Optimal Splitters for Temporal and Multi-version Database

Wangchao Le, Feifei Li, Yufei Tao, Robert Christensen

Motivation and Problem Formulation

Temporal and multi-version data used extensively in:
- financial market
- scientific applications
- data warehousing

![Optimal Splitters](image)

Score

Segmentation

Temporal database

An example, \( m = 2 \)

Value

An object with 3 versions

Multi-version data

Baseline Method – Dynamic Programming

Given a splitter \( l \) and a set of intervals \( I \) stored in an array.

![Baseline Method](image)

Algorithm cost: \( O(kN^2) \)

Internal Memory Method

Sketch of Cost-\( t \) algorithm

1. Optimal cost \( t^* \) is in range of \( R = [1, N] \)
2. Binary Search on \( R \)
   - if \( t \) is infeasible, then any \( t' < t \) is also infeasible
   - Solve \( O(N) \) instances of Cost-\( t \) splitters problem
3. Report \( t^* \) as result when \( t \) is feasible but \( t' < t \) is infeasible

Cost Analysis

sort disk-resident array of \( O(N/B) \) blocks

\( \text{Cost to find } t^* = 3 \times (\text{Cost of Concurrent Cost-} t \text{ jumps}) + \) \( \text{Cost of construction of stabbing count array} \)

Concurrent Cost-\( t \) jumps:

- \( t \)-jump scans forwardly, next block to be read is uniquely defined.
- One execution requires \( O(1) \) space.

Stabbing Count Array

- Sort disk-resident array of \( O(N/B) \) blocks
- \( \# \) rounds of Concurrent Cost-\( t \) tests:
- \( \# \) rounds of Concurrent Cost-\( t \) tests:
- \( \text{Cost to find } t^* = 3 \times (\text{Cost of Concurrent Cost-} t \text{ jumps}) + \) \( \text{Cost of construction of stabbing count array} \)
- Retrieve the optimal splitter

Experimental Results

![Experimental Results](image)

Conclusion

Temporal and multi-version databases often generate massive amounts of data. Therefore, it becomes increasingly important to store and process this data in a distributed and parallel fashion. This makes an important contribution in solving the optimal splitters problem, which is essential in enabling efficient distributed and parallel processing of such data.