Index-Based Similarity Joins
Yasin N. Silva, Spencer S. Pearson
Arizona State University

The Problem
• Similarity joins are a key tool in analyzing and processing data.
• Some standalone Similarity Join algorithms have been proposed.
• Little work on implementing Similarity Joins as physical database operators has been done.

Our Contribution
• Modification of the D-Index to allow for similarity joins
• Implementation of a similarity join algorithm utilizing successive searches through the index
• Implementation of a similarity join algorithm in one pass through combining indexes

Our Approach – Building the Index
• Build a common index structure for multiple datasets
  • Each level uses the same pivot points
  • Therefore the same bucket structure is generated
• The buckets from each index can then be accessed simultaneously

Our Approach – Querying
• Load both relations using the common index
• Traverse through all buckets in each index simultaneously, reporting matches between the relations in each bucket

D-Index Structure
• D-index is structured as an array of buckets
  • For a level \( m \), \( m \) has \( n_m \) buckets
    • \( n \) can be different for each level
  • Each exclusion set is used to build the next level
• The exclusion set of the final level is not partitioned any further

D-Index Bucket Addressing
• D-index consists of multiple levels
  • Levels are partitioned into \( n \) separable buckets and one exclusion set
  • Objects in a separable bucket cannot be within \( p \) of another separable bucket, where \( p \) is the parameter used to build the d-index
  • All objects not inside a separable bucket are placed in the exclusion set

Separable Buckets:
• \( B[0] \)
• \( B[1] \)
• \( B[2] \)
• \( B[3] \)
Exclusion set is everywhere not included in the separable buckets (the blue areas)

Partitioning Example

D-Index Bucket Addressing