Web Data Benchmarking

By Tham Manjing

Contents

Overview

- An Empirical Study of Real-World SPAQL Queries
  - Mario Arias Gallego
  - Javier D. Fernández
  - Miguel A. Martínez-Prieto
  - Pablo de la Fuente
- The paper focuses on analyzing real-world queries
  - Log from the USEWOD2011 Challenge
  - Frequency of usage of different SPARQL queries
  - The difference in queries among different data sources
  - Patterns that users use in the real-world

Overview

- Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets
  - The flaws in current benchmarking datasets
  - Experiment using various real and benchmark datasets
  - Comparison of real and benchmark structuredness
  - Computation of coherence
  - Proposing a new benchmark generation algorithm

Real-World Queries - Setup

- USEWOD2011 provides
  - 5 million DBPedia queries
  - 2 million Semantic Web Dog Food (SWDF) queries
- Remove duplicated queries generated from the same host
- Remove queries which result in parsing error

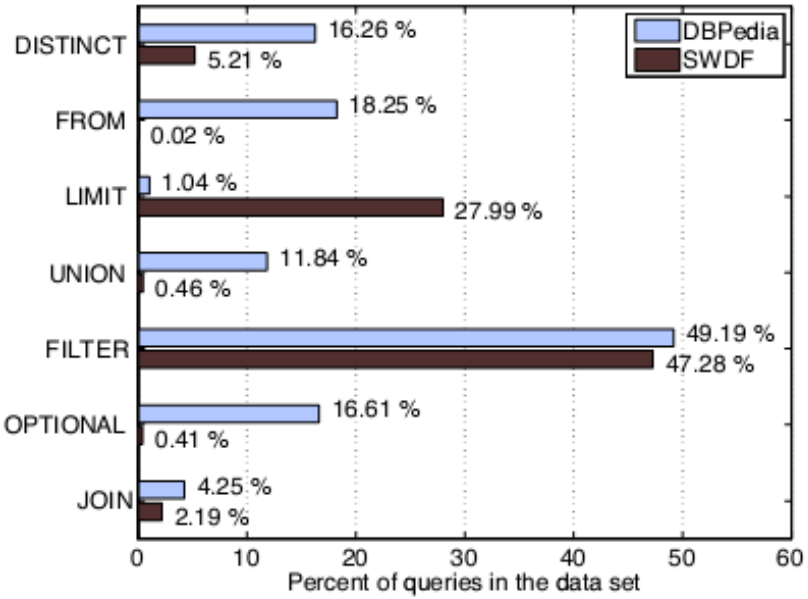
	DBPedia	SWDF
Total Queries	5 166 272	2 062 508
Duplicates from same host	51.7%	69.8%
Parse error	4.37%	1.03%
Analyzed	43.9%	29.1%

Most frequently used types of query

	DBPedia	SWDF
SELECT	96.9%	99.7%
ASK	1.6%	0.2%
CONSTRUCT	1.5%	0.01%
DESCRIBE	0.002%	0.002%

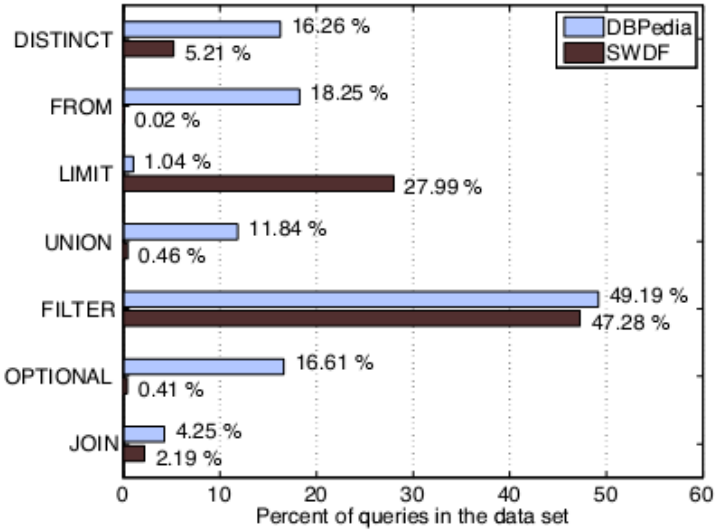
Queries using the different SPARQL features

- FILTER is the most used since it greatly reduces the space of RDF triples needed to be explored
- 99.4% of the filters only affect one variable
- LANG occurs 28% in DBPedia but none in SWDF because SWDF is published solely in English



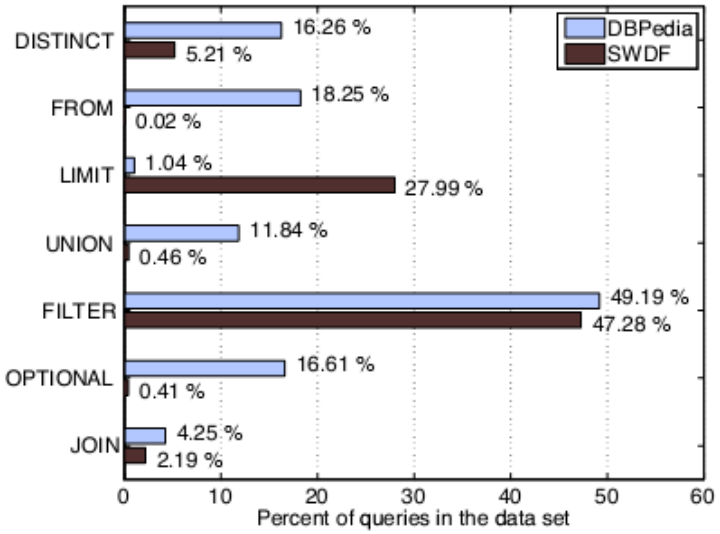
Queries using the different SPARQL features

- Equal comparator (DBPedia - 23%) which holds the first position in SWDF (93%)
- DISTINCT is more popular on DBPedia than SWDF, perhaps because of the complexity of its schema
- SELECT REDUCED is found only twice, therefore RDF store designers should not rely on users using this modifier.



Queries using the different SPARQL features

- The FROM feature is widely used on DBPedia, which is composed by several data sources, but it is not used on SWDF
- ORDER BY, GRAPH, FROM NAMED and OFFSET which occur less than 0.5%
- UNION appears in 11.84% of DBPedia queries



Triple Patterns

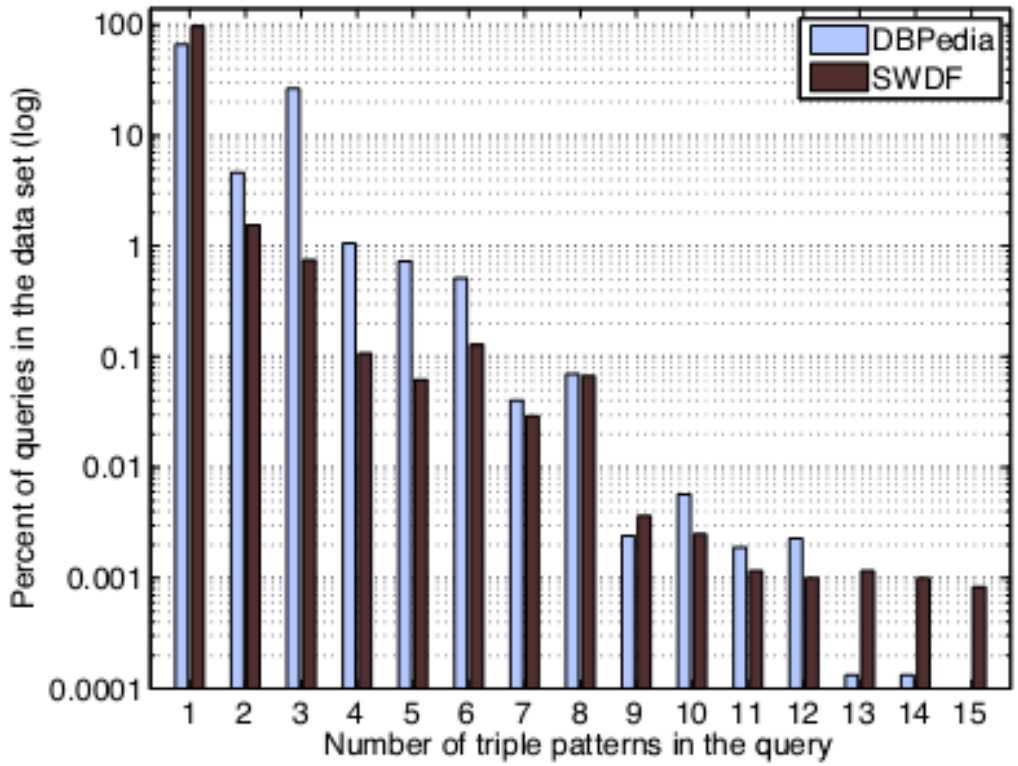
Pattern	DBPedia	SWDF
C C V	66.35%	47.79%
C V V	21.56%	0.52%
V C C	7.00%	46.08%
V C V	3.45%	4.21%
C C C	1.01%	0.001%
V V C	0.37%	0.19%
C V C	0.20%	0.006%
V V V	0.04%	1.18%

Triple Patterns

- A comparison of DBPedia and SWDF shows a significant difference suggesting that the usage of triple patterns is highly-dependent on the kind of information provided and its structure
- Since C C V is a very common access operation, we can foresee that a multi-field index on (Subject-Predicate) would significantly improve search performance

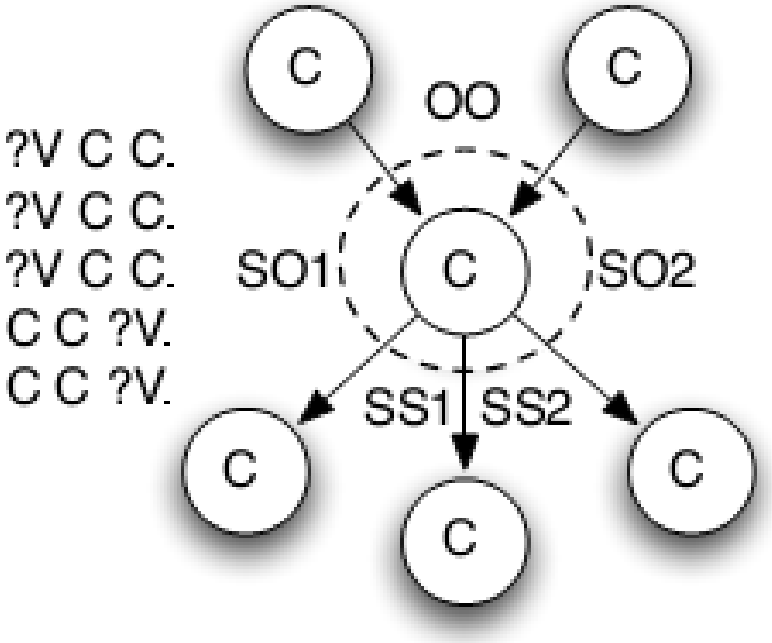
Triple Patterns

- Most of the queries contain one single triple pattern (66.41% in DBPedia, 97.25% in SWDF)
- Note that the figure is in logarithmic scale

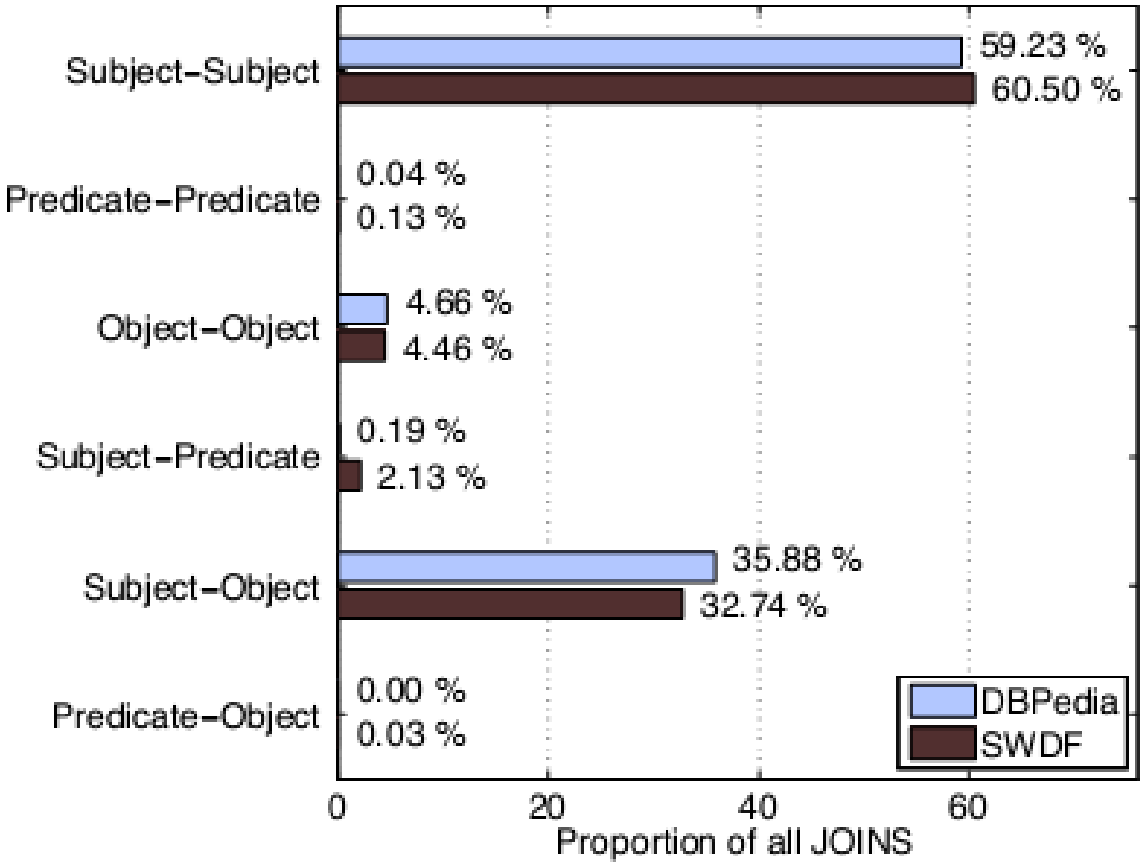


Join Types

- Six types of joins:
  - Subject-Subject
  - Predicate-Predicate
  - Object-Object
  - Subject-Predicate
  - Subject-Object
  - Predicate-Object

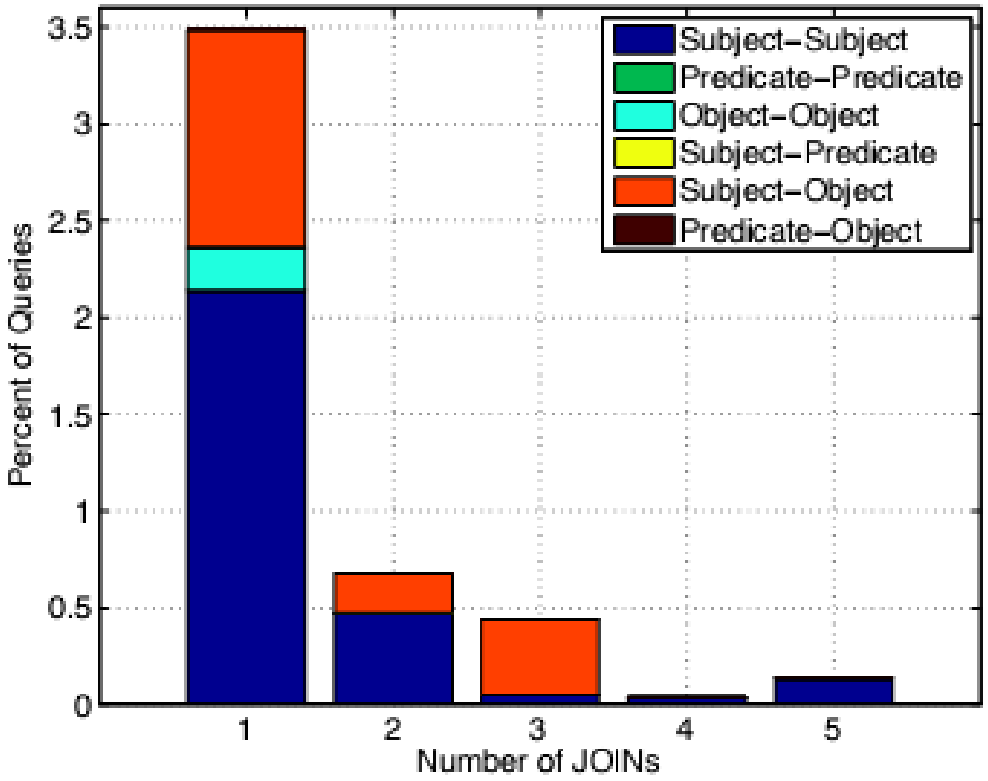


Join Types


Subject–Subject
59.23 %
60.50 %
Predicate–Predicate
0.04 %
0.13 %
Object–Object
4.66 %
4.46 %
Subject–Predicate
0.19 %
2.13 %
Subject–Object
35.88 %
32.74 %
Predicate–Object
0.00 %
0.03 %
DBPedia
SWDF
0
20
40
60
Proportion of all JOINS

Join Types

- 2.66% of the total queries in DBPedia have a single join
- 0.75% have two joins
- In all, 4.25%



Graph Patterns

- Neumann and Weikum claimed in “The RDF-3X engine for scalable management of RDF data” that SPARQL graph patterns are typically star-shaped or include long-chains
- A directed graph have been constructed to each query
  - 98% of both DBPedia and SWDF queries had a length of 1
  - 1.8% had 2
  - very few queries had up to 5 jumps
  - Thus, there conclude that graph pattern do include chains but they are very scarce
- Note that these data came from only two sources

Graph Patterns

- In order to characterize whether pattern graphs have star shapes, they propose using a serialization of the out-degree of each node of the graph in decreasing order
- Star-shaped graphs will have a central node with a high out-degree, and several leaf nodes with null out-degree (For instance: 3 0 0 0)

Graph Patterns

- Most common is the most simple one with one single triple, which might be considered a trivial star
- There is a big proportion of appearances (more than the 99.93%) of almost-star-shaped graph structures between 3 and 9 nodes.

Pattern	DBPedia	SWDF
1 0	66.512%	97.463%
3 0 0 0	26.683%	0.106%
2 0 0	3.773%	1.024%
1 1 0	1.371%	0.482%
5 0 0 0 0 0	0.701%	0.010%
2 1 0 0	0.313%	0.432%
3 1 0 0 0	0.195%	0.040%
4 0 0 0 0	0.179%	0.020%
6 0 0 0 0 0 0	0.107%	0.001%
8 0 0 0 0 0 0 0 0	0.068%	0.000%
6 1 0 0 0 0 0 0	0.029%	0.001%
<Others>	0.07%	0.420%

Real-World Queries - Summary

- The research provides a statistics view of how real users use SPARQL.
- RDF designers can use this information to optimize the frequently used queries
- The structure of the data influences how users query
- Information from just two data sources is not enough to justify the SPAQL query patterns as a whole!!

Real and Benchmark Comparison- Setup

- Dbpedia (Real):
  - Extracted structured information from Wikipedia
  - Entities stored in the triples come from a wide range of data types, (Person, Film, Music, Place, etc.)
  - 150 million triples (22 GB)
- UniProt (Real):
  - information about proteins
  - 1.6 billion triples (280 GB)
- YAGO (Real):
  - knowledge from both Wikipedia and Wordnet
  - 19 million triples (2.4GB)

Real and Benchmark Comparison- Setup

- Barton Library Dataset (Real):
  - converted data from the Machine Readable Catalog (MARC) of the MIT Libraries Barton catalog to RDF
  - 45 million RDF triples (5.5 GB)
- WordNet (Real):
  - information about English Language
  - 1.9 million triples (300MB)
- Linked Sensor Dataset (Real):
  - Descriptions of approximately 20,000 weather stations in the United States
  - 500 million triples (101 GB)

Real and Benchmark Comparison- Setup

- TPC-H (Benchmark):
  - Well-known decision support benchmark in relational Database
  - They used the DBGENTPC-H generator to generate a TPC-H relational dataset of approximately 10 million tuples
  - Then, they used D2R tool to convert the relational dataset to the equivalent RDF one.
  - 130 million triples (19 GB)

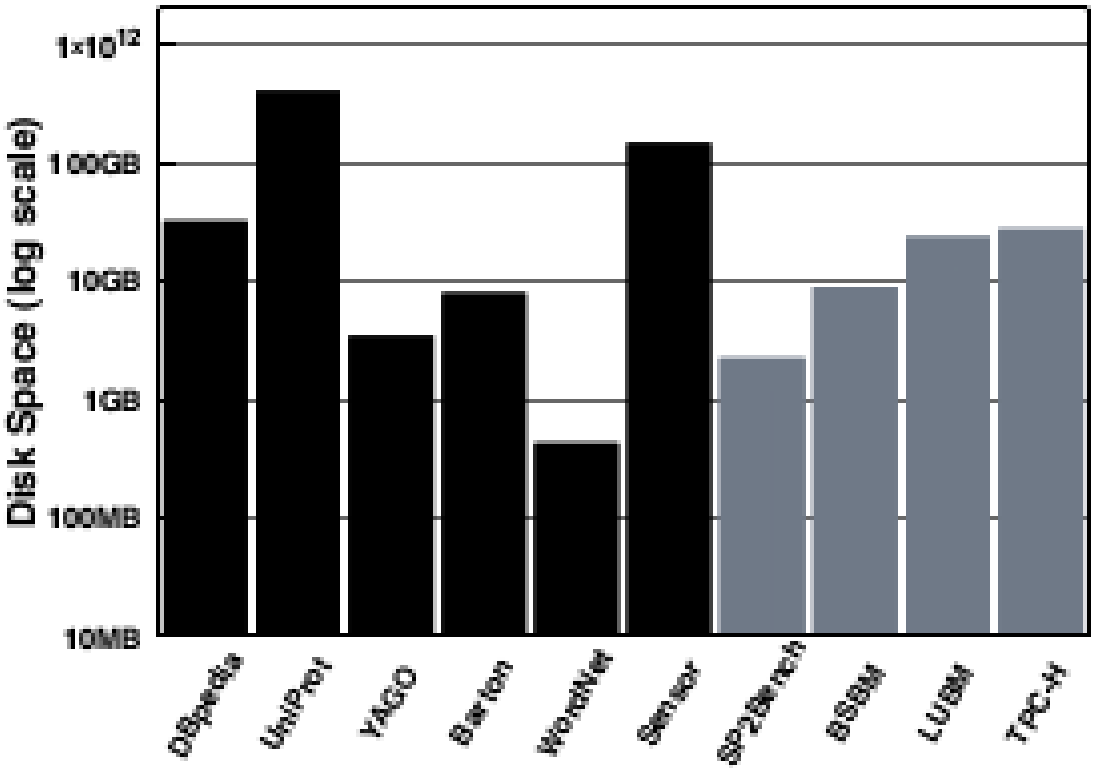
Real and Benchmark Comparison- Setup

- BSBM Dataset (Benchmark):
  - Berlin SPARQL Benchmark considers an e-commerce domain
  - Types Product, Offer and Vendor are used to model the relationships between products and the vendors offering them
  - Types Person and Review are used to model the relationship between users and product reviews these users write.
  - 25 million triples (6.1 GB)

Real and Benchmark Comparison- Setup

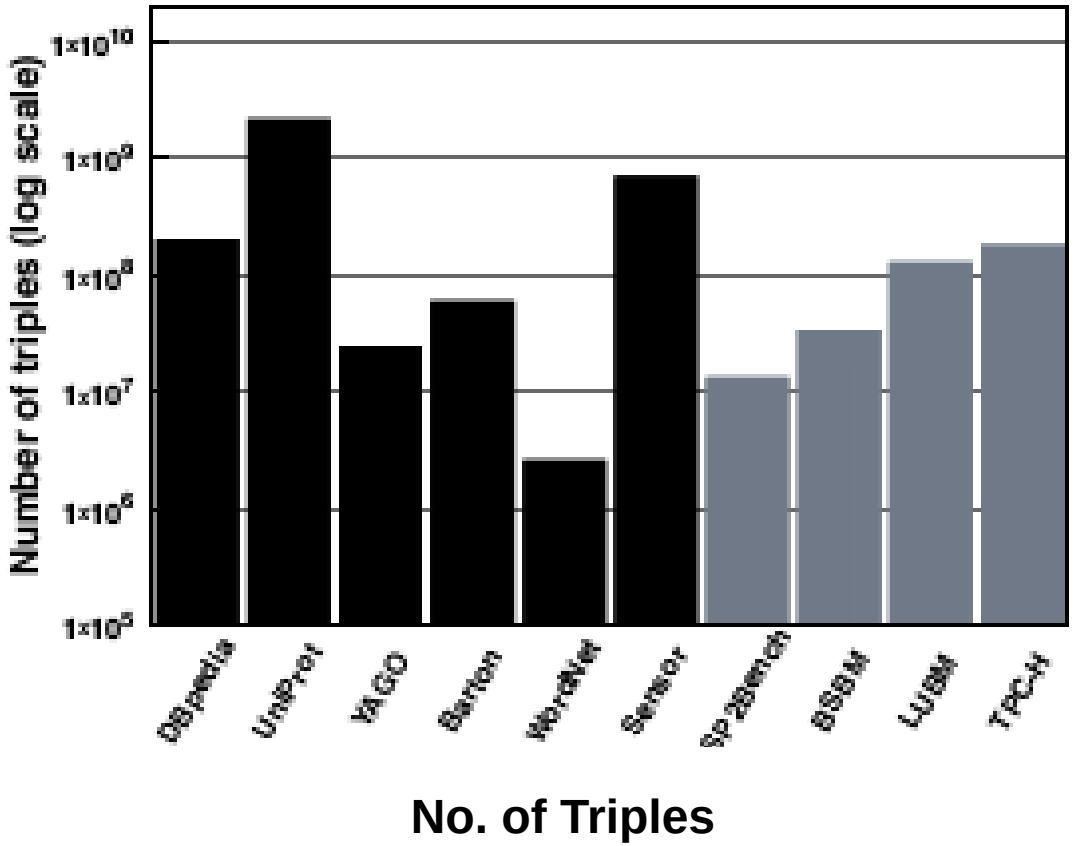
- LUBM Dataset (Benchmark):
  - The Lehigh University Benchmark considers a University domain, with types like UndergraduateStudent, Publication, GraduateCourse, or AssistantProfessor
  - 100 million triples (17 GB) generated using LUBM generator
- SP2Bench Dataset (Benchmark):
  - Types in the dataset include Person, Inproceedings, and Article
  - 10 million triples (1.6 GB) generated using SP2 Bench generator

Real and Benchmark Comparison- Setup



Disk Space

Real and Benchmark Comparison- Setup



No. of Triples

Real and Benchmark Comparison- Setup

1. For some of the datasets (e.g., LUBM) that dataset triples were distributed over a (large) number of files, they assembled all the triples into a single file called SDF.rdf (Single Dataset File)
2. They performed data cleansing. They fixed the data if the fix is obvious (e.g., missing quote or angle bracket symbols) or droped the triple from consideration, when the information in the triple is incomplete.

Real and Benchmark Comparison- Setup

3. They generated three new files, SDF_subj.nt, SDF_prop.nt, and SDF_obj.nt, by sorting SDF.nt along the subjects, properties and objects
   - Each sorting took more than two days in a dual processor server with 24GB of memory and 6TB of disk space
   - The corresponding metrics can be collected by making a single pass
   - Duplicate triples are eliminated during the sorting process

Real and Benchmark Comparison- Setup

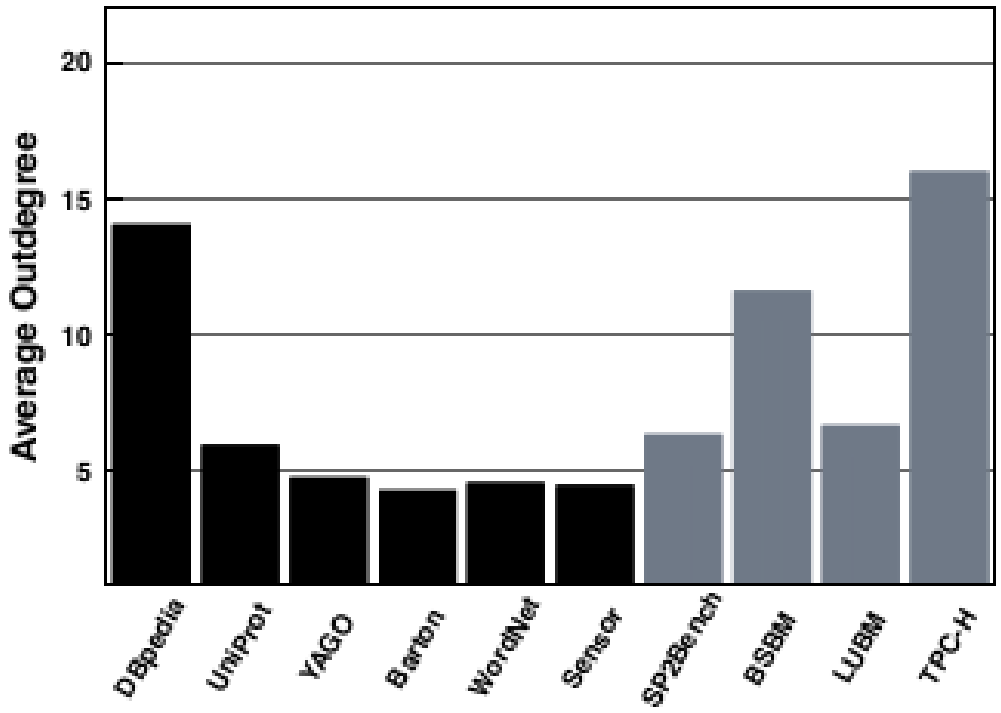
4. SDF_subj.nt is used to extract the type system. The reason for extracting the type system will become clear in the next section after the introduction of the structuredness metrics.

Real and Benchmark Comparison- Setup

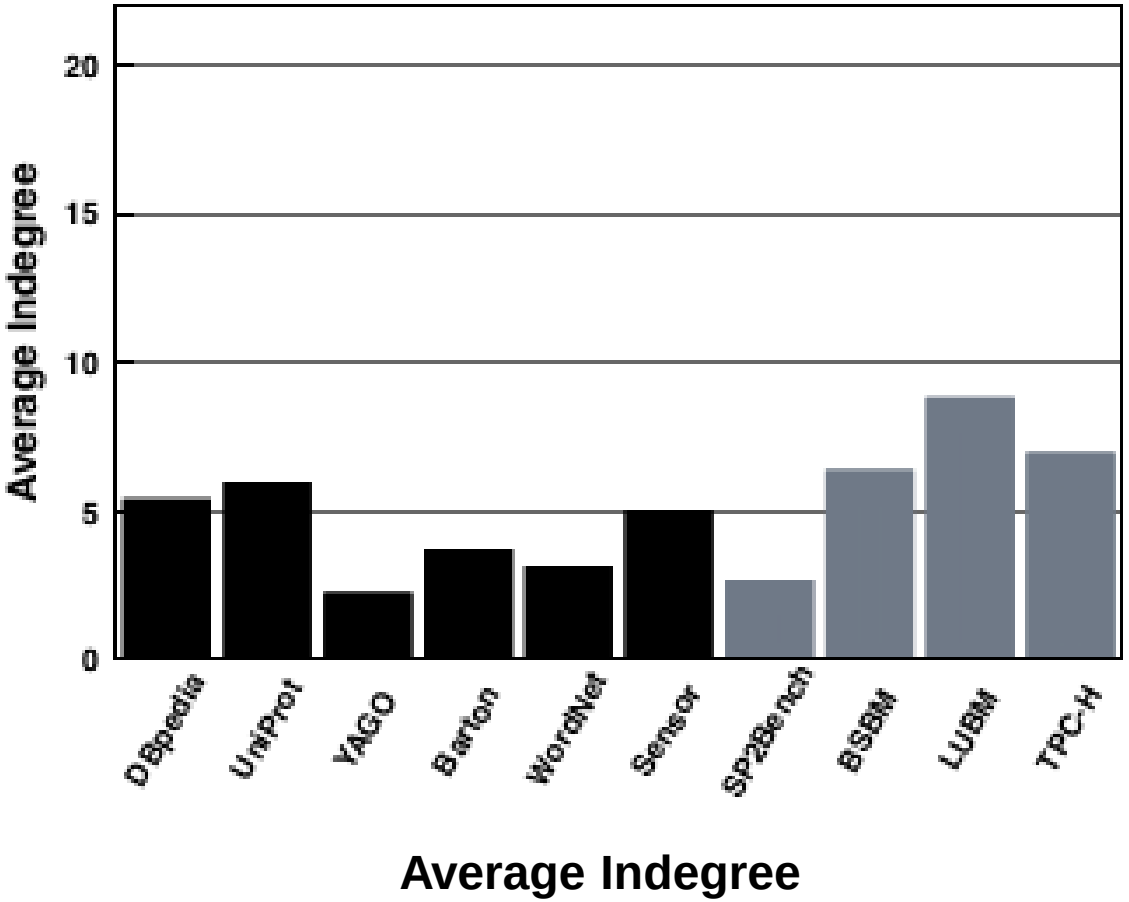
5. They collected matrices from SDF_subj.nt, SDF_prop.nt, and SDF_obj.nt.
   - number of *subjects* and *triples* along with *outdegree (*the number of properties associated with the subject) from SDF_subj.nt
   - *Number* and *occurrence* of properties from SDF_prop.nt
   - *Number* and *indegree* (the number of properties associated with the object) of objects from SDF_obj.nt

Real and Benchmark – Basic Matrices

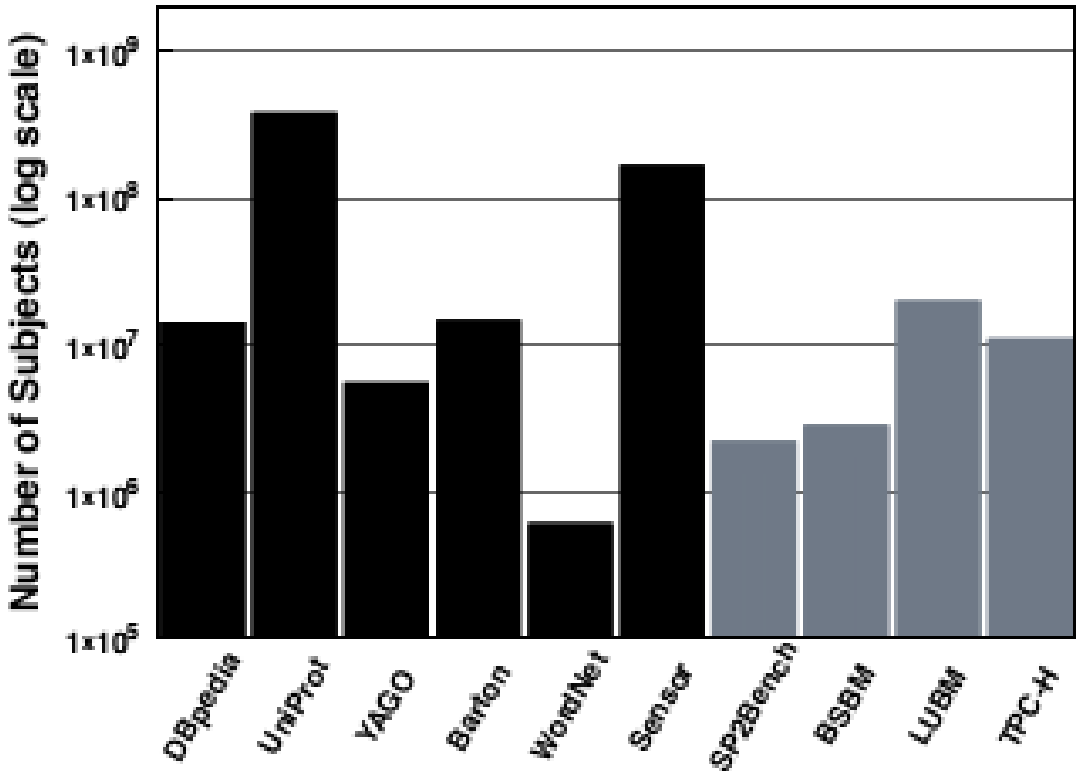
- Outdegree, in most datasets, is relatively small (3-6)
- The exceptions are DBpedia, Sensor, and Barton datasets



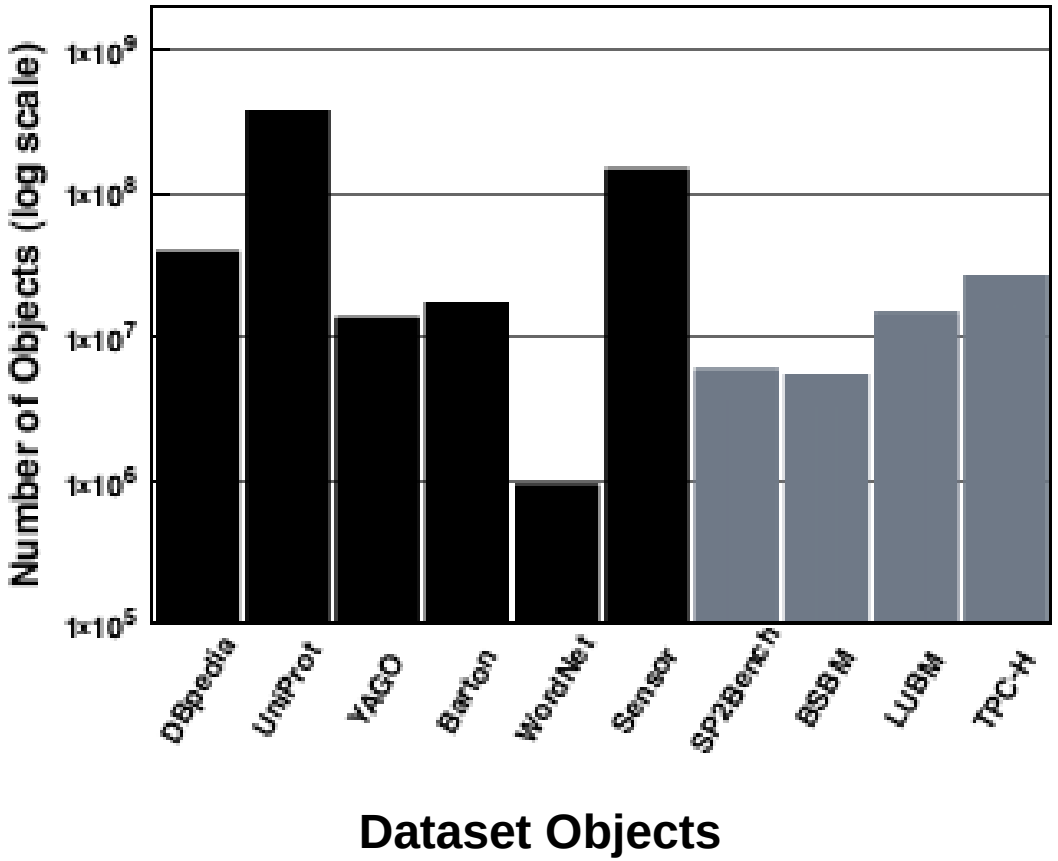
Real and Benchmark – Basic Matrices



Real and Benchmark – Basic Matrices

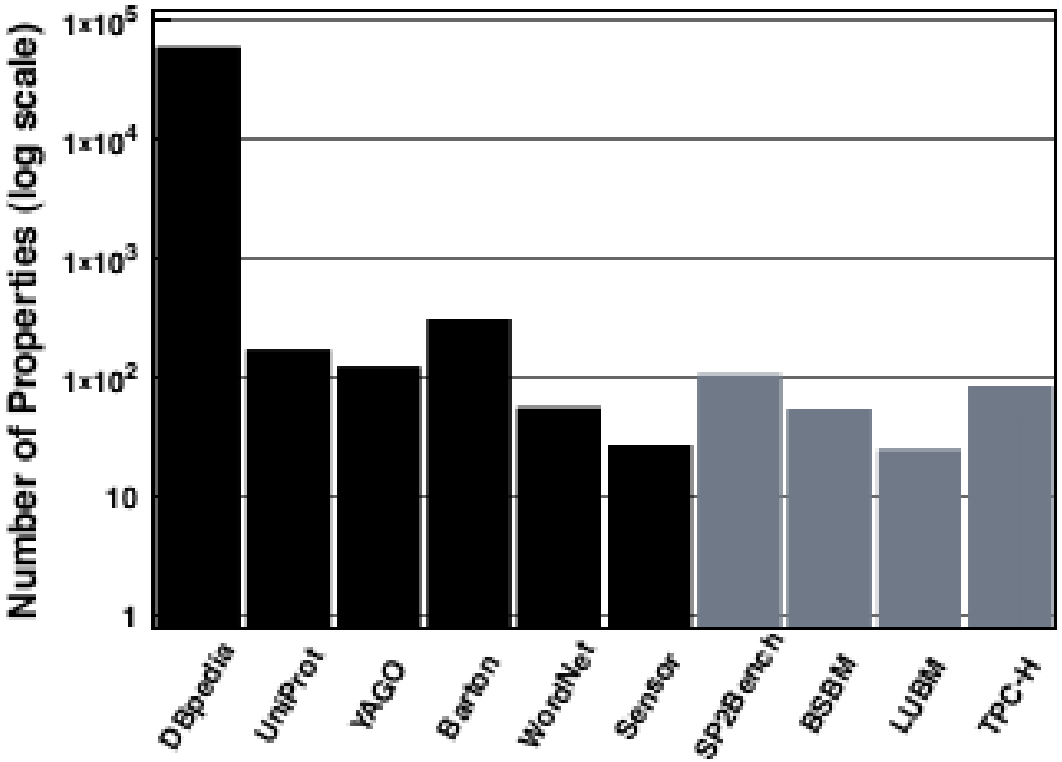


Real and Benchmark – Basic Matrices



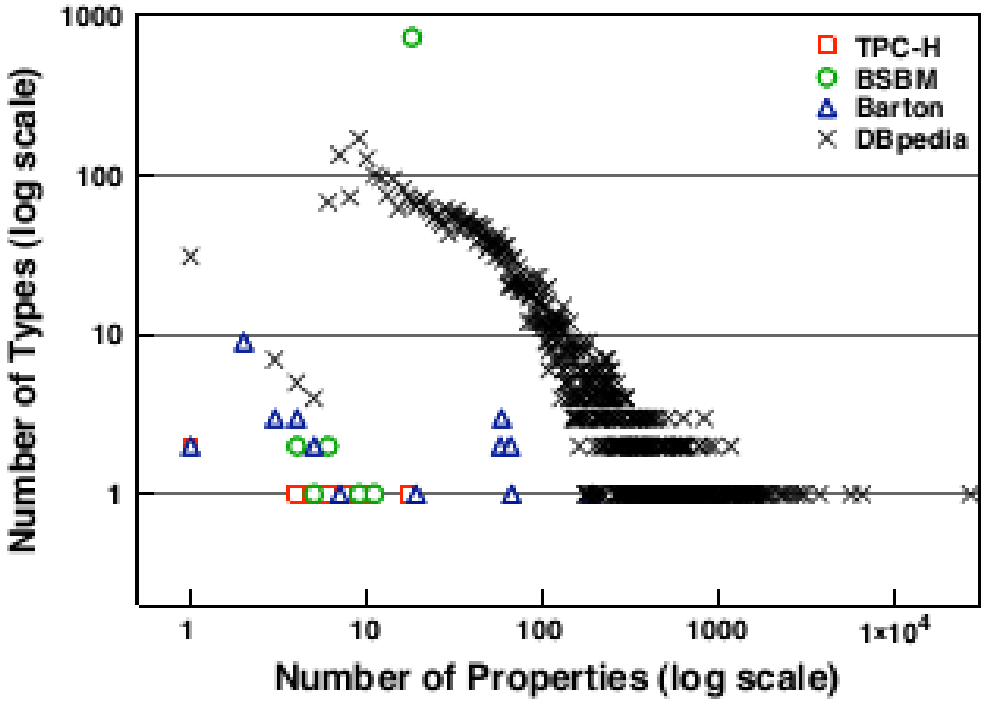
Real and Benchmark – Basic Matrices

- Most datasets have around 100 distinct properties
- DBpedia has almost 3 orders of magnitude more properties.



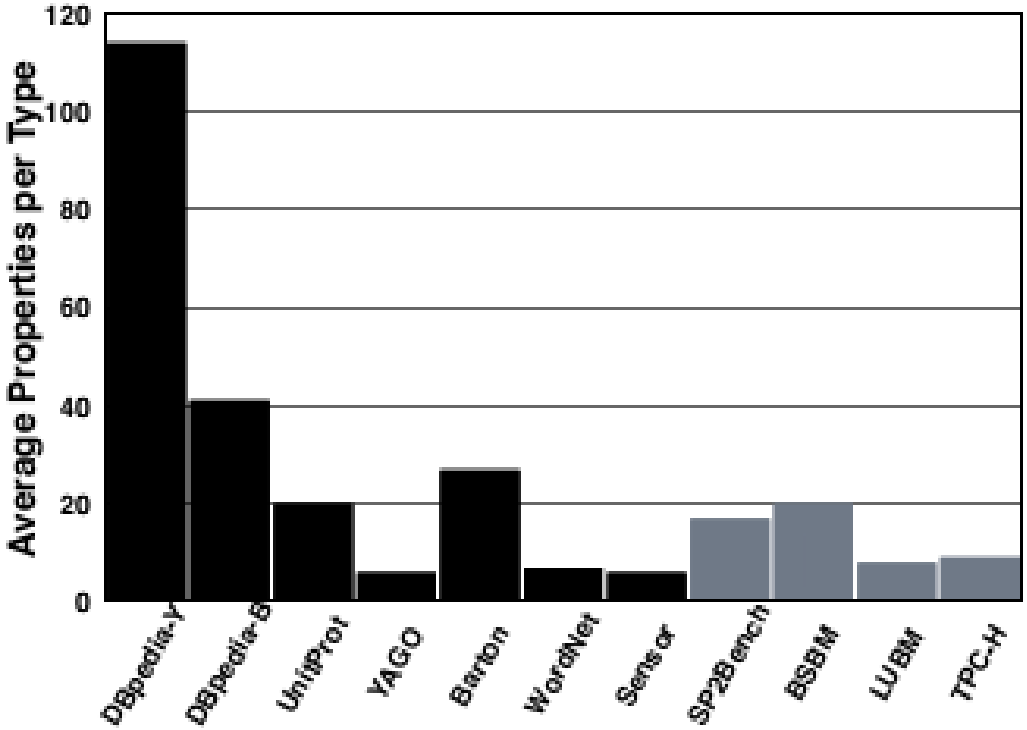
Real and Benchmark – Basic Matrices

- To compute the average number of properties per type, we cannot just divide the distinct number of properties
- We have to find the number of properties for each type then find the means of those numbers



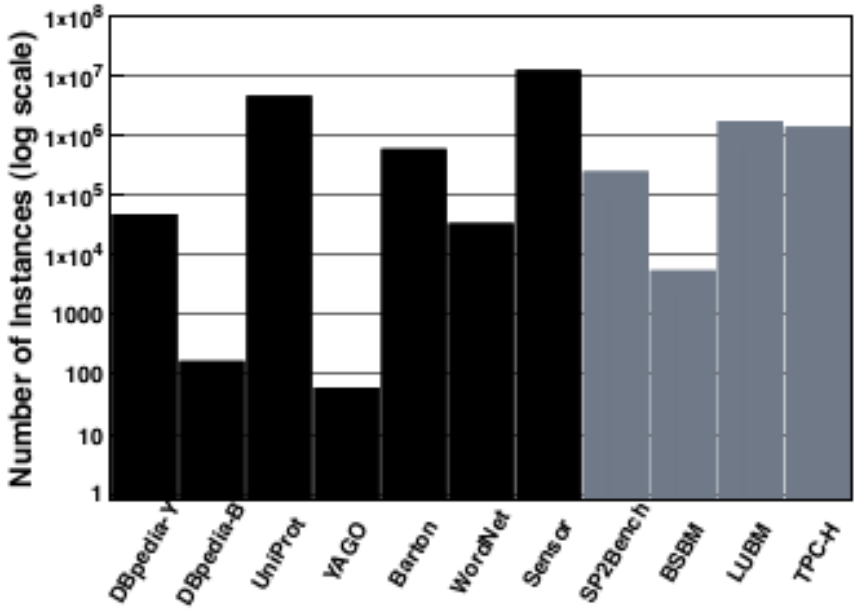
Real and Benchmark – Basic Matrices

- Most datasets have less than 20 properties per type
- DBpedia-Y has approximately 111 properties per type
- DBpedia-B has 38 properties per type



Real and Benchmark – Basic Matrices

- Most datasets a type has approximately 100000 instances
- Since YAGO is essentially an ontology, there are only a few in-stances in the dataset
- Since the type system in DBpedia-B is more fine-grained, there are only a few instances for each type



Real and Benchmark - Structuredness

- These diagrams are just the introduction part of the paper
- We cannot conclude much information about the structure
- The authors introduced a new kind of matric for measuring structuredness
- Intuitively, the level of structuredness of a dataset D with respect to a type T is determined by how well the instance data in D conform to type T

Real and Benchmark - Structuredness

- Example:
  - Type T (Person) has properties name, office and ext
  - Consider for example the dataset D

(person0, name, Eric)
(person0, office, BA7430)
(person0, ext, x4402)
(person1, name, Kenny)
(person1, office, BA7349)
(person1, ext, x5304)
(person2, name, Kyle)
(person2, ext, x6282)

  - If each entity (subject) in T sets values for most (if not all) of the properties of T, then all the entities in D have a fairly similar structure that conforms to T

Real and Benchmark - Structuredness

- Example:
  - Type T' has properties major, GPA, name, office and ext
  - Consider for example the dataset D'

(person0, name, Eric)
(person0, office, BA7430)
(person0, ext, x4402)
(person1, name, Kenny)
(person1, office, BA7349)
(person1, ext, x5304)
(person2, name, Kyle)
(person2, ext, x6282)

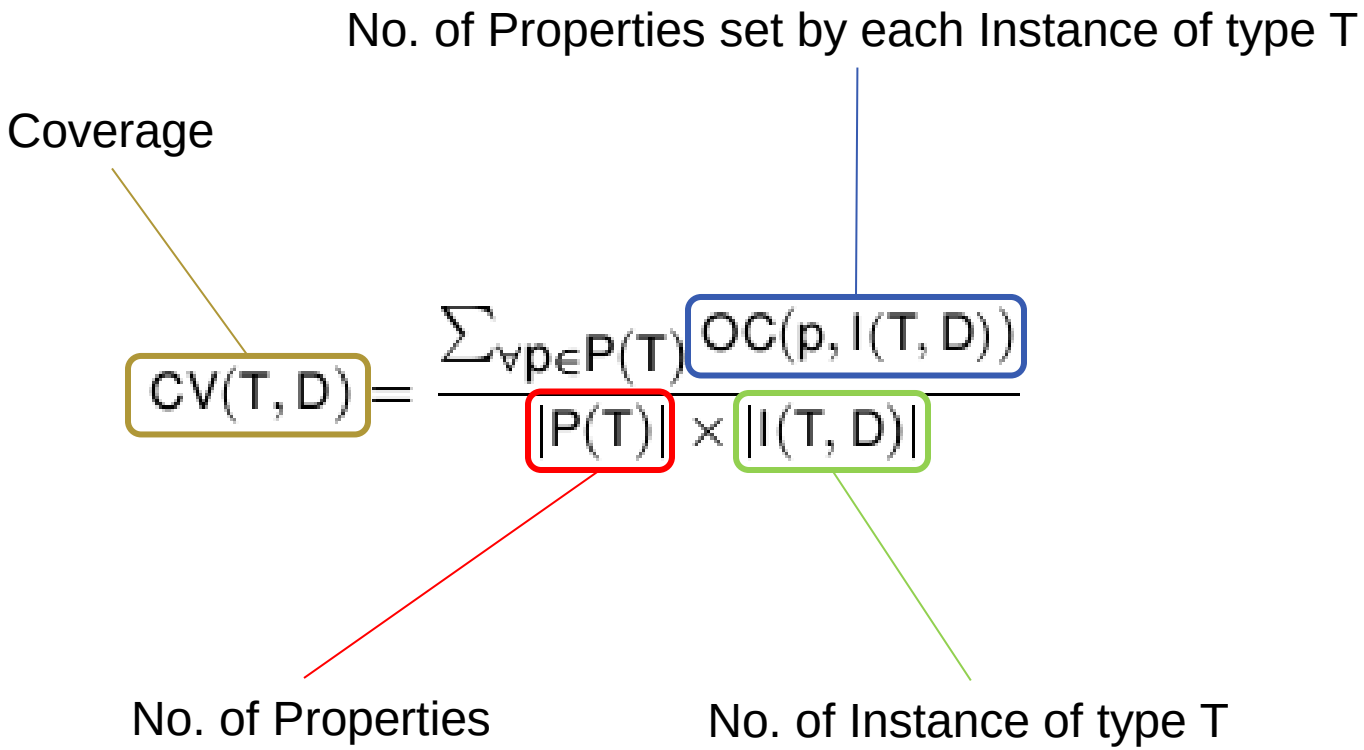
(person3, name, Timmy)
(person3, major, C.S.)
(person3, GPA, 3.4)
(person4, name, Stan)
(person4, GPA, 3.8)
(person5, name, Jimmy)
(person5, GPA, 3.7)

  - D' has low structuredness with respect to T' since the first 3 entities (person0, person1, person2) set values to only name, office, ext but not to GPA and major

Real and Benchmark - Coverage

$$CV(T, D) = \frac{\sum_{\forall p \in P(T)} OC(p, I(T, D))}{|P(T)| \times |I(T, D)|}$$

Real and Benchmark - Coverage



Real and Benchmark - Coverage

- Example:
  - Type P' has properties major, GPA, name, office and ext
  - Dataset D'

(person0, name, Eric)
(person0, office, BA7430)
(person0, ext, x4402)
(person1, name, Kenny)
(person1, office, BA7349)
(person1, ext, x5304)
(person2, name, Kyle)
(person2, ext, x6282)

(person3, name, Timmy)
(person3, major, C.S.)
(person3, GPA, 3.4)
(person4, name, Stan)
(person4, GPA, 3.8)
(person5, name, Jimmy)
(person5, GPA, 3.7)

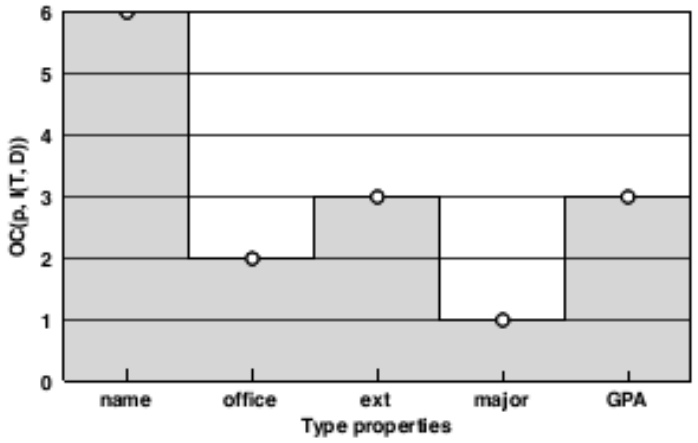
  - $|P(T')| = 5$
  - $|I(T',D')| = 6$
  - $OC(name, I(T',D')) = 6$
  - $OC(major, I(T',D')) = 1$

Real and Benchmark - Coverage

- Example:
  - Property name has been set by 6 out of 6 instances
  - Property major has been set by 1 out of 6 instances

Real and Benchmark - Coverage

- Example:
  - For a dataset with perfect structuredness, every instance of the type in D' sets all its properties which is the whole area of the rectangle below
  - The coverage of the example, according to the formula is: (6+2+3+1+3)/30 = 0.5
  - On average, each instance set half of its properties



Real and Benchmark – Weighed Sum

- Coverage considers only the structuredness of a dataset with respect to a single type
- In practice, a dataset D has entities from multiple types
- D might have a high structuredness for a type T
  - CV(T; D) = 0.8
- D has a low structuredness for another type T'
- CV(T'; D) = 0.15
- What is the structuredness of the whole dataset?

Real and Benchmark – Weighed Sum

Weighted Sum of the Coverage

$$WT(CV(T,D)) = \frac{|P(T)| + |I(T,D)|}{\sum_{\forall T \in \mathcal{T}} (|P(T)| + |I(T,D)|)}$$

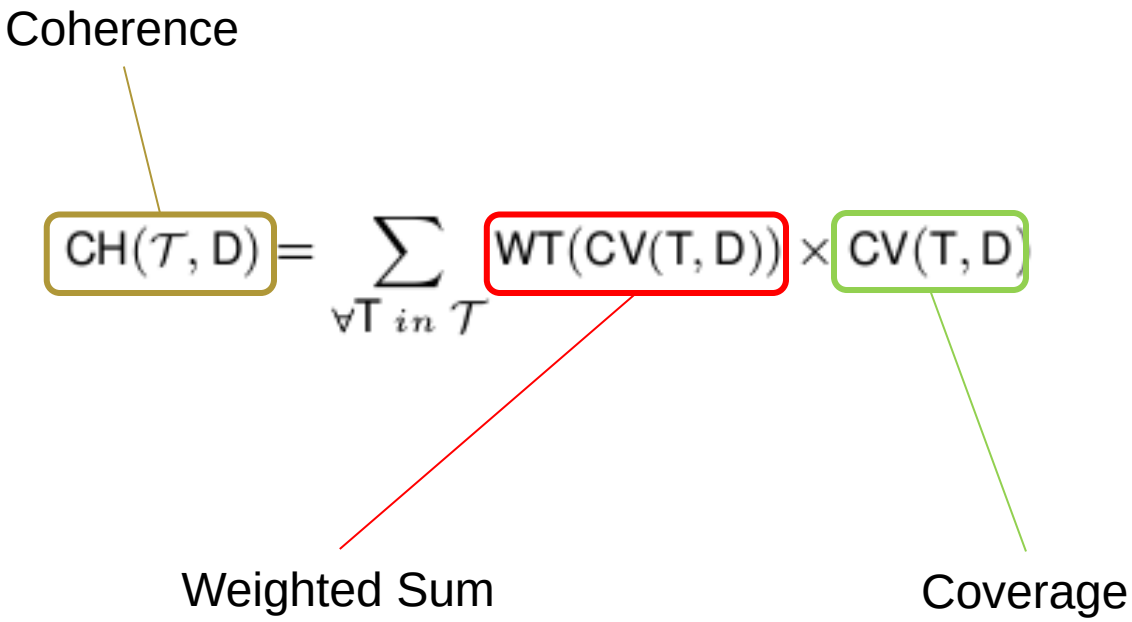
No. of Properties

No. of Instance of type T

Real and Benchmark – Weighed Sum

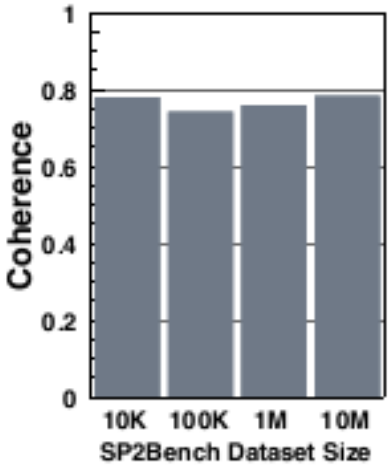
- Calculated the contribution of Coverage of each type to the whole dataset with multiple types
- The formula gives higher weight to types with more instances
  - A single, comparing with 100 instances, has a lower influence in the computation of structuredness of the whole dataset
- The formula gives higher weight to types with a larger number properties
  - A type with hundreds of properties carries more weight than a type with only two properties even if its Coverage is high

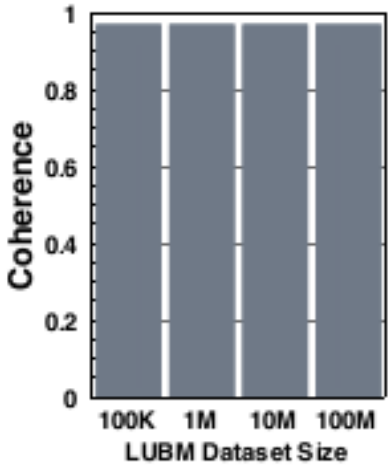
Real and Benchmark – Coherence



Real and Benchmark – Coherence

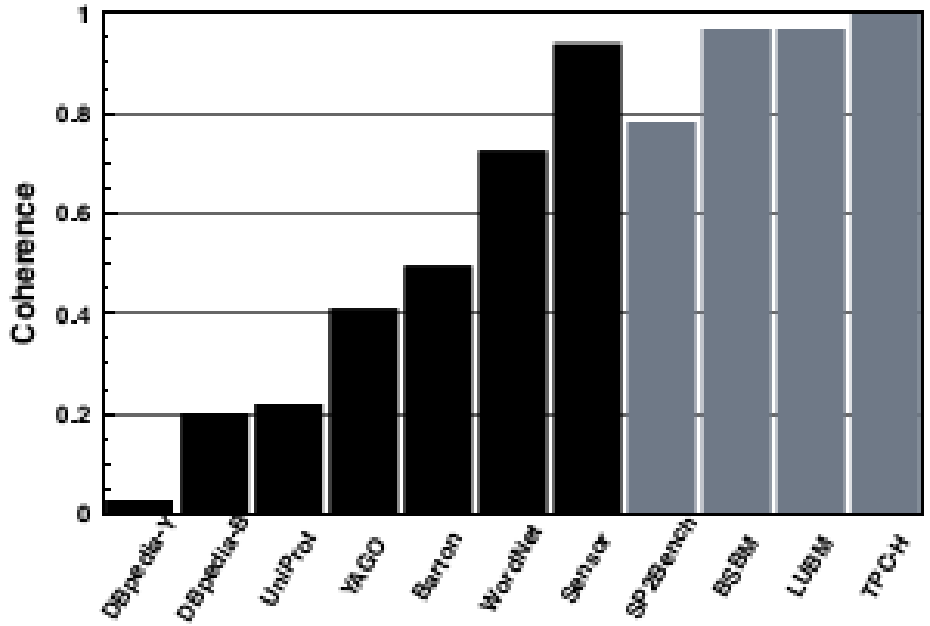
- As the authors suspected, benchmark dataset structuredness is relatively fixed despite the size generate
  - LUBM benchmark coherence is practically constant
  - SP2 Bench datasets barely change when the size increases





Real and Benchmark – Coherence

- Since TPC-H is generated from relational data, it has a coherence of 1.
- Existing benchmark datasets are not representative of all RDF data



Real and Benchmark – Benchmark Gen.

- The authors proposed a new method of generating benchmark datasets which allows the user to control the size and coherence of the dataset
- They decided not to used bottom-up approach implemented by LUBM, SP2 Bench and BSBM

Real and Benchmark – Benchmark Gen.

- Their benchmark generator can take any datasets (Real or Benchmark) as input to generate a datasets top-down with specify size |D'| and Coherence CH under the conditions that:

$$|D'| < |D| \qquad CH(\mathcal{T}, D') < CH(\mathcal{T}, D)$$

- This can be done by removing a set of triples with the same subject and property under the constrains that will be later presented.

Real and Benchmark – Benchmark Gen.

- The impact on the coherence of D of the removal :

$$\mathrm{coin}(\mathcal{T}(s), \mathrm{p}) = \mathrm{CH}(\mathcal{T}, \mathrm{D}) - \mathrm{CH}(\mathcal{T}, \mathrm{D})'$$

- From the previous example; if we remove the triple (person1, ext, x5304) from D' , the new coherence will become:

(6+2+2+1+3)/30 = 0.467

- The impact on the coverage is approximately:

0.5 - 0.467 = 0.033

Real and Benchmark – Benchmark Gen.

- Representation in the Algorithm
  - D -> Original dataset
  - D’ -> Target dataset
  - γ -> Target coverage
  - σ -> Target size
  - *S* -> A set of subjects meant to be removed
  - |coin(*S*, p)| -> the number of subjects that are instances of all the types in *S* and have at least one triple with property p

Real and Benchmark – Benchmark Gen.

- To achieve the desired coherence, they formulated the constraints the integer programming problem
  - (C1) The coherence removed is not more than the original coherence
  - (C2.1) We are not completely removing property p from any of the types
  - (C2.2) We are not completely removing instance s from the dataset

$$CV(T,D) = \frac{\sum_{\forall p \in P(T)} OC(p, I(T,D))}{|P(T)| \times |I(T,D)|}$$

This cannot be 0

Real and Benchmark – Benchmark Gen.

- Constraints (cont.)
  - (C3 and C4) ρ are used for fine-tuning the algorithm to get the desired size

$$(1-\rho) \times (|\mathsf{D}| - \sigma) \leq \sum_{\mathcal{S} \subseteq \mathcal{T}, p} \mathsf{X}(\mathcal{S}, \mathsf{p}) \times \mathsf{ct}(\mathcal{S}, \mathsf{p}) \qquad \text{(C3)}$$

$$\sum_{\mathcal{S} \subseteq \mathcal{T}, p} \mathsf{X}(\mathcal{S}, \mathsf{p}) \times \mathsf{ct}(\mathcal{S}, \mathsf{p}) \leq (1+\rho) \times (|\mathsf{D}| - \sigma) \qquad \text{(C4)}$$

average number of triples per coin

  - (M) Maximizing the coherence while not going below γ

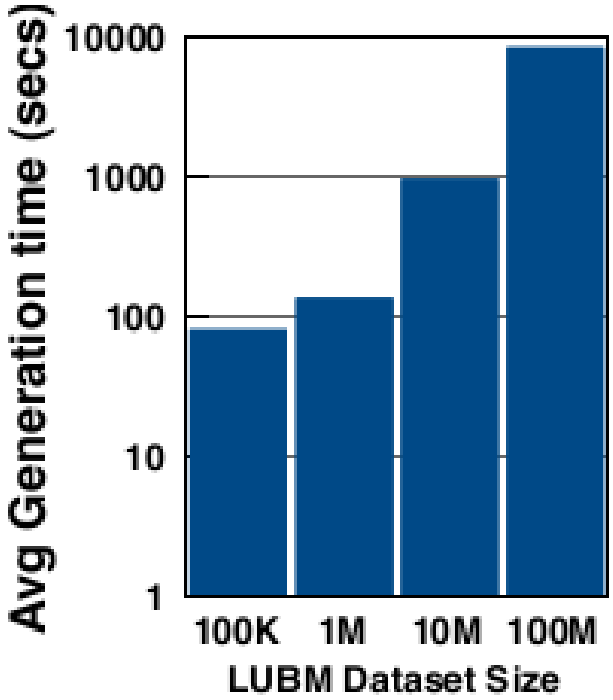
Real and Benchmark – Benchmark Gen.

- Step 1: Compute Coherence, coin(*S*,p) and average triples per coin
- Step 2: Formulate the integer programming problem by C1, C2, C3, C4, and M using lpsolve 5.5
- Step 3: If the problem did not have a solution, try to reduce the dataset by removing a percentage of instances uniformly and continue from Step 1
  - Certain size and coherence doesn't have a solution (coherence is high but size is low)
  - We should reduce the size uniformly to reduce the chance to change the coherence unintentionally

Real and Benchmark – Benchmark Gen.

- Step 4: If the problem had a solution, then for each coin given by S and p, remove triples subjects that are instances of types in S and have property p
- Step 5: If the resulting dataset size is larger than σ, perform post-processing by attempting to remove from triples with the same subject and property

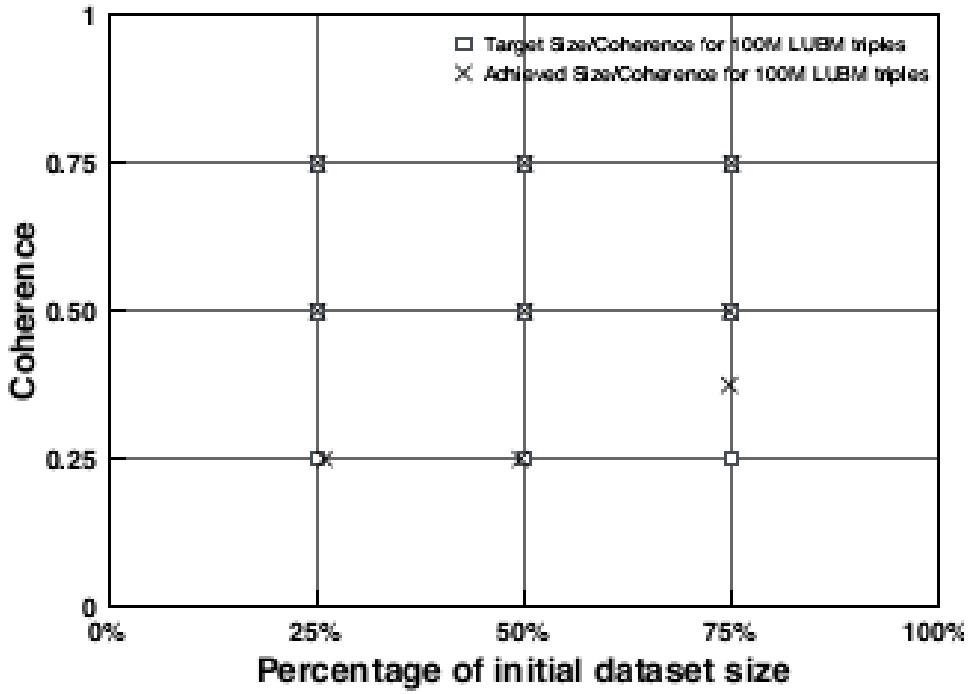
Real and Benchmark – Result



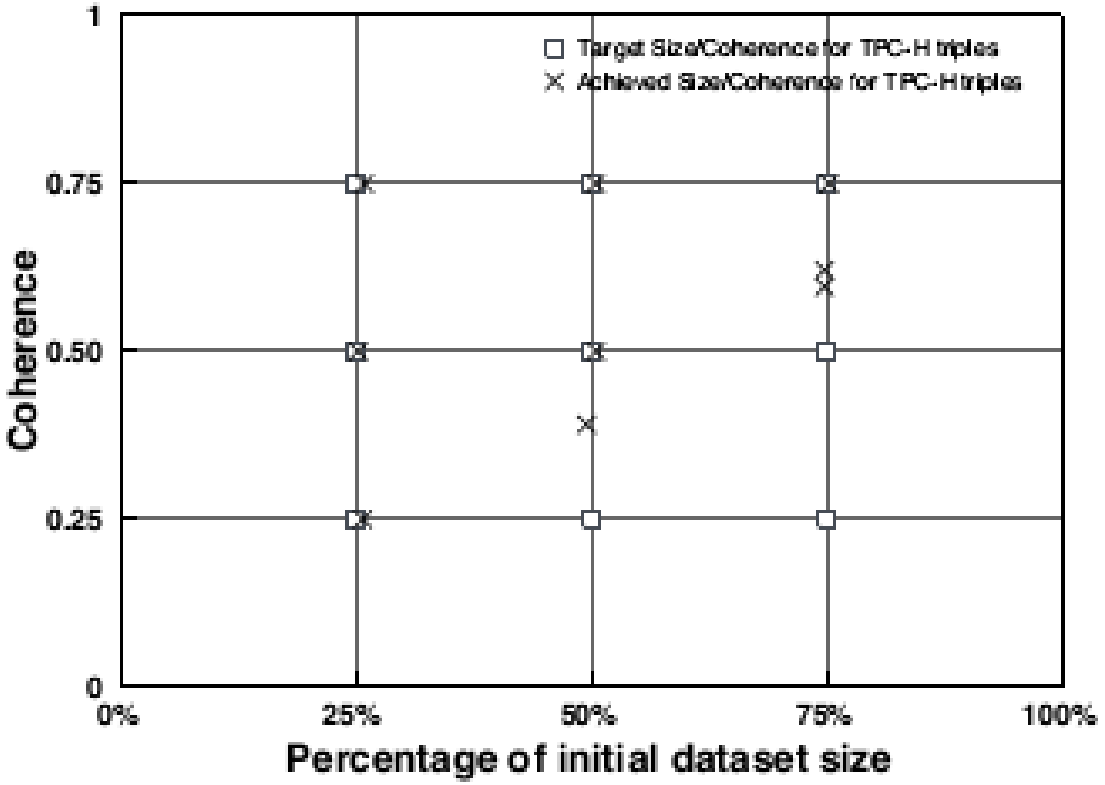
Running Time

Real and Benchmark – Result

- For LUBM, the result missed the target at 75% size 0.25 coherence because they need to lower coherence by a large margin comparing to size



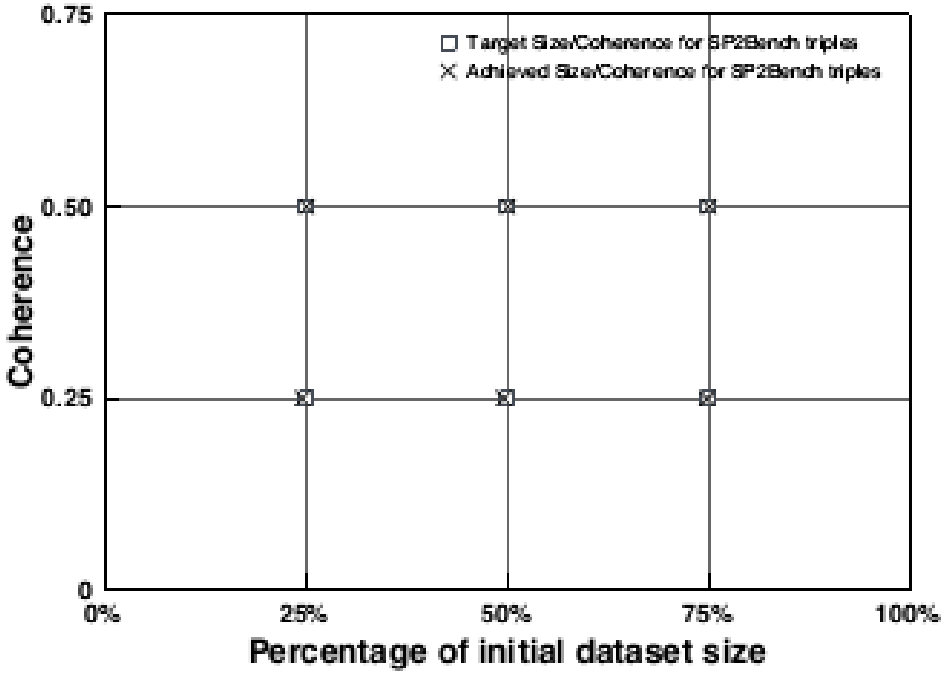
Real and Benchmark – Result



TPC-H

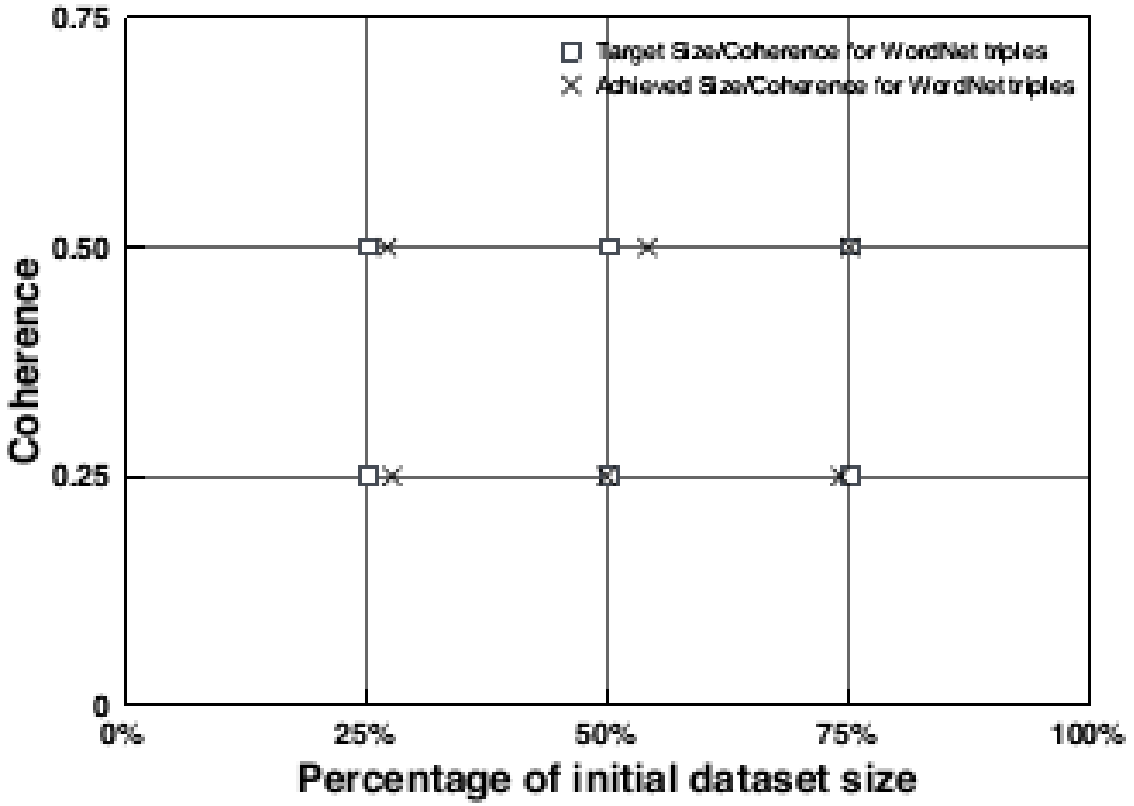
Real and Benchmark – Result

- Wordnet and SP2 Bench, the maximal achievable coherence is 0.5 (the original Coherence of them are between 70-80)



SP2 Bench

Real and Benchmark – Result



Wordnet

Real and Benchmark - Summary

- The research statistically compared various real and benchmark datasets. (They claimed to be the first to do this)
- They proposed matrices for measuring structuredness of datasets called Coverage and Coherence
- They developed a new technique to generate a benchmark dataset top-down using existing dataset (real or benchmark) but the limitation was both the coherence and size have to be lower than the original.