Multi-way Similarity Joins

Yasin Silva, Spencer Pearson, Jaime Chon, Ryan Roberts

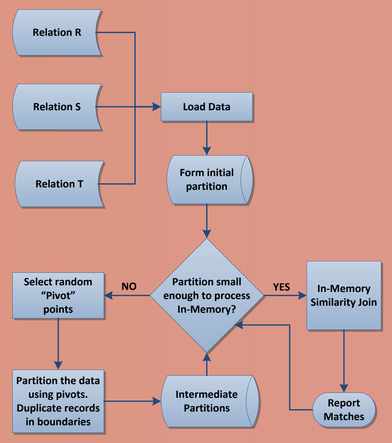
The Problem

- Similarity join queries are a key data analysis tool that identifies pairs of similar records
- Applications in:
  - Marketing
  - Multimedia and Video Applications
  - Sensor Networks
- Very little research addresses the problem of similarity joins among more than two datasets
- The support of Similarity Join for multi-dimensional data is key because this data type is extensively used to analyze complex objects:
  - Images
  - Videos
  - Geographical data

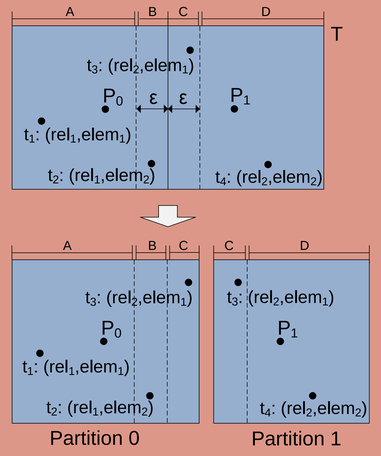
Our Contributions

- The design of i-MSimJoin: Index-based Similarity Join algorithm (high performance for pre-defined ranges of similarity)
- The design of p-MSimJoin: On-the-fly Similarity Join algorithm (can be used with any similarity values)
- Both algorithms support any dataset in a metric space
- Implementation of both algorithms using the C++ programming language
- Performance evaluation of the implemented algorithms

p-MSimJoin: Partition-Based SimJoins



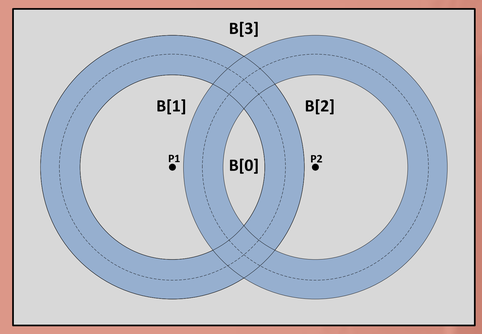
Overall Process



Partitioning Example

- Our algorithm recursively partitions the data on-the-fly using a set of reference points (pivots) for each partition
- Each point is assigned to the partition of its closest pivot
- Points in the ε-windows are duplicated
- The first round partitions the input data. All partitions too large to be processed immediately are stored on-disk
- Additional rounds re-partition partitions that have been stored on-disk
- There is no need to build index structures for the data

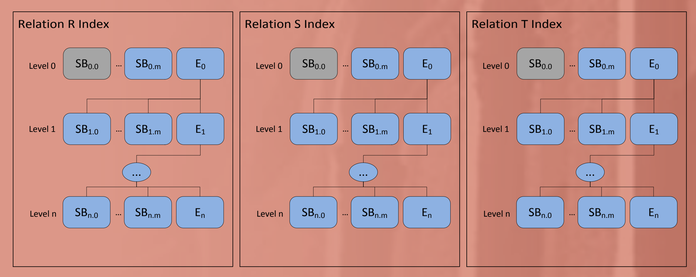
i-MSimJoin: Index-Based SimJoins



Gray areas:
Separable Buckets

Blue area:
Exclusion Set

- Indexes are powerful tools for improving performance
- The D-Index and eD-Index are efficient indexing structures for processing similarity queries
- Separable buckets partition tuples so that dissimilar tuples are in different buckets



- Construct identical indexes for each relation
- These can be used to process Similarity Join queries between the relations by traversing them synchronously
- At each combined bucket, use a sliding window algorithm to identify similar tuples

ASU Arizona State University