Vector spaces are back

Prabhakar Raghavan

Yahoo! Research

35 years of document retrieval

- Given a corpus of *m* documents, over a vocabulary of *n* terms
- View each doc as a (unit) vector in *n* dimensions, one for each term
  - j^{th} component of each vector is the *weight* of term *j* in doc
- Immediately induces a measure of proximity between docs – cosine

Queries and docs

- View a query also as a unit vector in this vector space
  - Induces a proximity measure between query and each doc – aka score
- Used for ranking

Technical challenge

- Ranking entails computing cosine similarity from query to every doc
- Slow – *m* dot products in *n* dimensions
  - "Unavoidable" in the worst case
  - So … heuristics

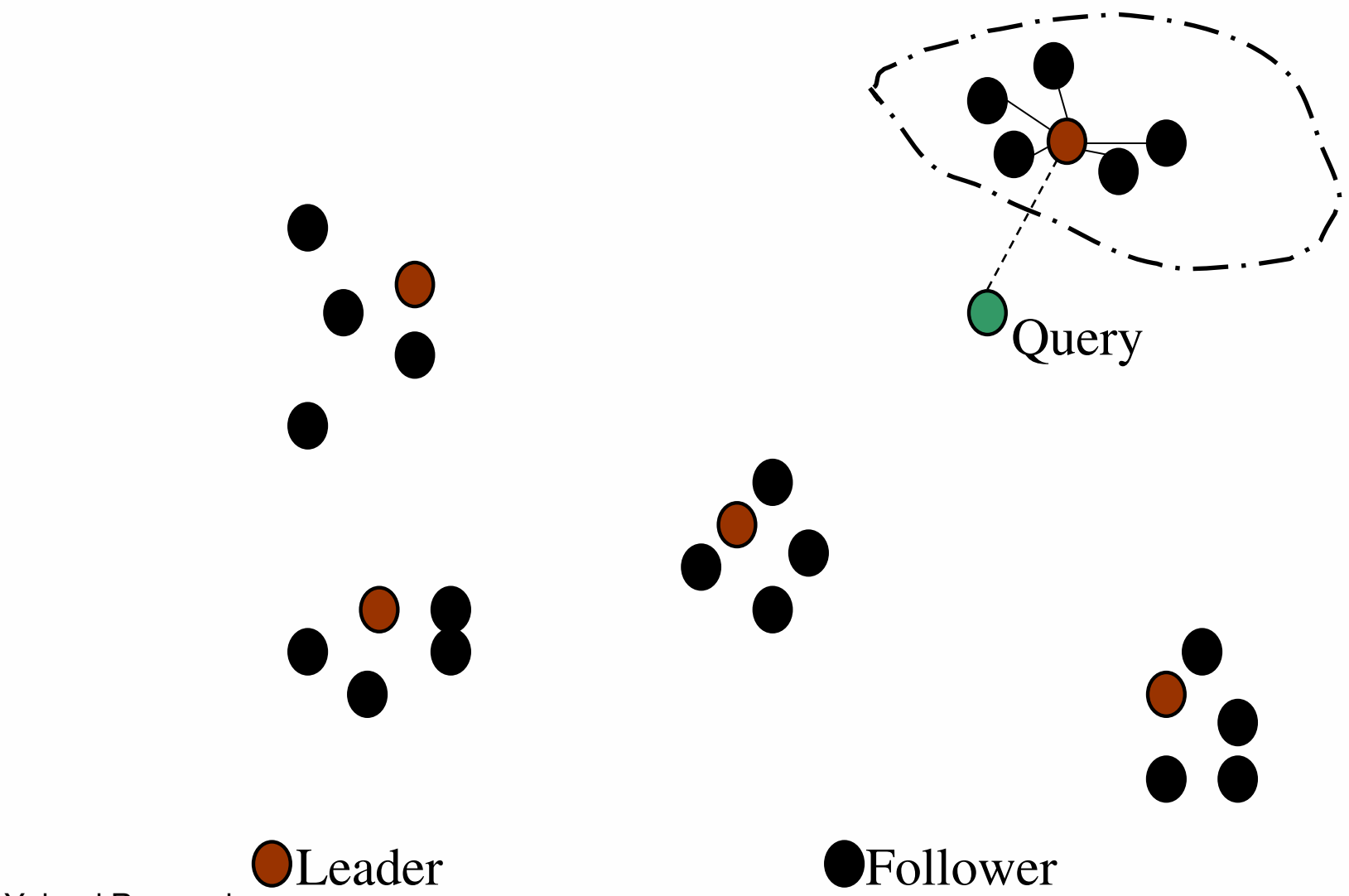
Cluster pruning: preprocessing

- Pick √m *docs* at random: call these *leaders*
- For each other doc, pre-compute nearest leader
  - Docs attached to a leader: its *followers;*
  - Likely: each leader has ~ √*m* followers.

Cluster pruning: query processing

- Process a query as follows:
  - Given query *Q*, find its nearest *leader L*.
  - Seek *k* nearest docs from among *L*'s followers.

Visualization



Why use random sampling

- Fast
- Leaders reflect data distribution

General variants

- Have each follower attached to a=3 (say) nearest leaders.
- From query, find b=4 (say) nearest leaders and their followers.
- Can recur on leader/follower construction.

Empirical study

- Sitarama et al. WWW 2003
- Varied parameters, studied retrieval effectiveness

Multiple vector spaces

- In settings like XML and multimedia retrieval – have multiple vector spaces
- E.g., a query for Author, a query for Title and a query for Abstract

Efficiency Quality Tradeoffs in Vector Score Aggregation

VLDB 2004

Joint w/ Pavan Kumar C. Singitham†

Mahathi S. Mahabhashyam†

Microsoft Inc.

† Work done while at Stanford University.

Outline

- Problem
- Motivation
- Retrieval methods
  - Resource Allocation (two schemes)
  - Cell Decomposition Index
- Performance
- Summary

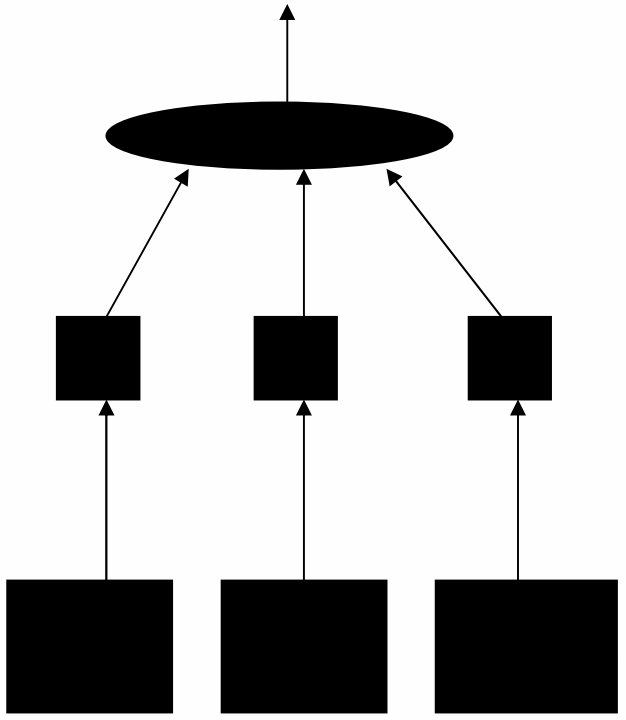
Problem: Score Aggregation

Given *n* objects

- Each represented by a set of *s* vectors, one for each *field*.

- Weighted queries
  - A query is a set of *s* vectors, plus a *weight* for each field.

- Score aggregation
  - Find *k* objects nearest the query, by weighted sum of field-wise cosine score.



Here *s*=3.

Motivation

- QBIC (Query By Image Content)
  - Image consists of fields – color, texture, hue, etc.
  - Query specifies relative importance of these fields.

Motivation

- Semi-structured retrieval for text and XML
  - E.g. Query weighted by a user:
    - 0.67* (Author = ‘Aho’) + 0.33*(Title=‘Algorithms’).
  - Overall score a weighted combination of Author, Title scores.

Key issue

- Best known method for Score Aggregation: Exhaustive search over entire collection. Costly!
- (Special case *s*=1 is standard nearest neighbor search.)

Multiple Vector Spaces: $s>1$

- Objective
  - Obtain approximate top-k results with minimal computational effort.
  - Avoid exhaustive search.
- Here: Two approaches based on cluster pruning
  - Resource Allocation.
  - Cell Decomposition.

Metrics – 1

- *Quality* of a method measured using two metrics: compare to *ground truth*
  - Aggregate Goodness
    - Sum of top *k* scores compared to the top *k* from the ground truth.
  - Competitive Recall
    - Number of objects from the ground truth that are in the top *k* from the method.

Metrics – 2

- *Computational cost*
  - Assume all vectors are unit vectors.
  - Thus "proximity" is cosine similarity.
  - Cost = # of query-to-document cosine similarity calculations.

    (In the worst case, *ns*).

Experimental Setup

- 480,000 documents – obtained by crawling citeseer.
- Clustered using *L*-means into 512 clusters
  - Independently in each field
- Each cluster is represented by its centroid – containing the “heaviest” 3000 terms.

Resource Allocation Schemes

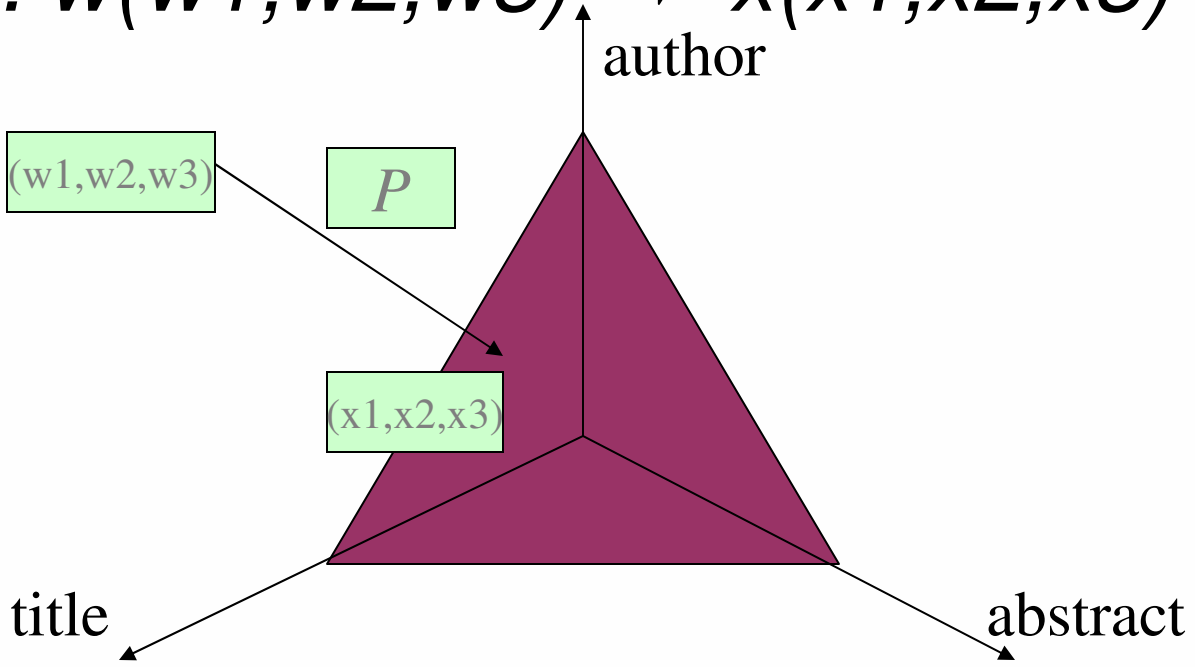
Terminology: *probe*

- Probing a cluster = computing similarity from query to every vector in a cluster
  - This is in a single field

Resource Allocation method

- Fixed budget of cluster probes, *B.*
- A portfolio allocation

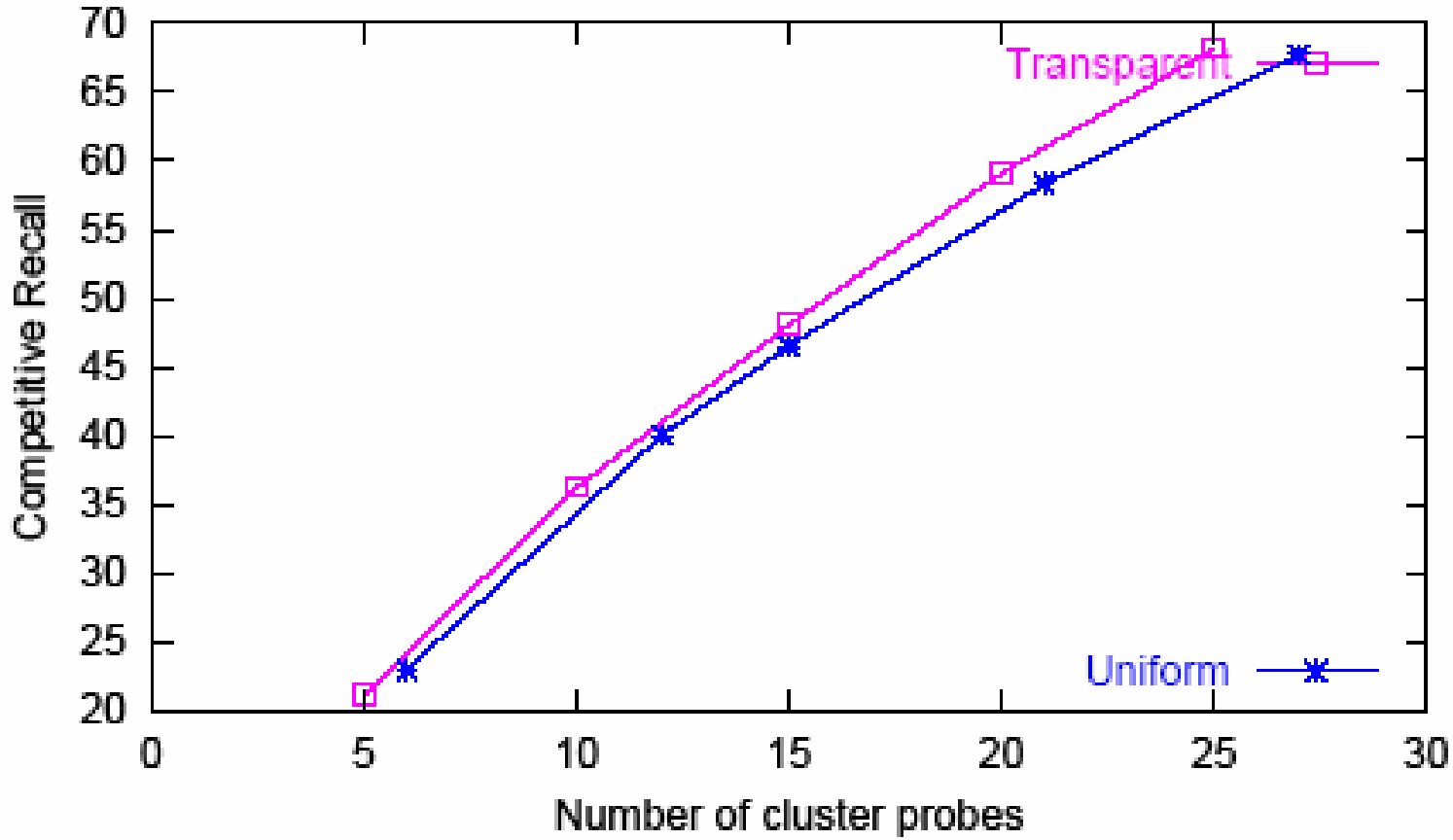
P: w(w1,w2,w3) ➡ *x(x1,x2,x3)*



Simple allocation schemes

- Uniform allocation
  - Probe equal number of clusters in all fields – indep of *w1,w2,w3.*
- Transparent allocation
  - Number of clusters probed proportional to query weights.

Transparent betters Uniform

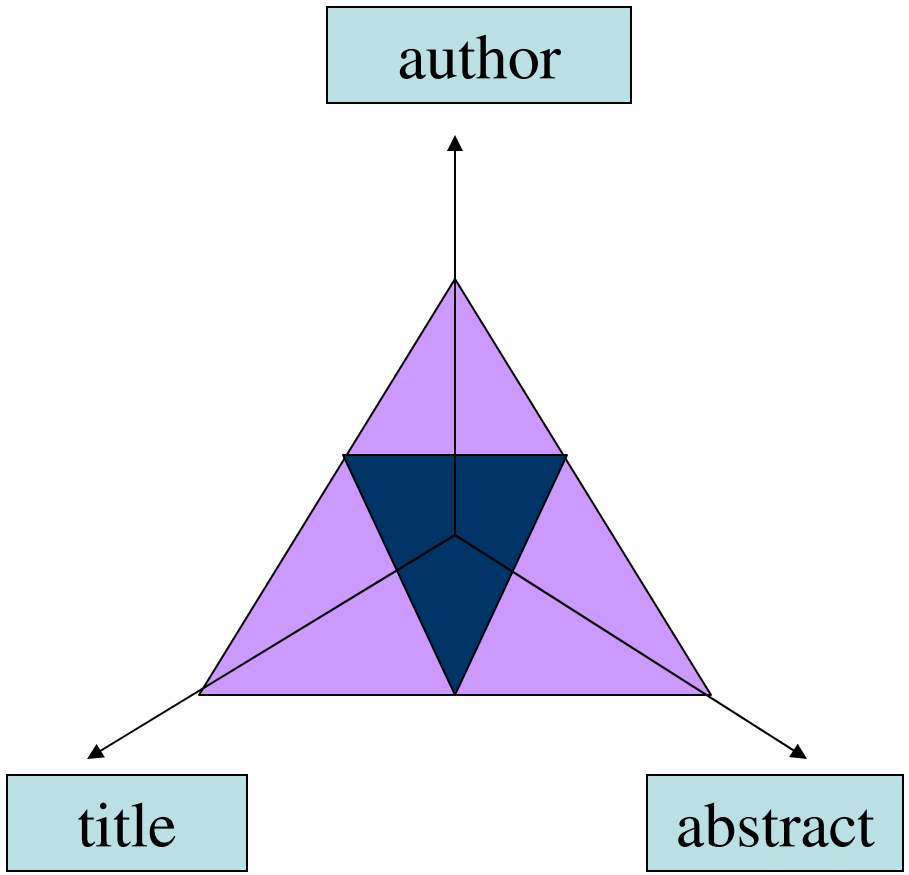


But is transparent the best?

- Could there be more sophisticated (nonlinear) mappings from the weights to the portfolio allocation?

Cell decomposition

Cell Decomposition Index



One gigantic space, treating each document as a bag of *field-words*

Axes look like

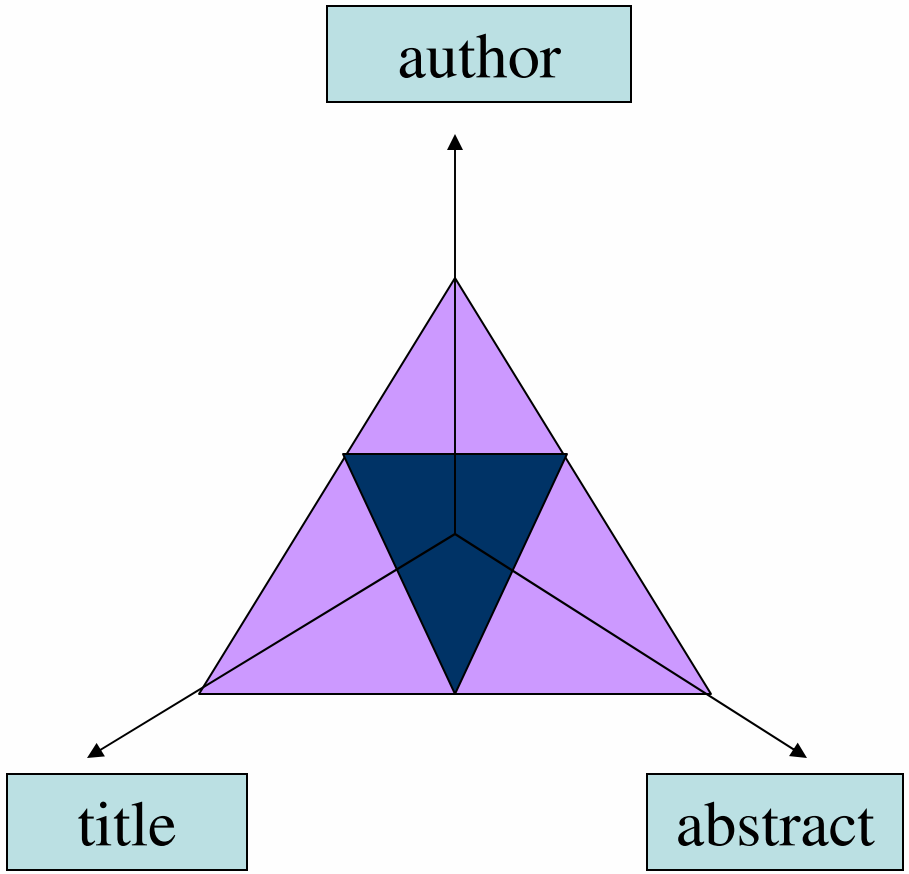
author.aho

author.bill

title.bill

body.bill…

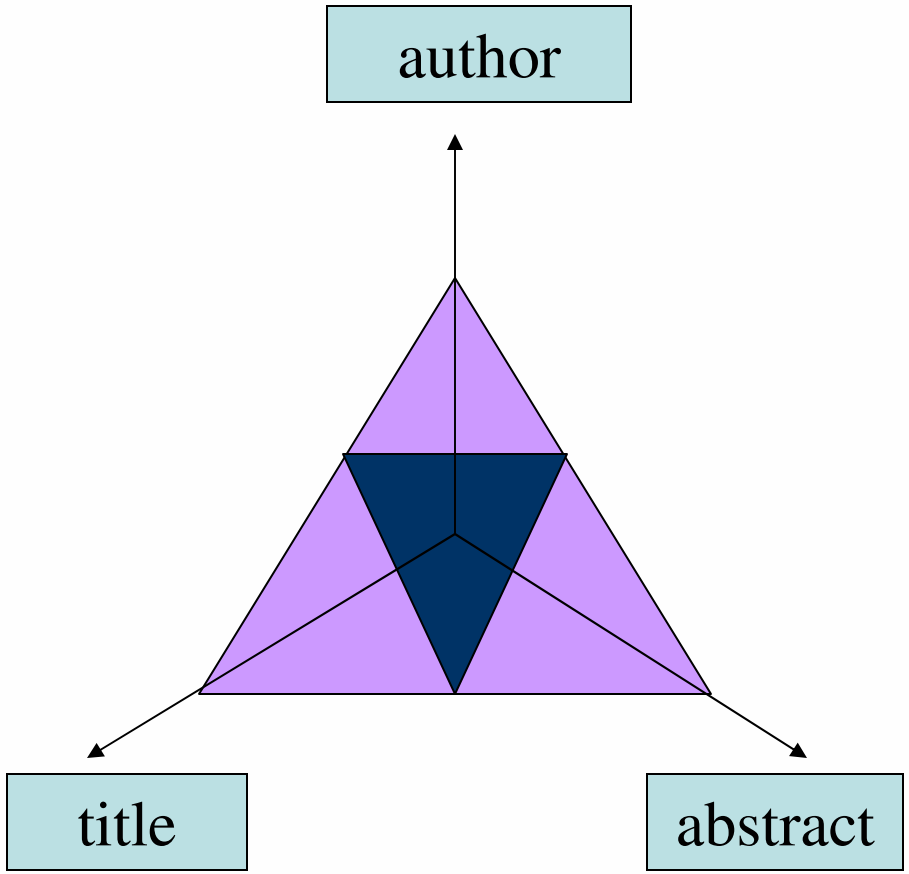
Cell Decomposition Index



Partition this space into cells within which the answer to any query “the same”

Build a cluster pruning index for each cell

Cell Decomposition Index



Given a query: map it into this big space

Search only the index for the cell containing the query

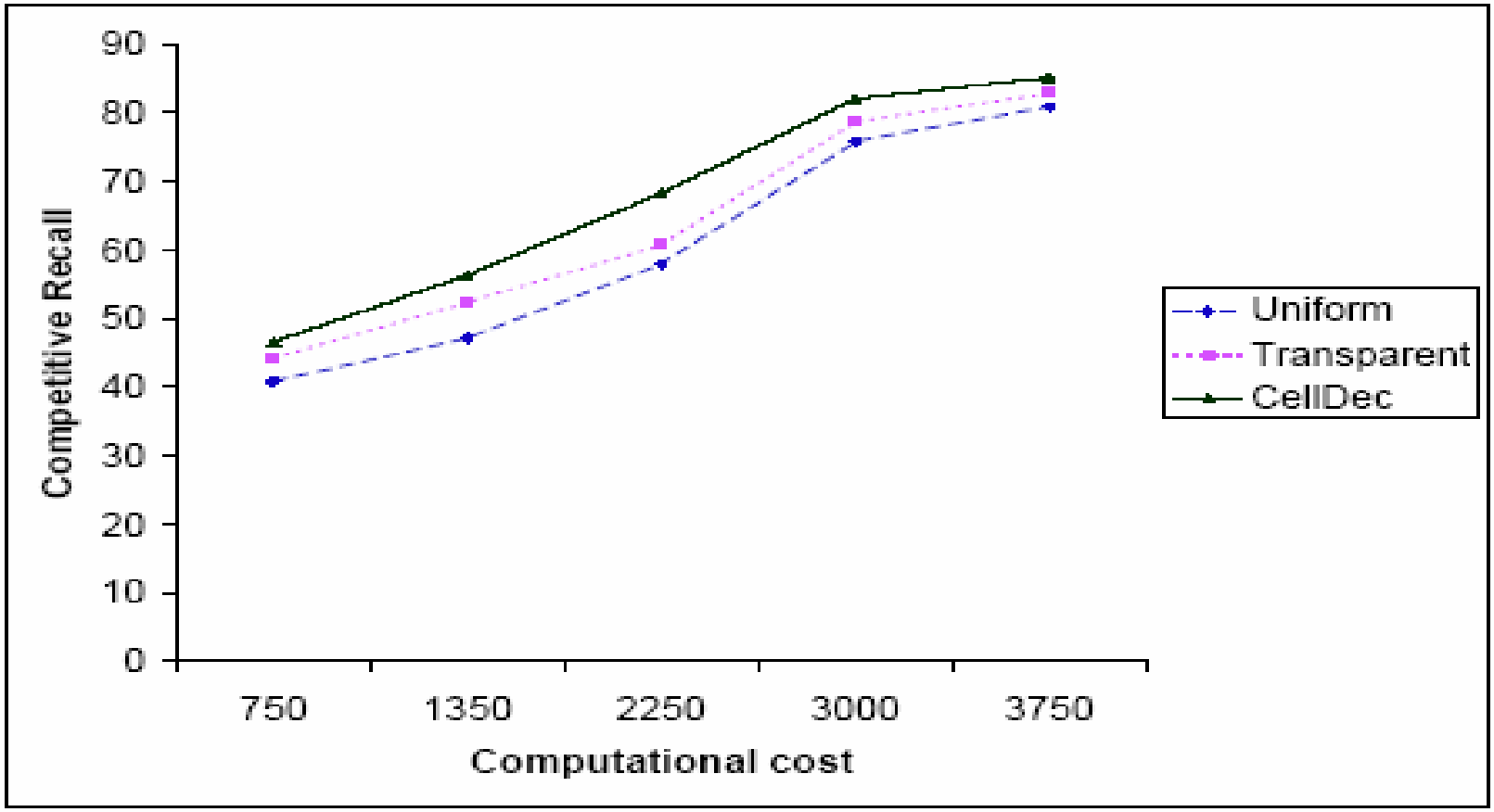
Experiments

- For *s*=3, broke big space up into 4 cells
  - Equilateral simplexes:
    - Closest to author
    - Closest to title
    - Closest to abstract
    - In the middle

Experiments

- Given query weights *w1,w2,w3,* first isolated cell
- Within each cell, used a fixed skew of fields relative to each other

Cell decomposition vs. allocation



Summary

- Can obtain high quality results with low effort compared with exhaustive search.
- Scope for more sophisticated portfolio allocation schemes.
- More sophisticated cell decomposition schemes
  - Deciding which indices to materialize
  - Which indices to choose for a given query
- How should the algorithm parameters be data dependent?

Other work

- With V. Kakade, ECIR 2005:
- Encode XML documents into vector spaces
- Each element in an XML doc becomes a vector
- Allows vector retrieval at element level


Questions?