tolerance
• Software design of test website: 100 best movies!
• Special features of 100 best movies!
• Comparing existing systems
• Conclusion
• References
Problem

• Develop an algorithm to obtain total order from partial orders.
• Partial orders are human-generated and cannot be produced by a computer program.
• Harness human intelligence to create more accurate total order.
• Build the algorithm into a test application: 100 best movies!
• 100 top movies of the total order is listed.
Existing Solutions

• Generate total order from user inputs using:
  – Absolute ratings
  – Absolute voting (vote for single item)
  – Multiple voting (vote for multiple items)
  – Voting and Ratings
Algorithm

QuickSort
  – Recursively sorts the input list based on the sorted partial order sequences.

Partition
  – Partitions the portion of input list into two parts. The last element of the list is chosen as the pivot element. The list is partitioned such that first part has elements that have occurred before the pivot element in most of the partial orders and the second part has numbers that have occurred after the pivot element in most of the partial orders.

GetRank
  – Gets the rank of an element in the input list with respect to the pivot element based on partial orders. If it occurs before pivot element more number of times than after in the partial orders, it returns a rank 1 otherwise it returns a rank 0.
Working Example of Partial Order QuickSort

Input list: \{3, 2, 6, 4, 7, 8\}

Output list:

\{8, 6, 7, 8\}

\{7, 8\}

\{3, 4\}

\{4\}

Partial orders:

\{2, 3, 4, 5\}

\{3, 6, 7, 8\}

\{2, 5, 7, 8\}

\{5, 6, 7, 8\}

\{5, 6\}

\{3, 4, 6\}

\{3, 4\}

\{2, 3, 4\}

\{2, 3\}

\{2\}

\{2\}

\{3, 4, 6\}

\{2, 3, 4\}

\{2, 3\}

\{2\}
Getting back accurate total order

• There must be at least one partial order comparing every pair of elements in the list.

• There must be at least one partial order comparing every element in the list to every pivot element.

• Compare number of partial orders containing one element as opposed to other. (Compare popularity of two elements.)
Experiments to analyze space complexity

- **Terminology:**
  - $n$ – total number of elements
  - $k$ – number of partial orders
  - $m$ – partial order set size

- Partial orders are stored in the database. Product of $k$ and $m$ ($k \times m$) gives space complexity of the algorithm.

- Number of partial orders is inversely proportional to partial order set size. ($k \propto 1/m$)

  ![Fig 2. Plot of $k$ vs. $m$ for $n = 100$ (k bounded by n)](image)

  ![Fig 3. Plot for $k = n$ vs. $m$](image)

- Setting $k=n$, we can prove that partial order set size is a function of total number of elements.

- Not practical to have a large partial order set size. Larger is the partial order set size, lower is the accuracy of relative rankings in the partial order.

- Set partial order set size to a practical size ($m=10$) and proceed to derive the relationship between total number of elements ($n$) and number of partial orders ($k$) required to get back the accurate total order. ($k \propto f(n)$)
Experiments to analyze space complexity (contd.)

**Terminology:**
- n – total number of elements
- k – number of partial orders
- m – partial order set size

**Experiment 1:** Derive the relationship between k and n for a fixed value of m (= 10).

Case: m=10

\[ k = C \times n \quad \{ C = 1, 2, 5, 10, 50 \} \]

Conclusion: As n increases, constant C must increase to get back correct total order. Therefore,

\[ C = f(n) \quad \text{or} \quad k = f(n) \times n \]

**Experiment 2:** Derive the functional dependency of k on n for fixed m (= 10).

Case: m=10

\[ k = f(n) \times n \quad \{ f(n) = n, n/2, \sqrt{n}, \log(n) \} \]

Conclusion: As n increases, \( f(n) = \sqrt{n} \) and \( \log(n) \) yeilds poor results. \( f(n) = n \) and \( n/2 \) always get back correct total order.

Therefore, \( f(n) = C \times n \) where \( 0 < C \leq 1 \) or

\[ k = C \times n^2 \quad \text{where} \quad 0 < C \leq 1 \]

**Experiment 3:** Derive the closest constant relating k with \( n^2 \) for a fixed value of m (= 10).

Case: m=10

\[ k = C \times n^2 \quad \{ C = 1/4, 1/6, 1/8 \} \]

Conclusion: \( k = (1/4) \times n^2 \) gives back a correct total order every single time.

\[ k = (1/6) \times n^2, \text{ give back the total order correctly too most of the time.} \]

\[ k = (1/8) \times n^2, \text{ does not give back the total order on many occasions.} \]

Therefore, \( 1/4 < C < 1/8 \) or \( C \sim 1/6 \)
Experiments to affirm fault tolerance

**Terminology:**
- n – total number of elements
- k – number of partial orders
- m – partial order set size

**Experiment:** For different values of n (300 >= n >= 20), errors are introduced into partial orders. Here:

\[ p = \text{% of error introduced.} \]

i.e p % of partial orders have single error in them.

**Observations:**
- A small percentage of error (p <= 10%) introduced cannot deduce the fault tolerance property of the algorithm.
- As the error introduced is a two-element error, there is a good possibility of getting back the correct total order even if the error is large.
- Even a 100% two-element error introduced had no greater than 5% chance of the total order being incorrect.

<table>
<thead>
<tr>
<th>% error in 30 values of n</th>
<th>Number of ‘n’ values giving incorrect total orders (Total no. of values = 30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>p = 5% error</td>
<td>2</td>
</tr>
<tr>
<td>p = 10% error</td>
<td>3</td>
</tr>
<tr>
<td>p = 20% error</td>
<td>4</td>
</tr>
<tr>
<td>p = 50% error</td>
<td>4</td>
</tr>
<tr>
<td>p = 75% error</td>
<td>10</td>
</tr>
<tr>
<td>p = 90% error</td>
<td>10</td>
</tr>
<tr>
<td>p = 100% error</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 1. Error introduced vs. Fault Tolerance
The software components and the functions associated with them are listed in Table 2.

Non-registered users can only view homepage listing top 100 movies.

Only registered users can use functionalities of 100 best movies!

User must be signed in to:
- Add movie
- Rank movie
- Remove Movie

<table>
<thead>
<tr>
<th>Components</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>register, signin</td>
</tr>
<tr>
<td>Movie</td>
<td>add, rank, remove, group, select</td>
</tr>
<tr>
<td>Group</td>
<td>create, select, modify, rank</td>
</tr>
<tr>
<td>Partial Orders</td>
<td>create, sort</td>
</tr>
</tbody>
</table>

Table 2. Software Components
Software design of test website: 100 best movies!

- Movies, Users and Groups are unique.
- Group may or may not be saved while creating a partial order.
- Each user can request for a movie removal only once.
- Partial order stored as unique 4-tuples: order_id, user_id, movie_id, rank

Fig 4. Entity Relationship diagram.
Special features of 100 best movies!

• Partial order sorting is performed over database.
• Variable partial order set size allowed.
• User can influence removal of movie.
• Ranking techniques allowed by 100 best movies!:
  – Sort a set
  – Sort a set and save it as group
  – Select existing group and sort
  – Select existing group, add and/or remove movies and sort
  – Select existing group, add and/or remove movies, sort and save set as new group
Comparing existing systems

- Case 1: Total order is derived from voting and user ratings.

Fig 5. www.imdb.com

Top 250 movies as voted by our users

For this top 250, only votes from regular voters are considered.
Comparing existing systems (contd.)

- Case 2: Total order is derived from multiple voting.
Comparing existing systems (contd.)

Comparing 100 best movies! and rankrz.com with an example case:

Input list: 5, 8, 1, 2, 4, 6, 10, 3, 7, 9
User input: 1, 2, 3, 4, 5, 6
1, 2, 3, 4, 5, 7
1, 2, 3, 4, 5, 8
1, 2, 3, 4, 5, 9
1, 2, 3, 4, 5, 10
5, 6, 7, 8, 9, 10

Note: 100 best movies! sees the user inputs as relative partial orders whereas rankrz.com sees the user inputs as list movies users voted for.

100 best movies! total order: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
rankrz.com total order: 5, 1, 2, 3, 4, 6, 7, 8, 9, 10
Conclusion

• Partial Order QuickSort algorithm sorts elements that are not possible to be ordered by a computer program.
• It harnesses human intelligence.
• It generates an accurate total order.
• Our work can be extended to have the total ordering done by variation of other sorting algorithms. The performance of using different sort algorithms can be compared and documented.
References

[8] www.rankrz.com
Thank you.

Q&A