Grammar-based Compression of RDF Graphs

Master's Thesis

Philip Frerk

July 10, 2019

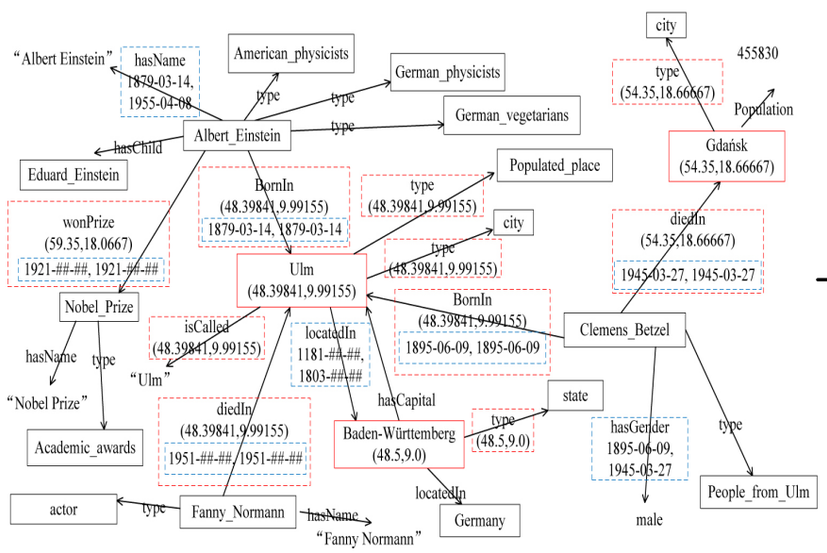
Why compression?

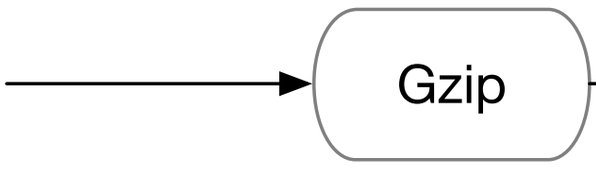
- Knowledge graphs become very big (millions or billions of triples)
  => Problems with **transmission**, **storage** and **consumption**
- Compression can help with respect to all three use cases

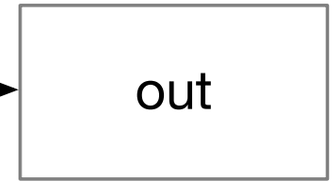
Why compression?

- Knowledge graphs become very big (millions or billions of triples)
  => Problems with **transmission**, **storage** and **consumption**

- Compression can help with respect to all three use cases







Not query-able!

RDF Compressors

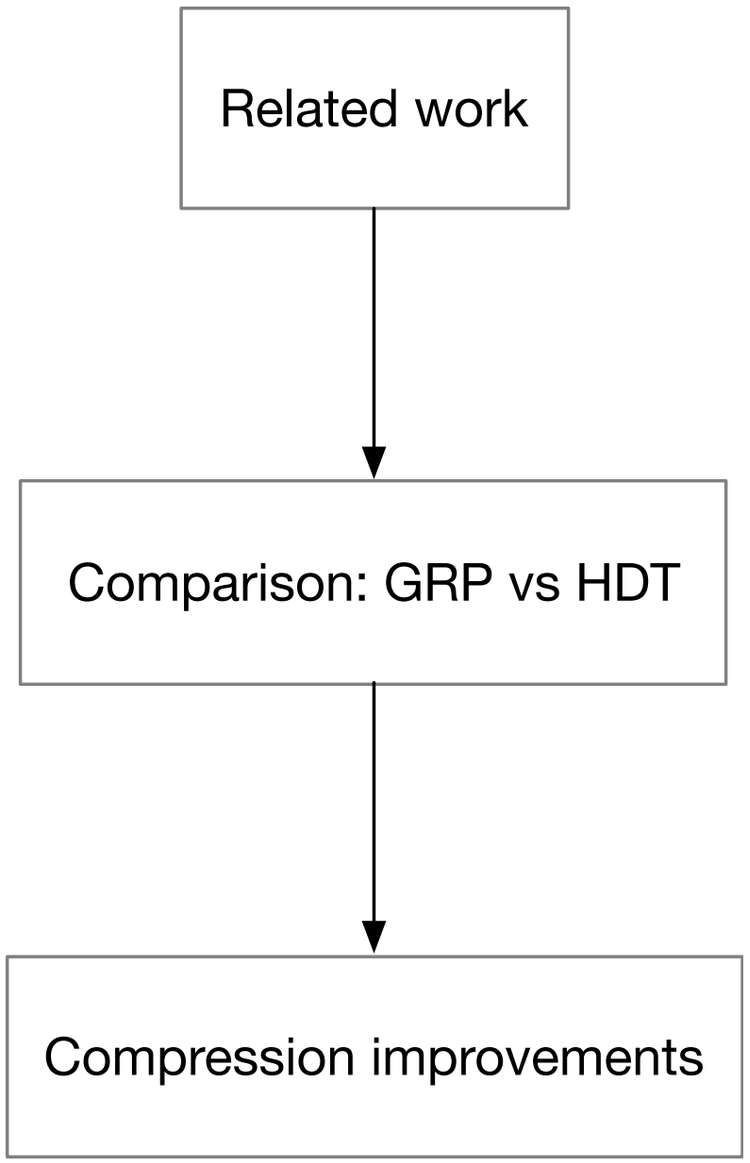
- Compressed data still query-able!
- Can take advantage of RDF features to achieve stronger compression (more domain knowledge)

RDF Compressors

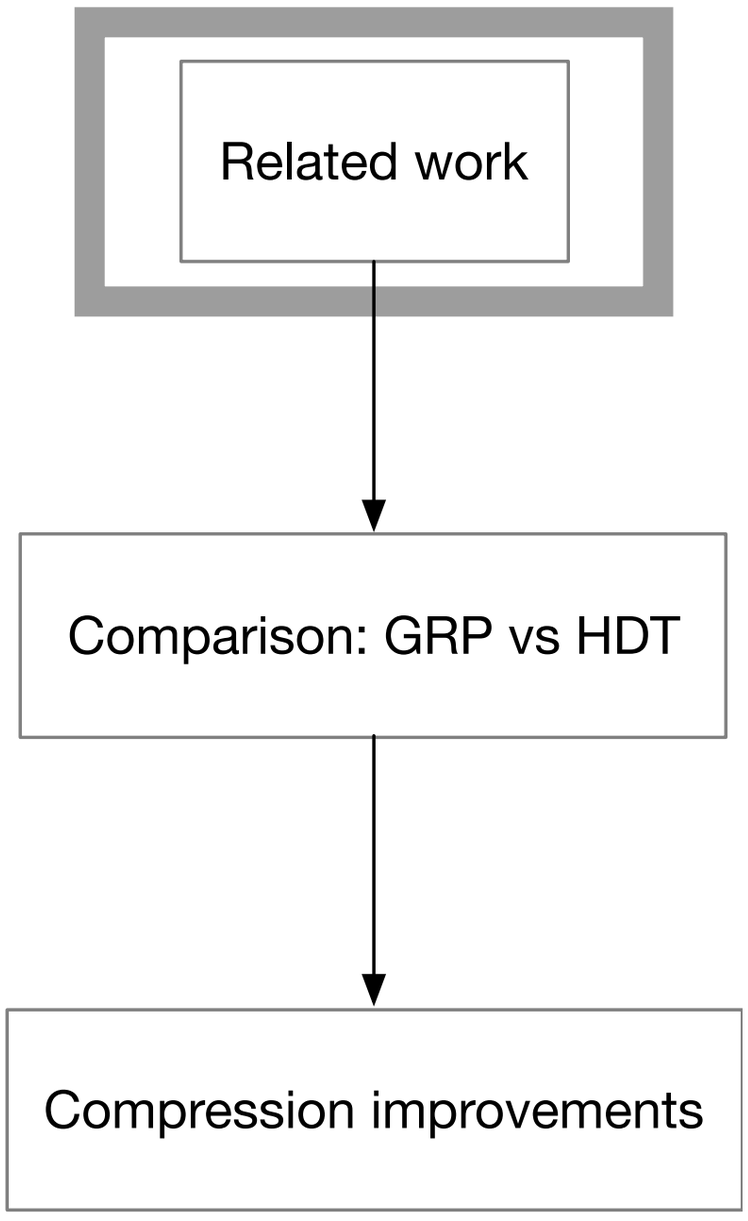
- Compressed data still query-able!
- Can take advantage of RDF features to achieve stronger compression (more domain knowledge)

1. Header Dictionary Triples (HDT)
2. **GraphRePair (GRP) (Grammar-based graph compression)**

Outline


Related work
Comparison: GRP vs HDT
Compression improvements

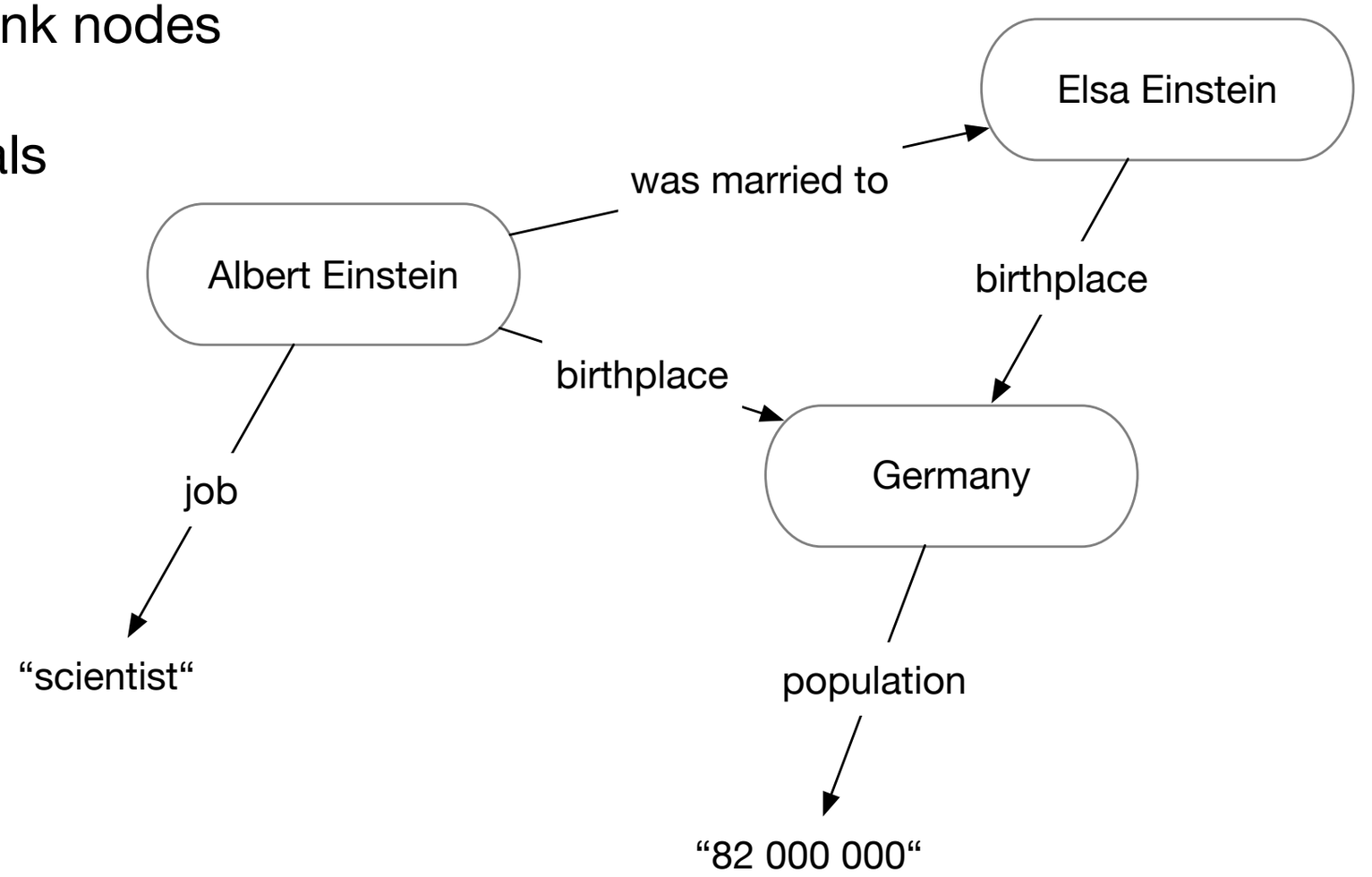
First: Related work



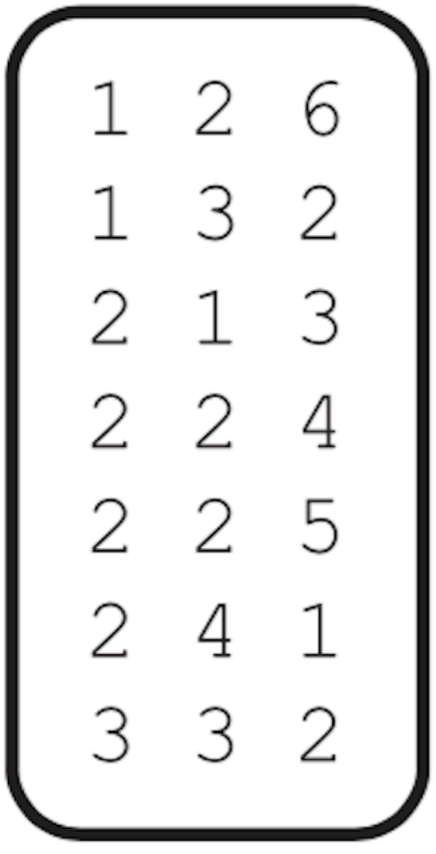
1. RDF
2. HDT
3. GRP

RDF

- subjects: entities (URIs) or blank nodes
- predicates/properties: URIs
- objects: entities (URIs) or literals

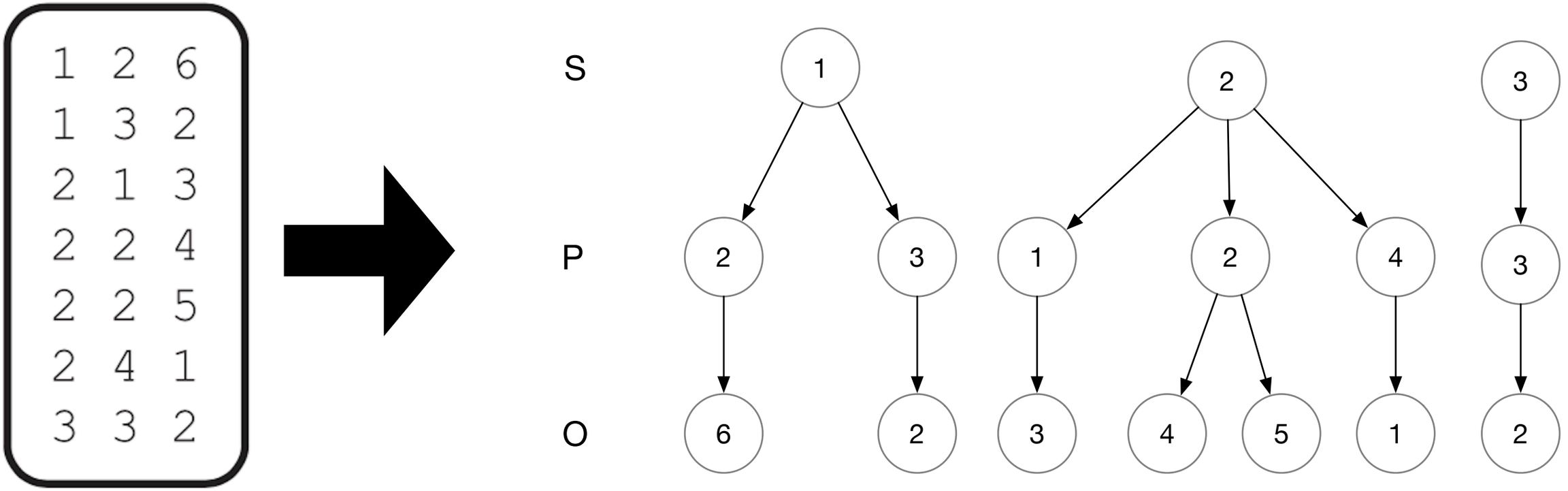


HDT - Dictionary



- Assign an ID to node label (URI, blank node or literal)
- Store that mapping (**dictionary**)
- Leads already to size reduction
- URIs have long common prefixes => use prefix-based text compression

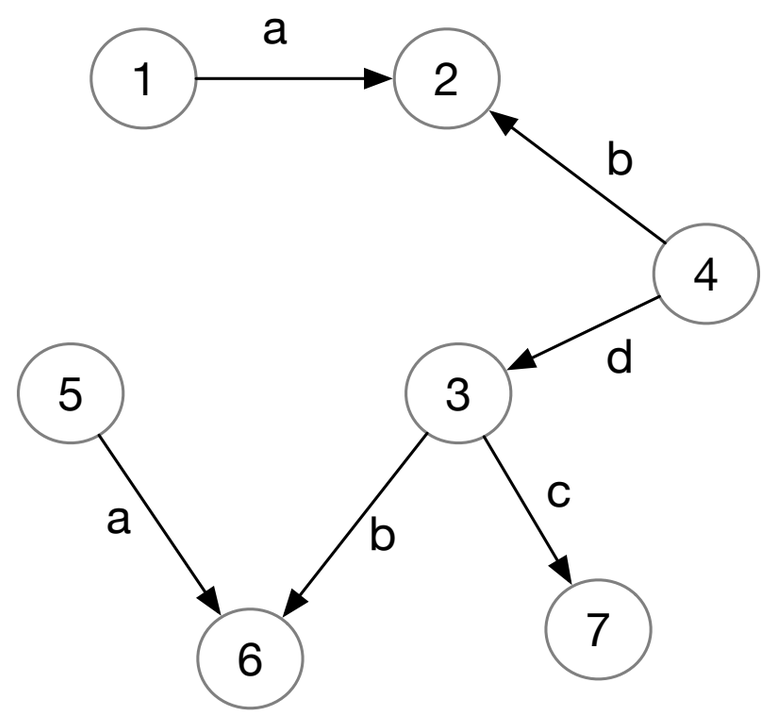
HDT - Compact Triples



Further size reduction!

GRP

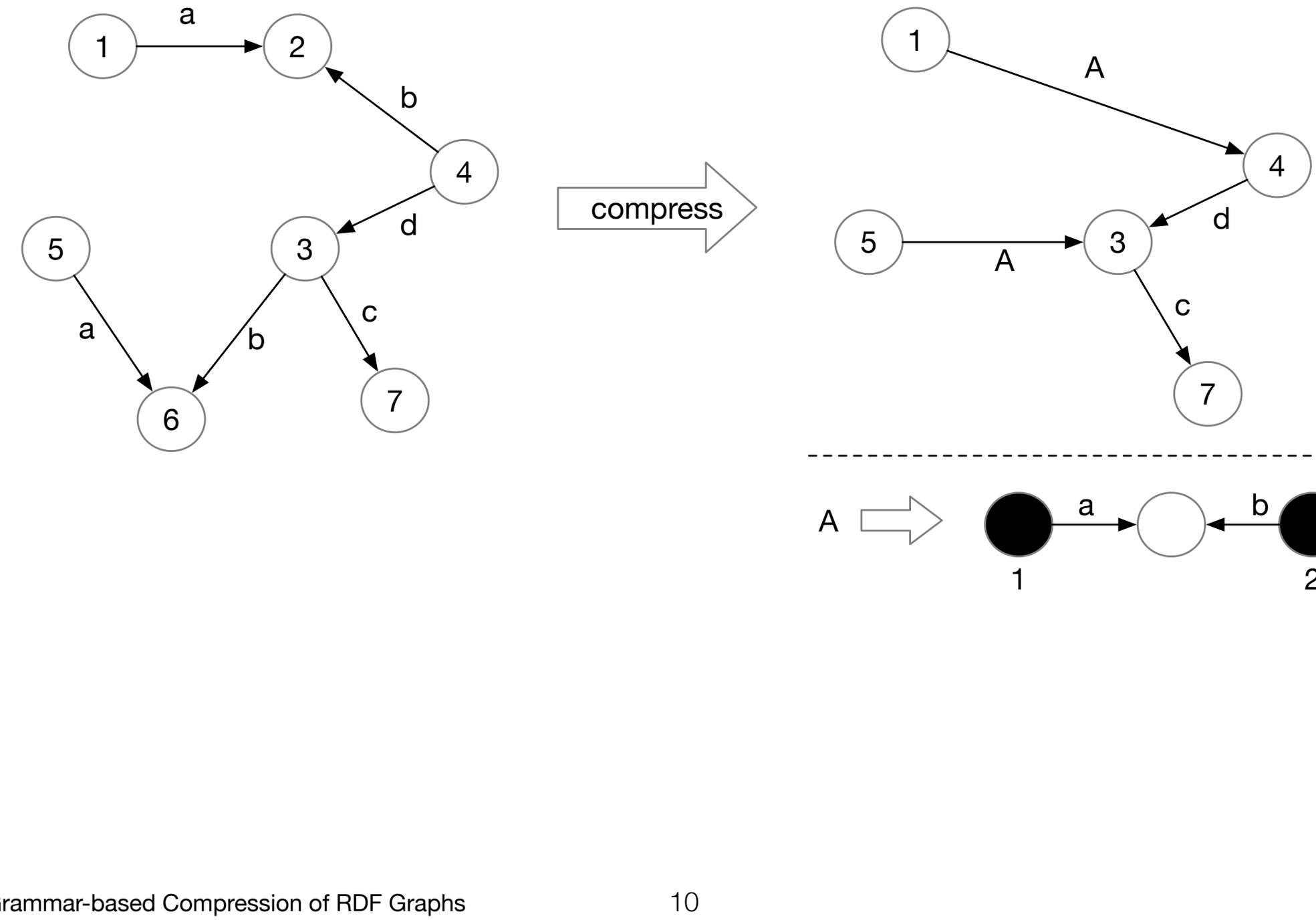
- Also mapping of node labels (URIs, blank nodes and literals) to IDs (**Dictionary**) (mapping is not stored)



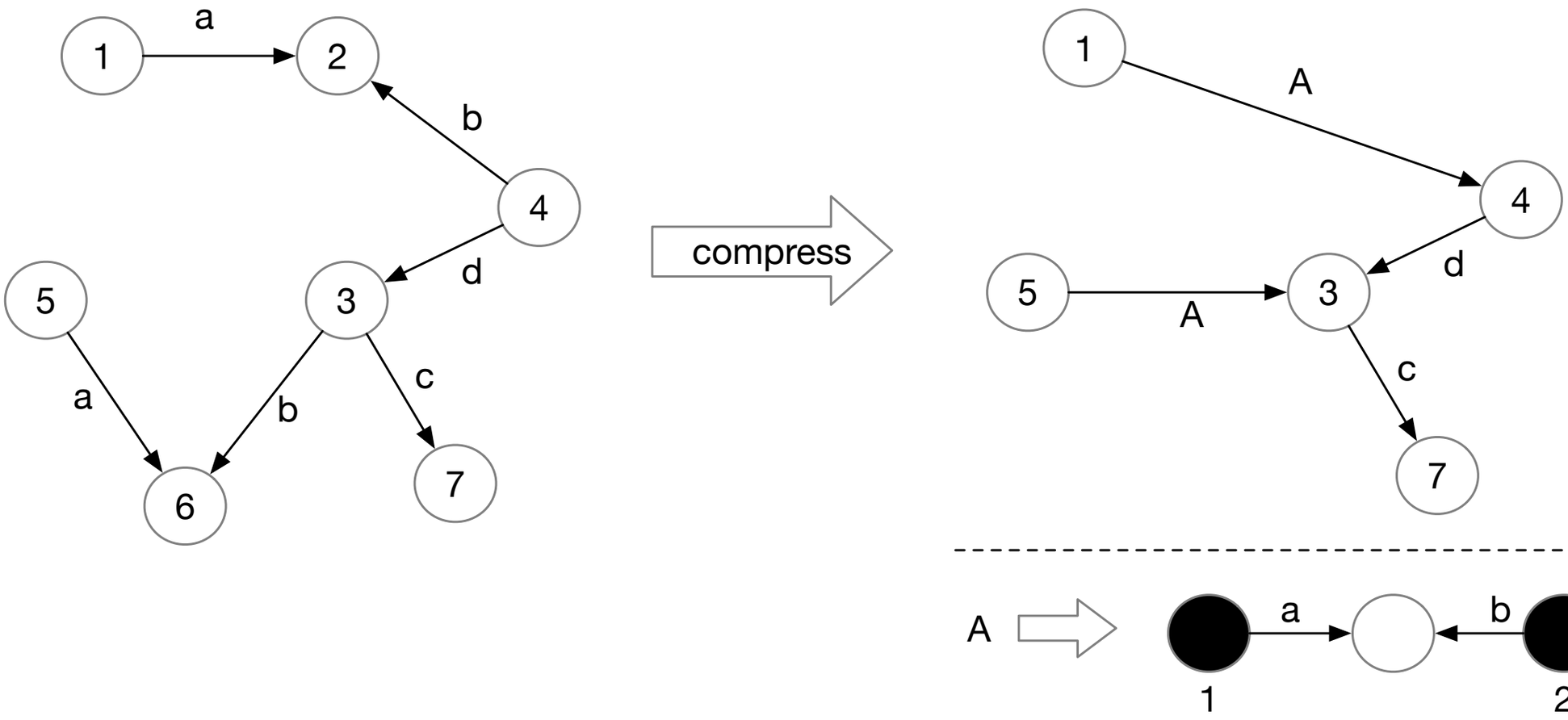
- Views RDF file as graph, not as list of triples

- Searches for repeating patterns (sub structures)

GRP - Example



GRP - Example



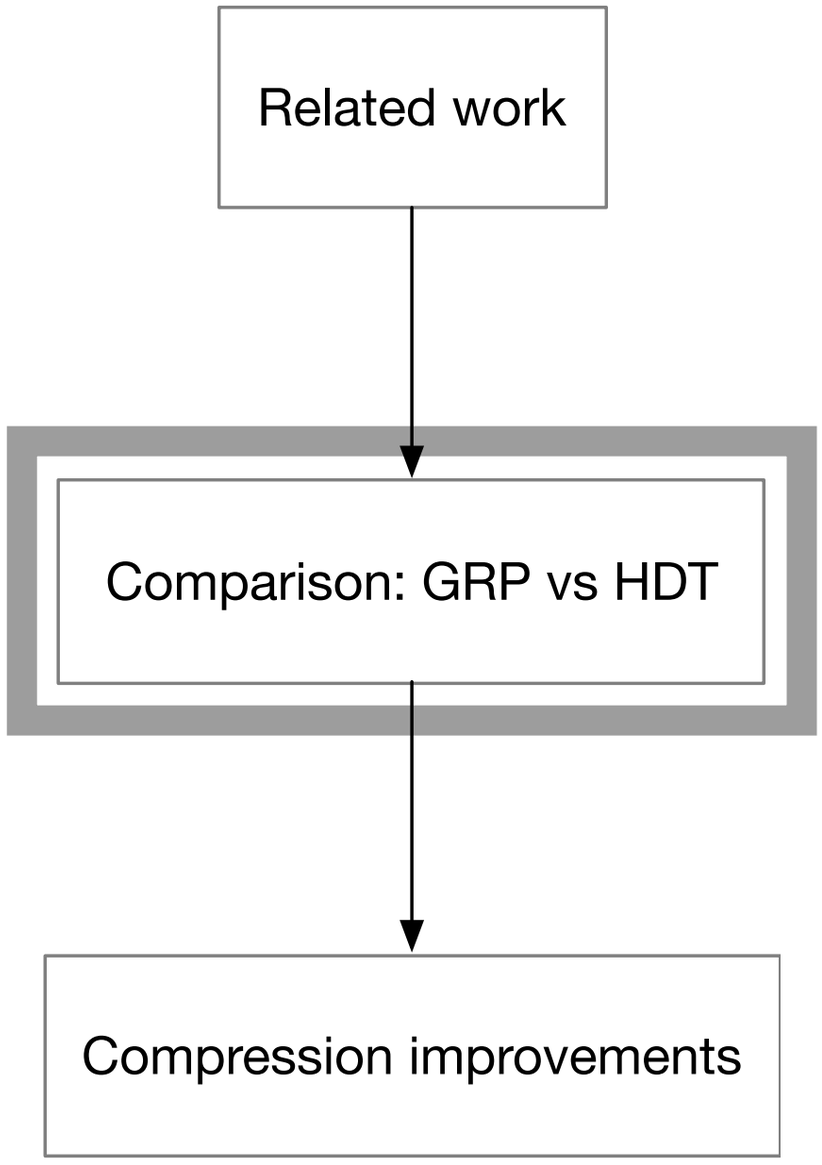
There are many other digram types!

GRP - Grammar Encoding

- Start rule: k^2-trees (way of compressing adjacency matrix)
    - Problems:
        1. Stronger compr. => sometimes more space (Rule contains edge labels which produces overhead)

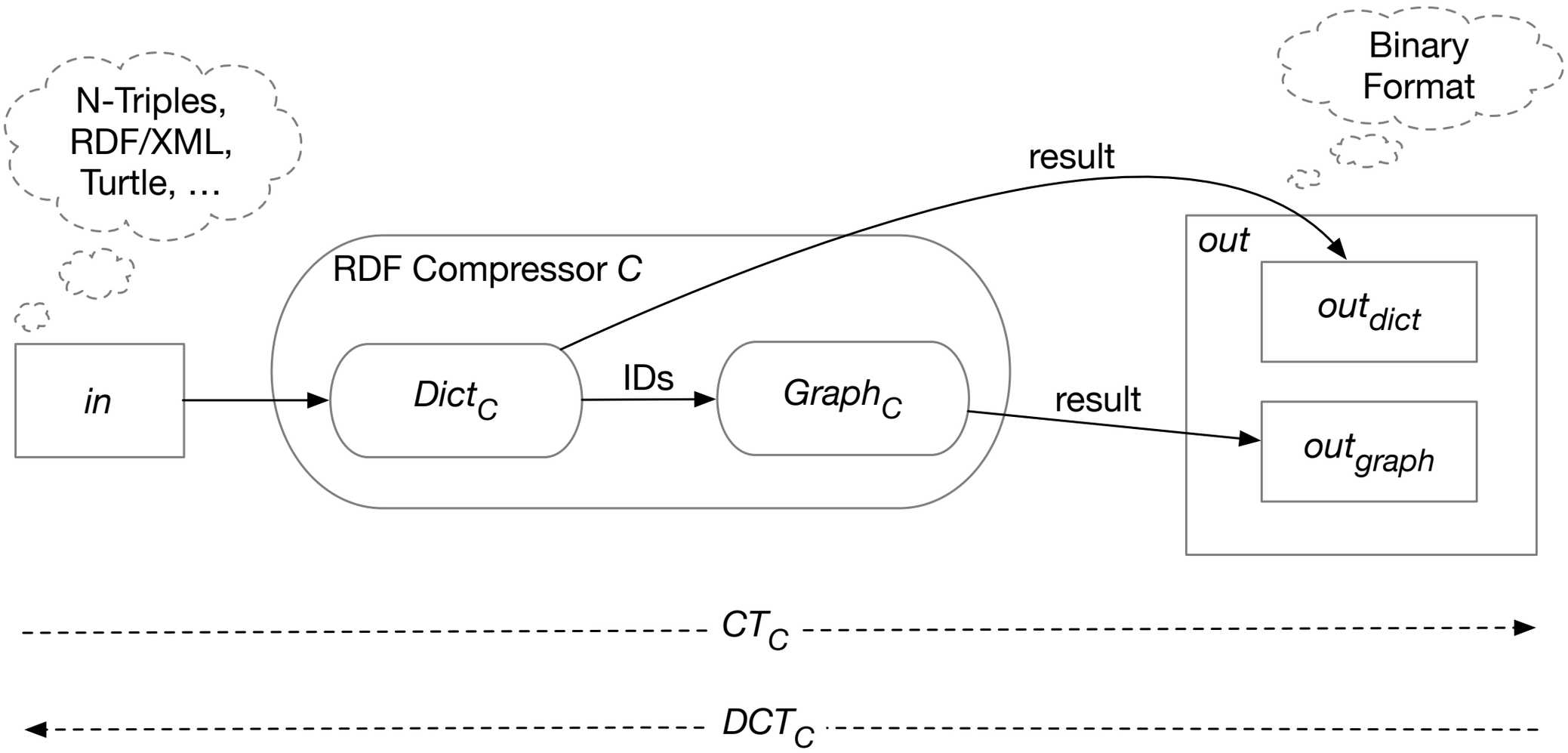
- Remaining rules: variable length delta codes method to express objects as bit-sequences

Next: Comparison

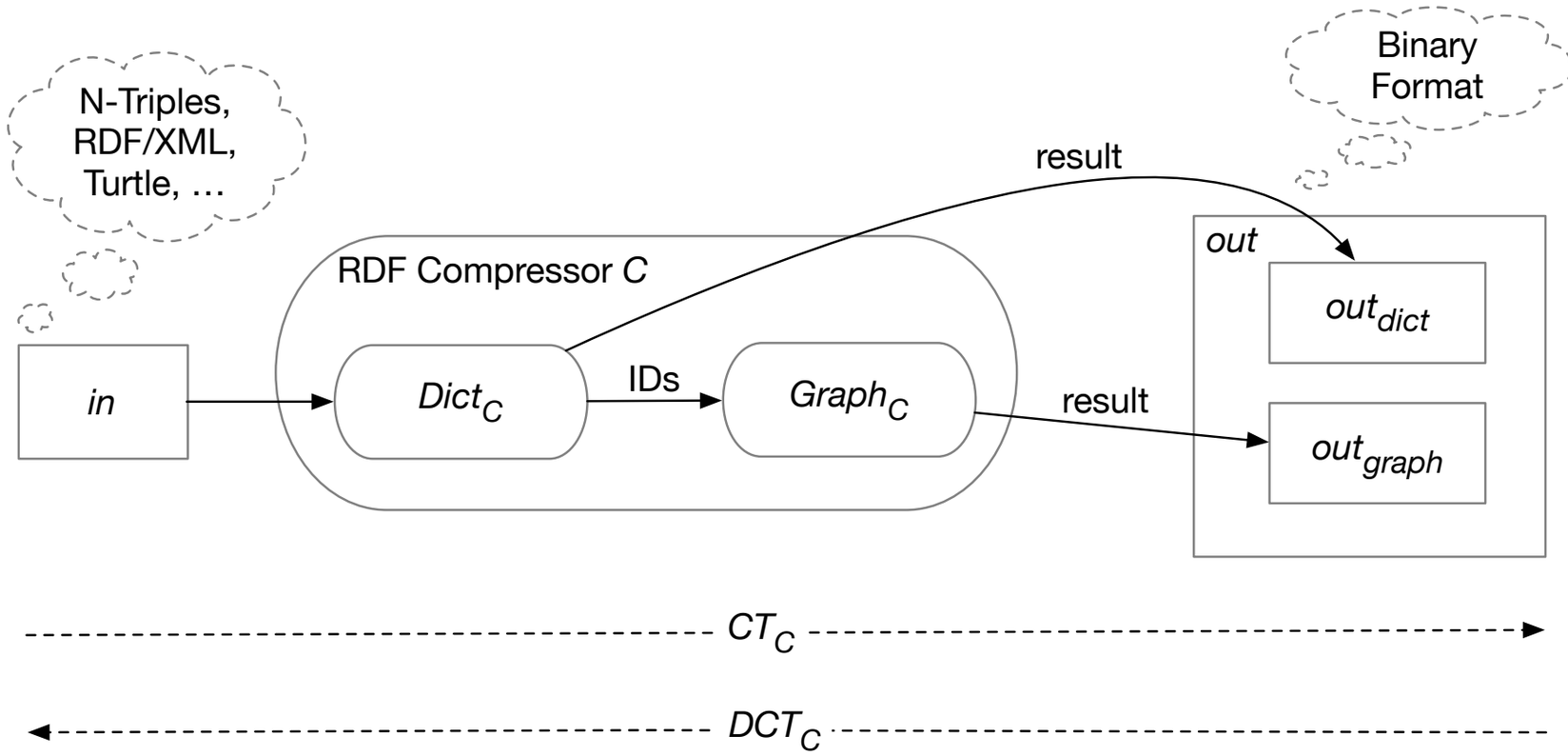


1. Formal compressor model
2. Input analysis of HDT and GRP
3. Evaluation results

RDF compressor model



Compression Ratio (CR)

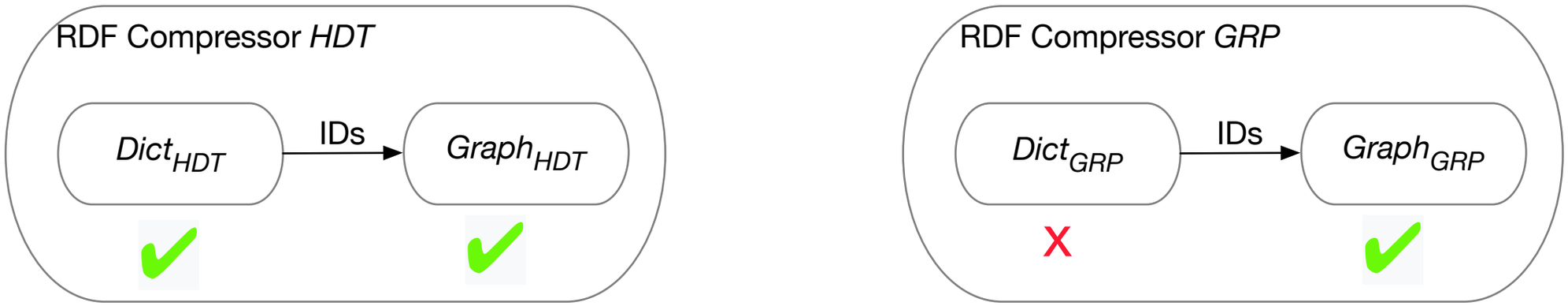


$$CR_C = \frac{|out|}{|in|} \qquad \text{(complete)}$$

$$CR_{Dict_C} = \frac{|out_{dict}|}{|in|} \qquad \text{(only dictionary)}$$

$$CR_{Graph_C} = \frac{|out_{graph}|}{|in|} \qquad \text{(only graph)}$$

Comparison: HDT and GRP



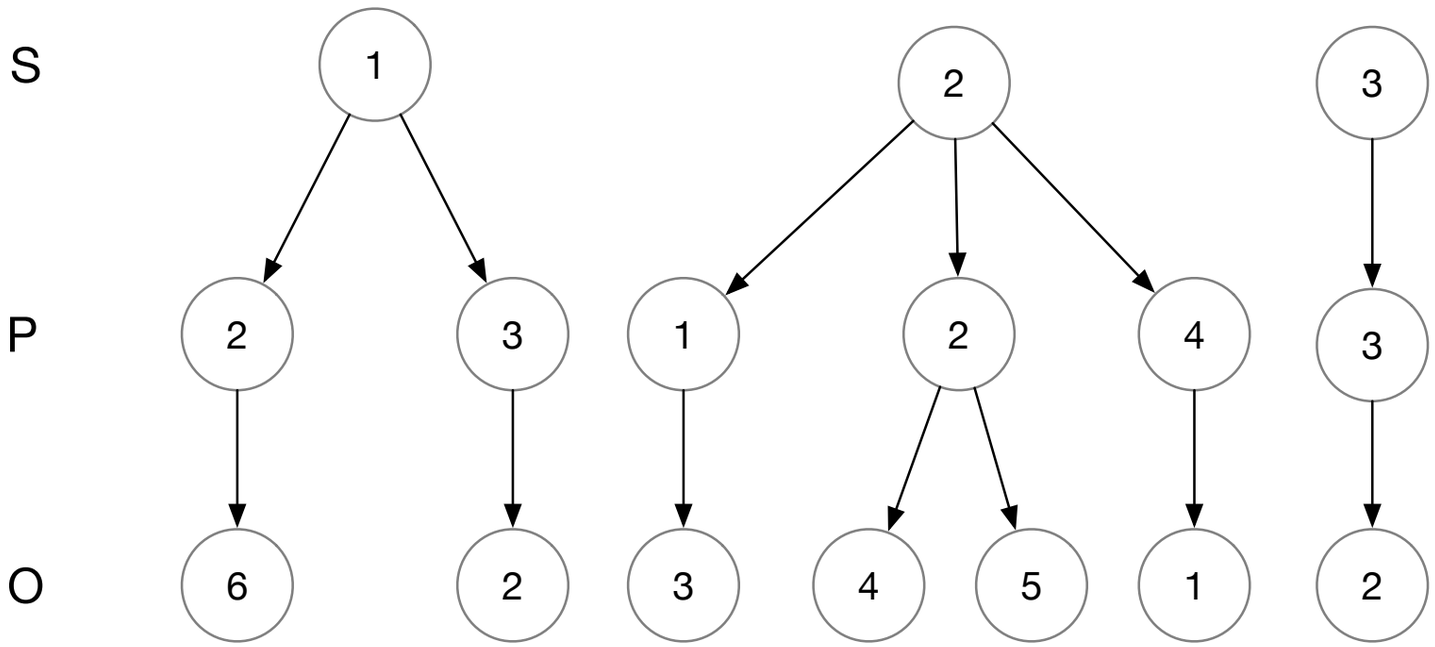
=> Replace $Dict_{GRP}$ with $Dict_{HDT}$ ($Dict_{GRP}$ <— $Dict_{HDT}$)

Comparison: HDT and GRP

$$CR_{Graph_C} = \frac{|out_{graph}|}{|in|}$$

=> Replace $Dict_{GRP}$ with $Dict_{HDT}$ ($Dict_{GRP}$ <— $Dict_{HDT}$)

Input analysis: HDT



- Lower number of subjects leads to lower compr. ratio (**hub pattern**)
- High number of subjects (few objects) leads to higher compr. ratio (**authority pattern**)

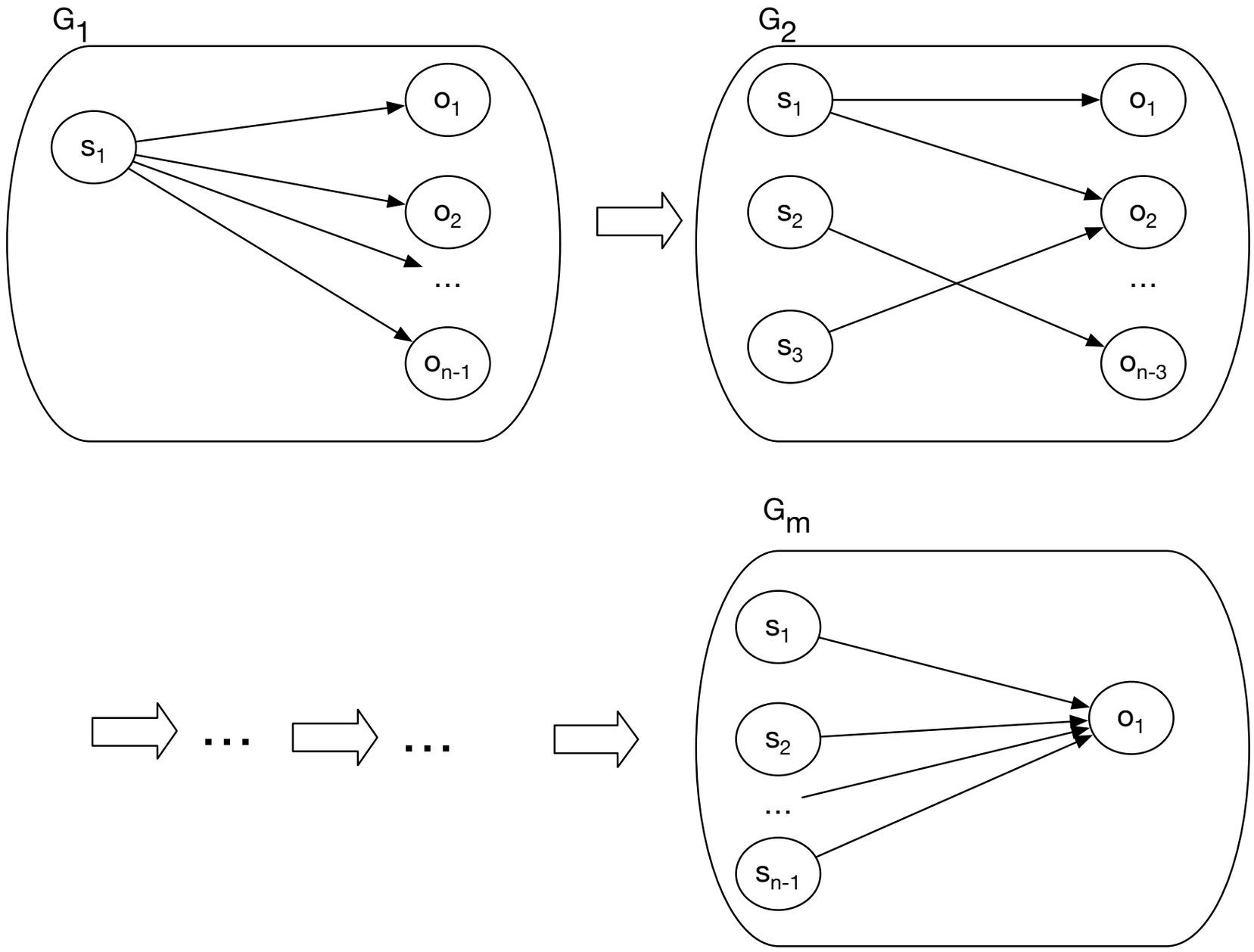
Input analysis: GRP

- More complex: GRP builds sub structures of many different kinds
  => highly depends on the graph structure

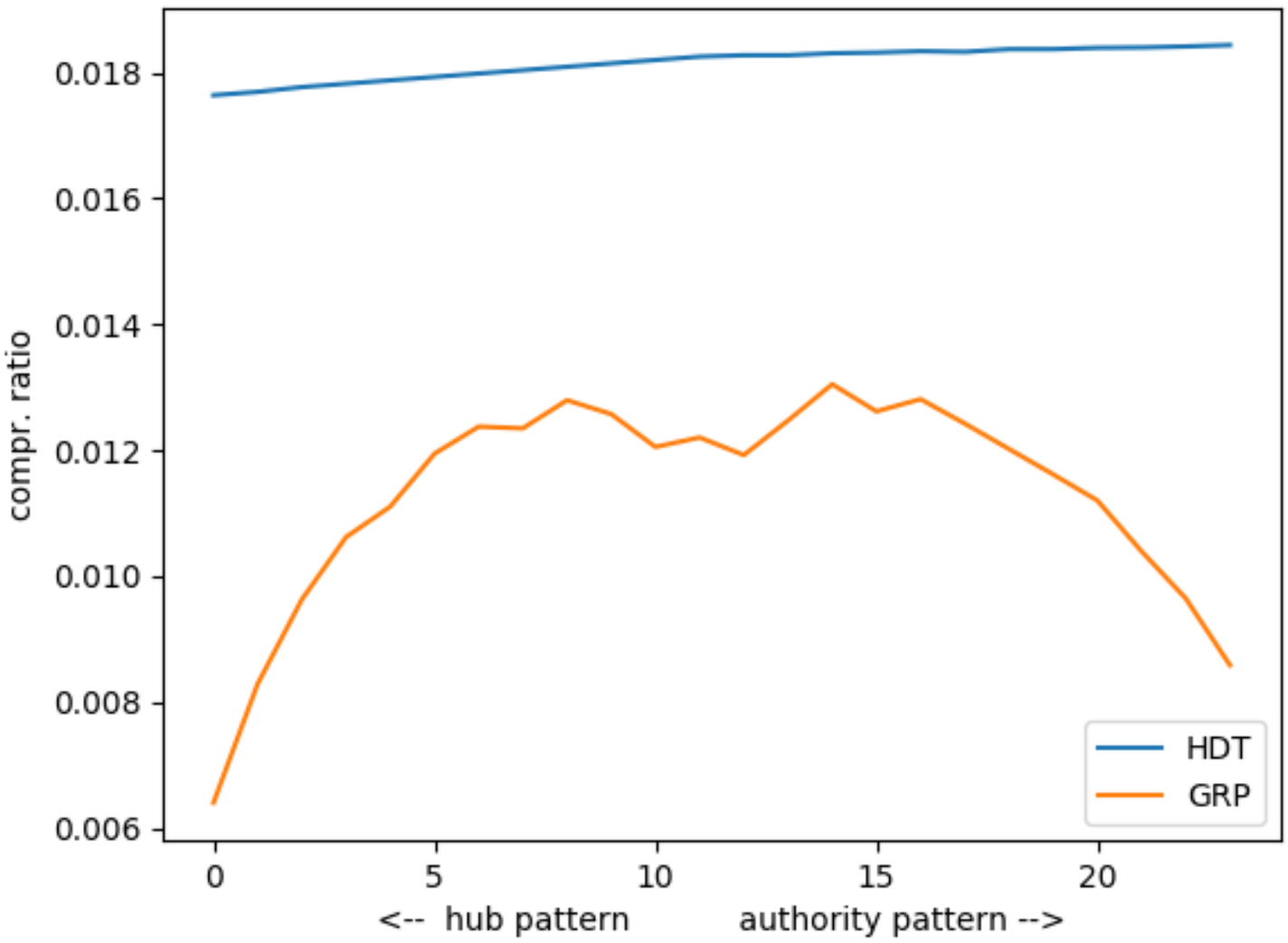
- But in general: Lower number of different edge labels/ properties leads to lower compression ratio

- Also: low compression ratio with star pattern (many digrams around star nodes)

Synthetic data generation

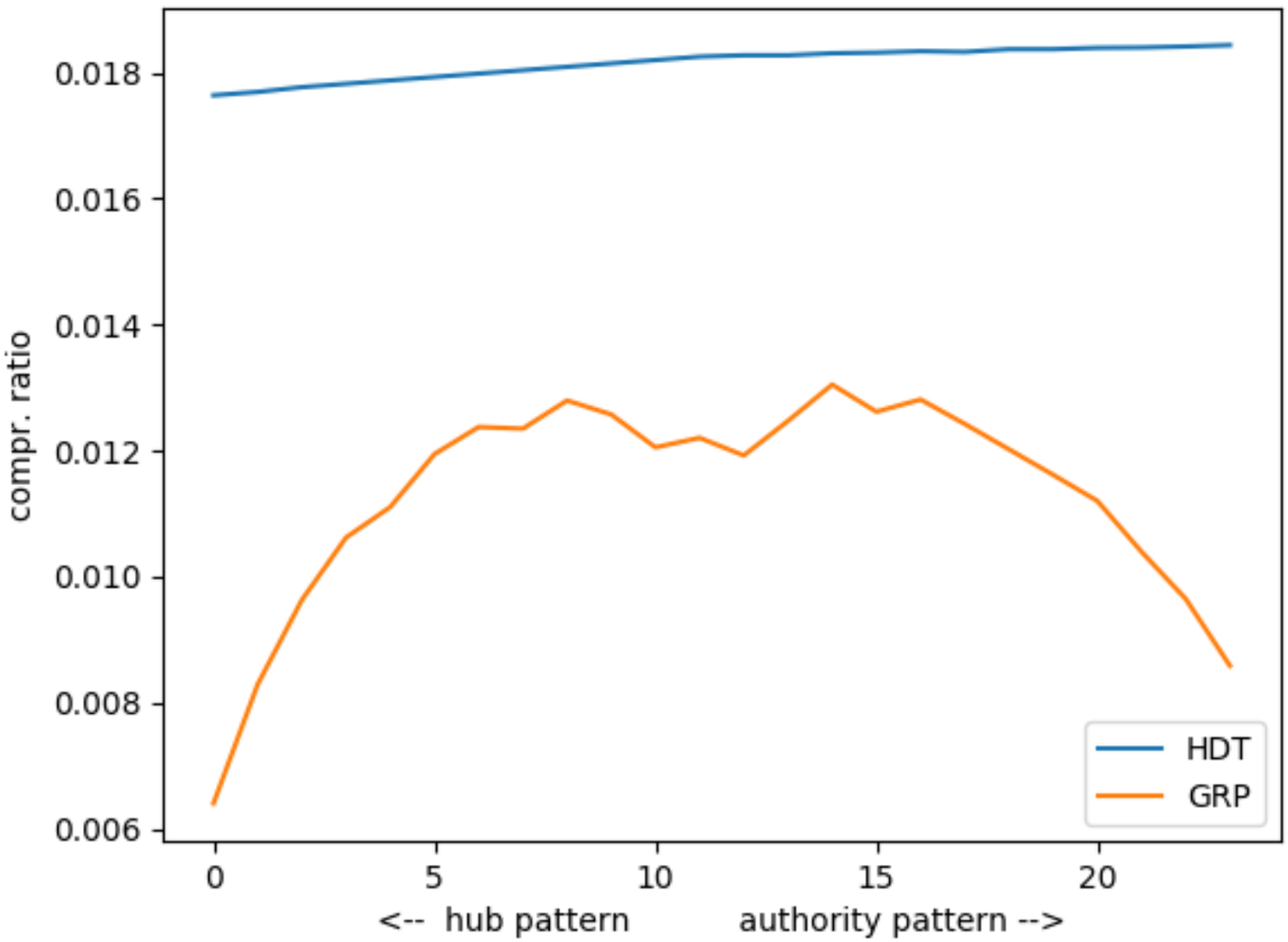


Evaluation results



without dictionary

Evaluation results



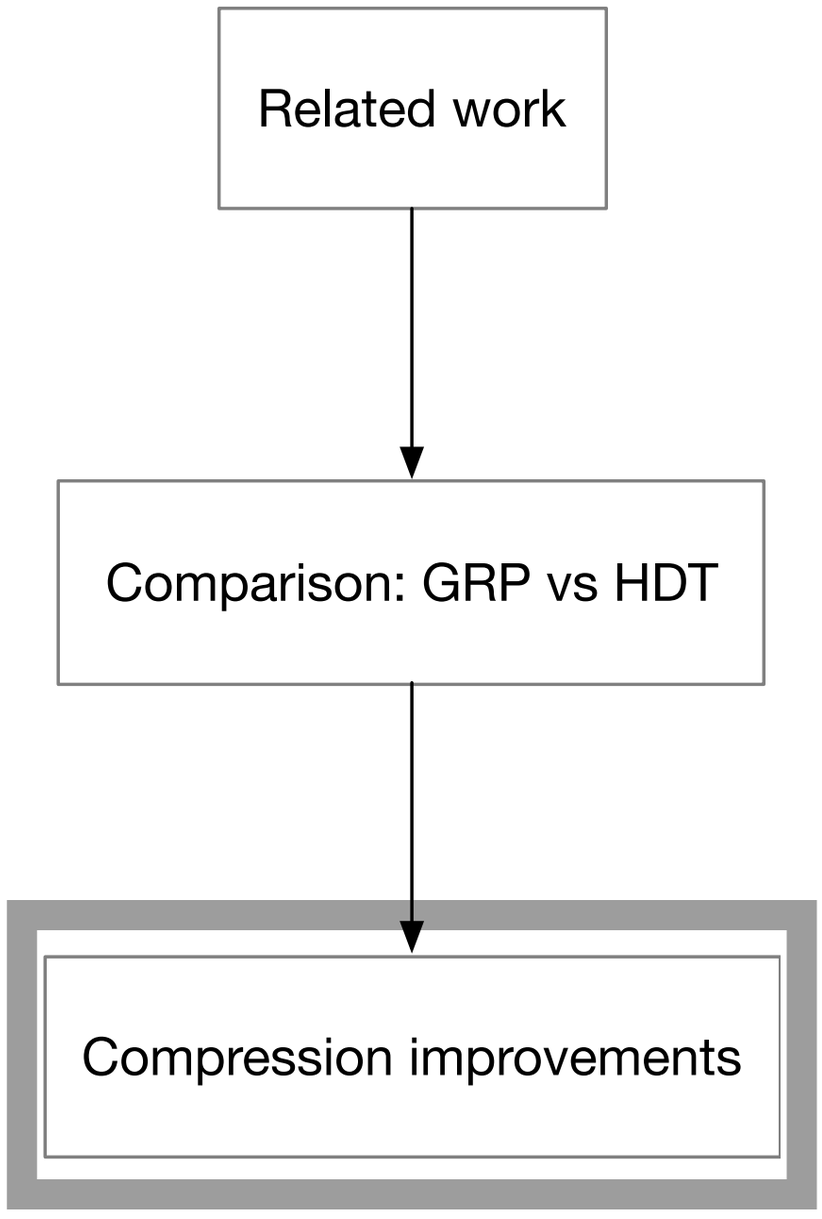
without dictionary

(only one distinct property)

Results on real world data

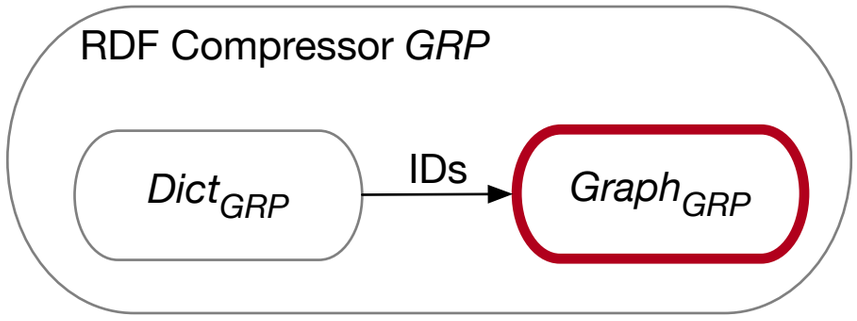
- 33 different graphs from datasets: DBPedia, Wikidata, Scholar Data, GB Government Data (Ministry of housing)
- On 31 of those, GRP outperforms HDT
- On average: HDT’s compr. ratio is 1.8 times higher
- But: run time of GRP is ca. 50 times higher

Next: Compression improvements



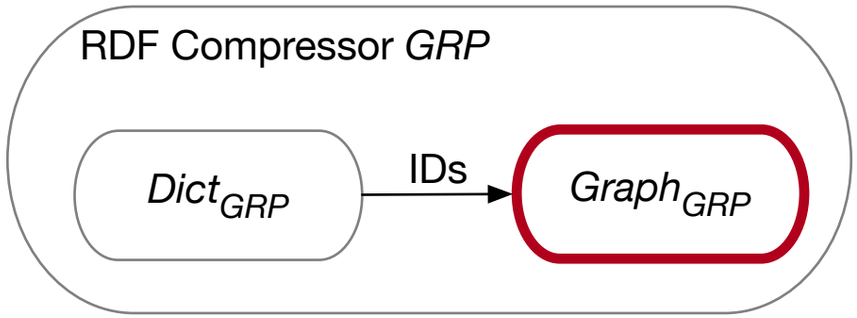
1. **Improve graph compression of GRP**
2. Improve dictionary compression of GRP and HDT
3. Combination of improved graph compression & dictionary

Improve graph compression

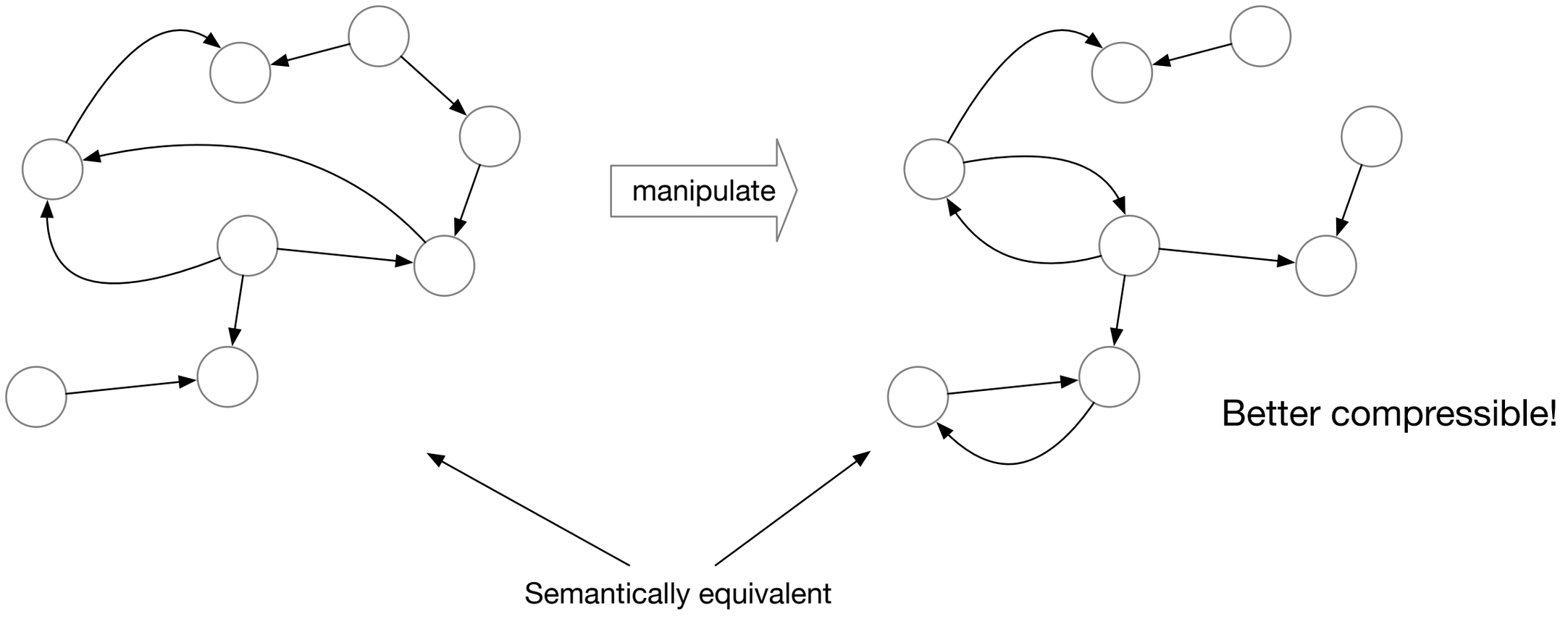


$$CR_{Graph_{GRP}} = \frac{|\,out_{graph}\,|}{|\,in\,|}$$

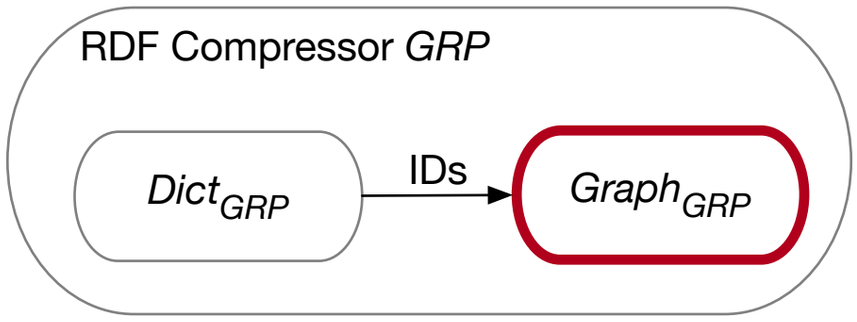
Improve graph compression



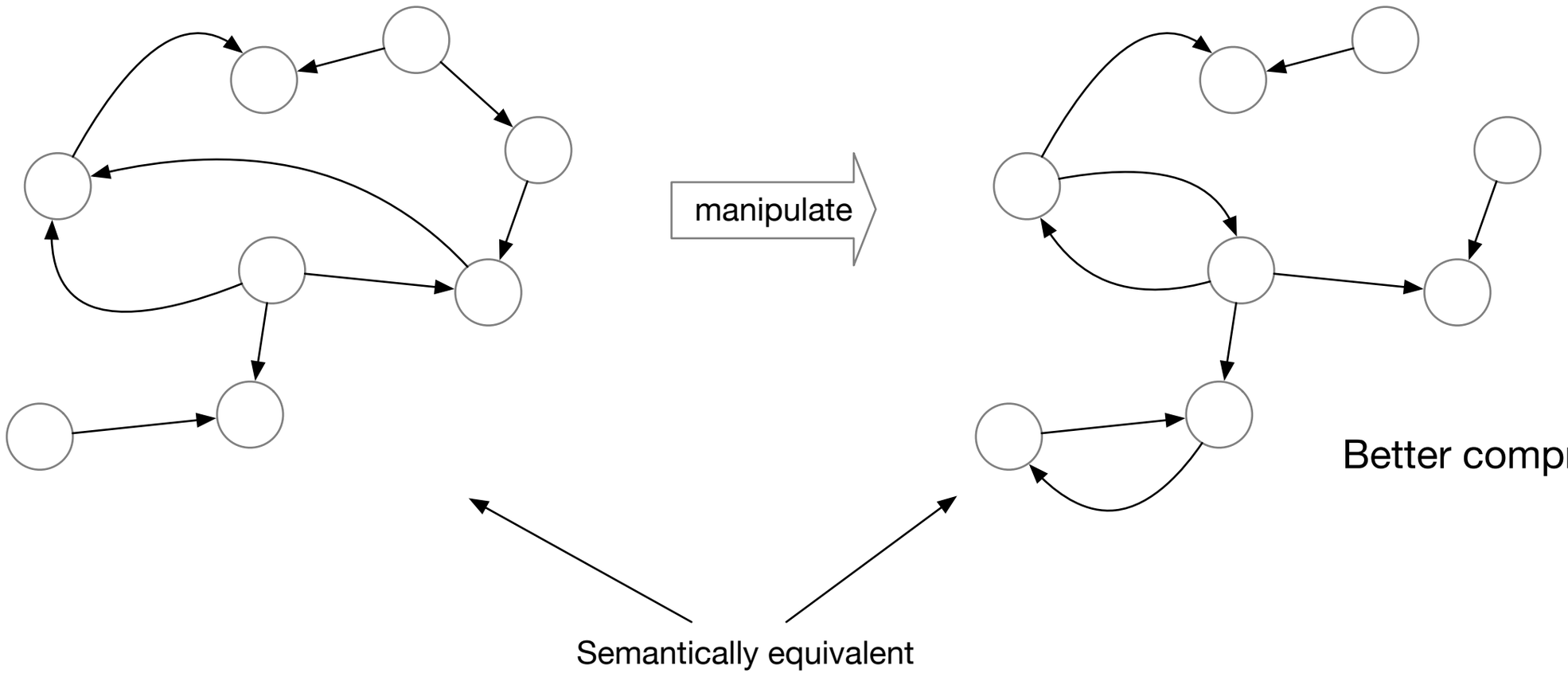
$$CR_{Graph_{GRP}} = \frac{|out_{graph}|}{|in|}$$



Improve graph compression

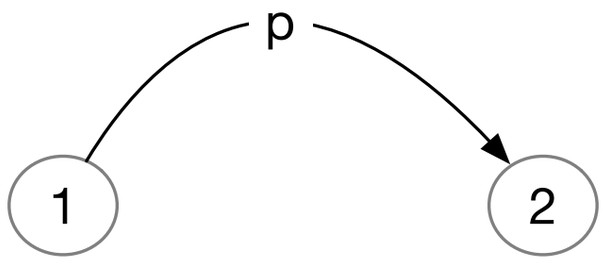


$$CR_{Graph_{GRP}} = \frac{|out_{graph}|}{|in|}$$

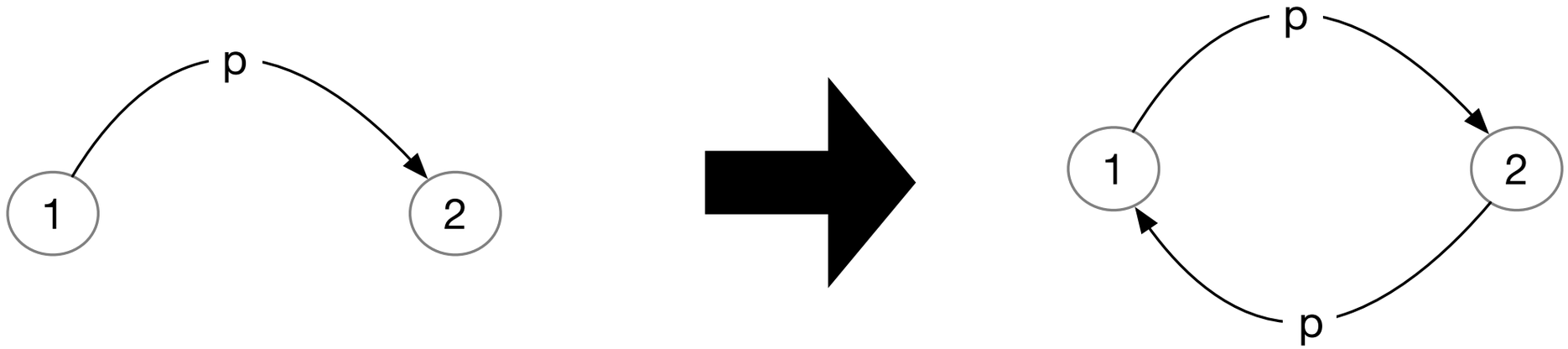


Parts of ontology have to be stored as well!

Symmetric or inverse properties

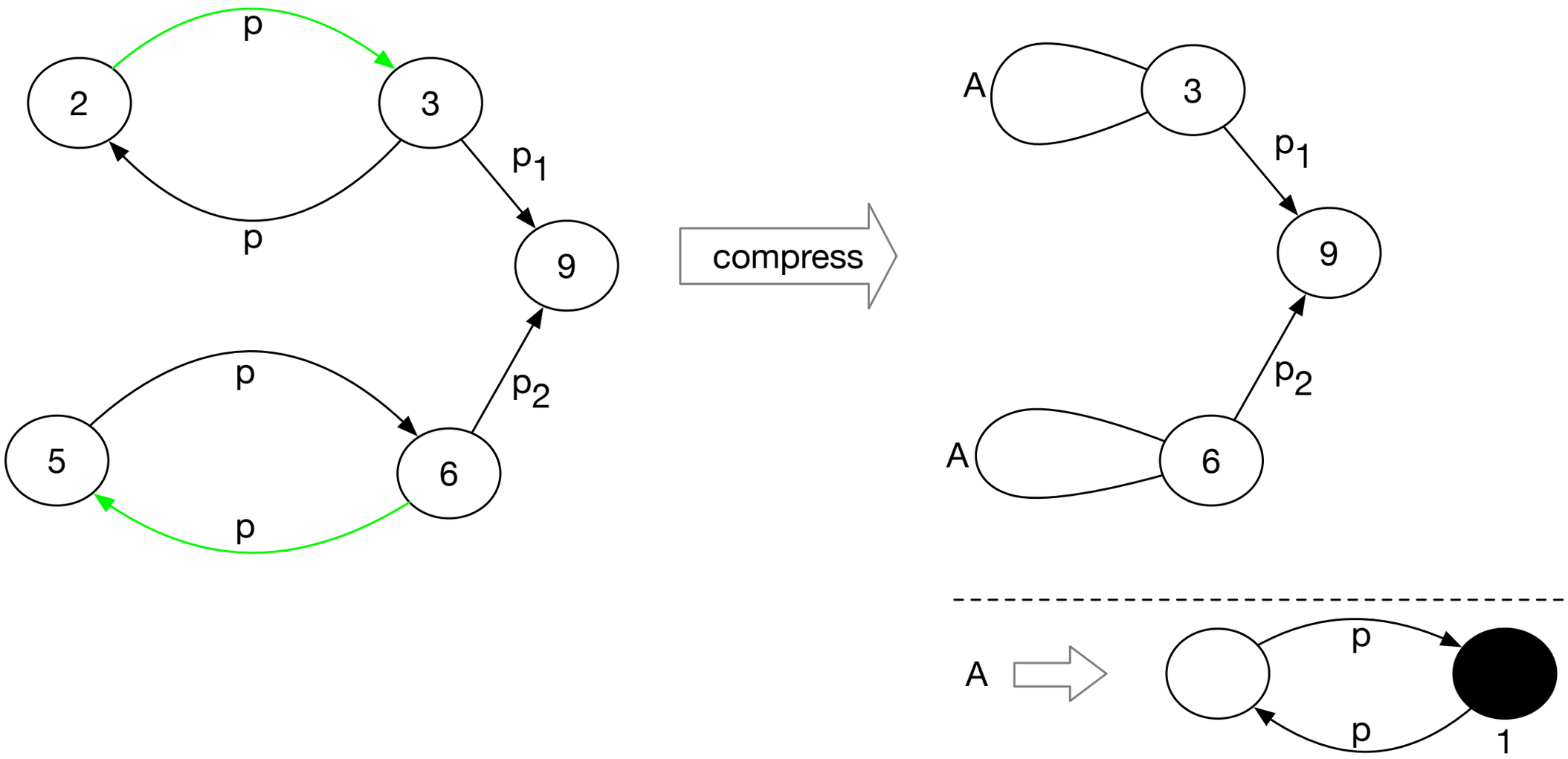


Symmetric or inverse properties

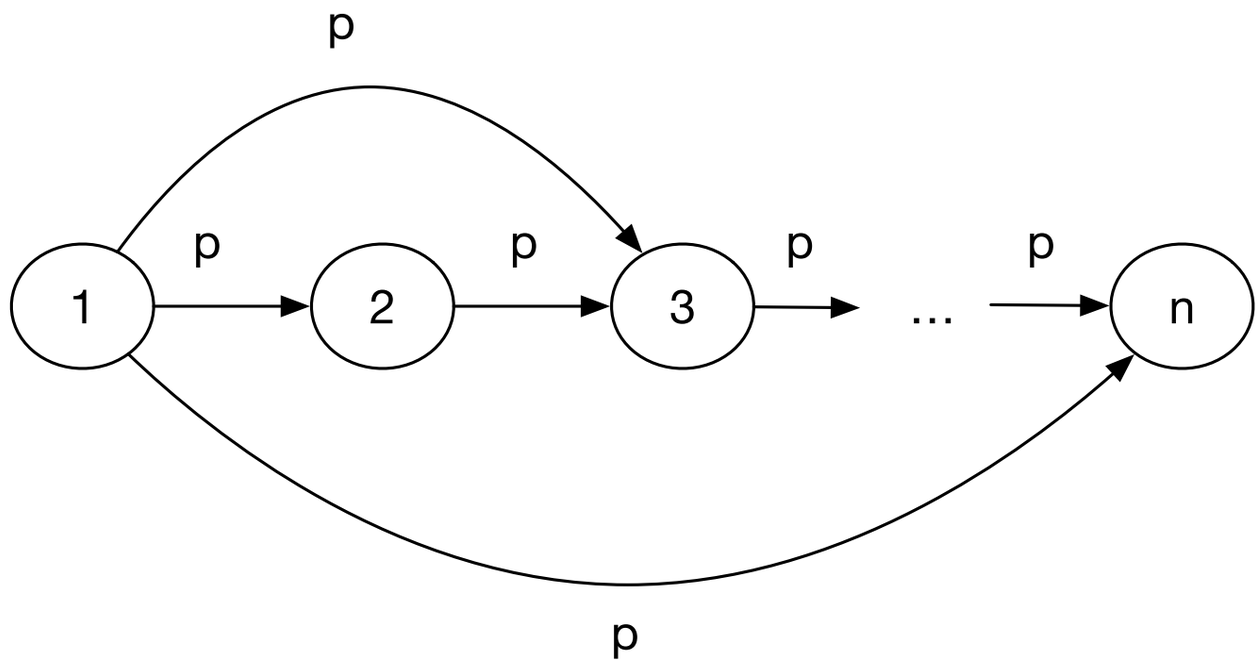


Approach: add all possible triple (with symmetric or inverse properties)

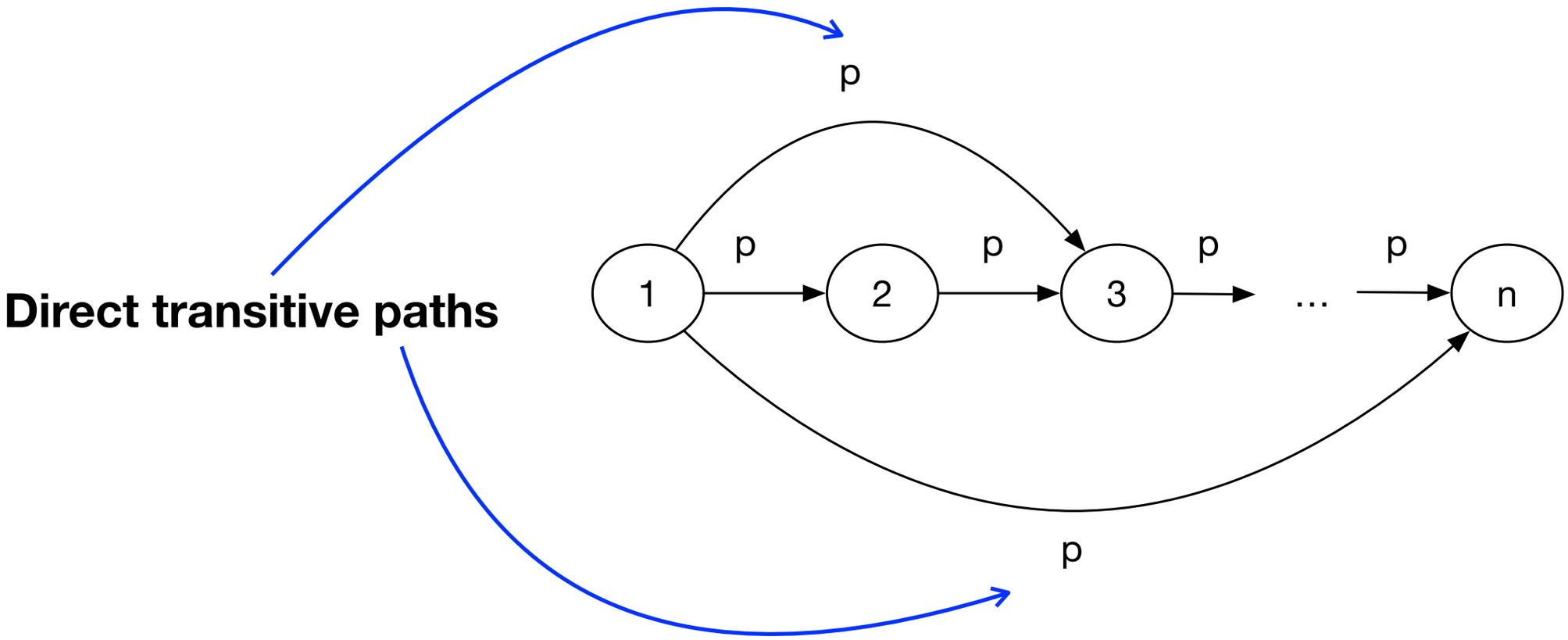
Symmetric or inverse properties



Transitive properties

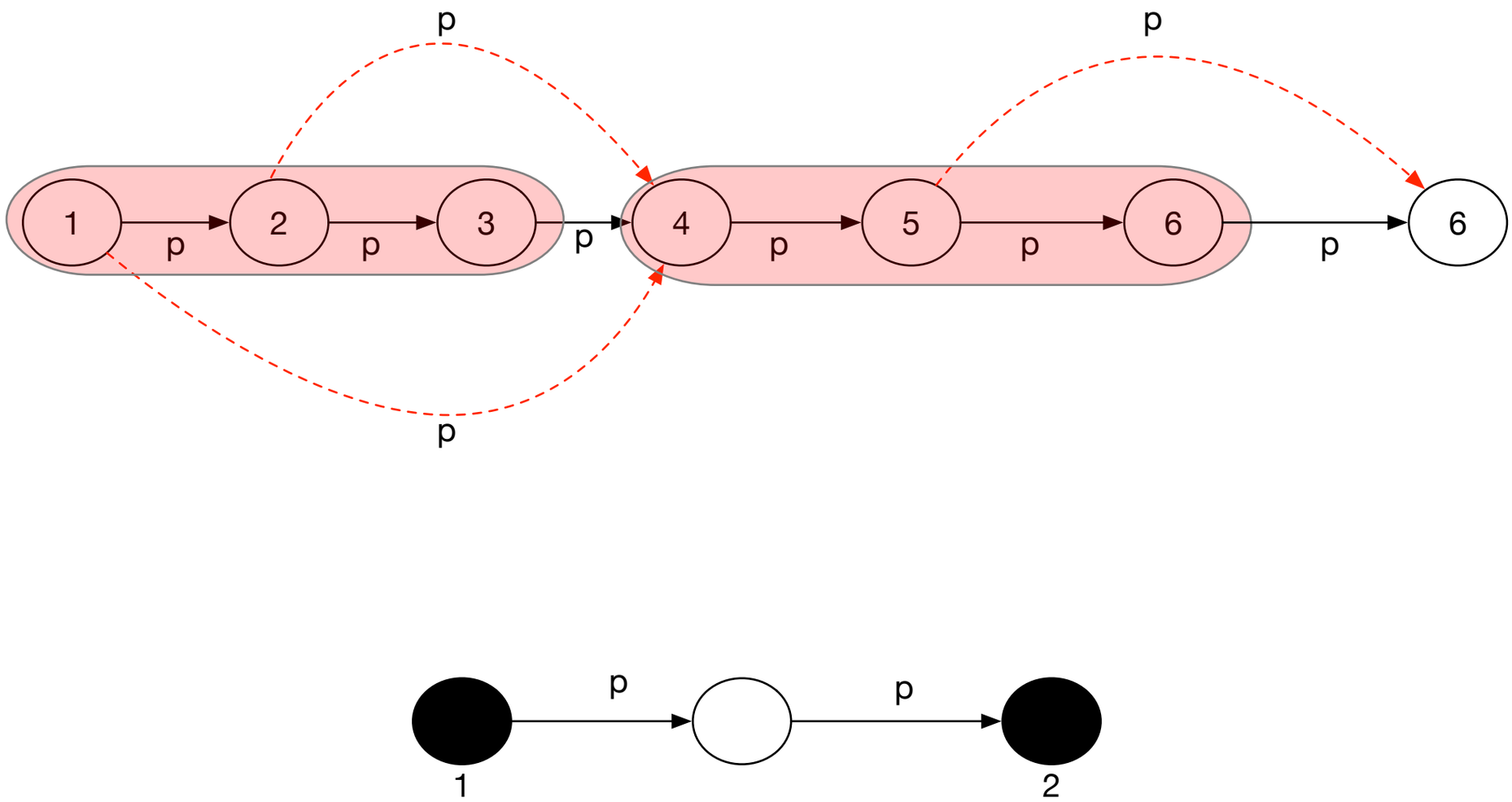


Transitive properties

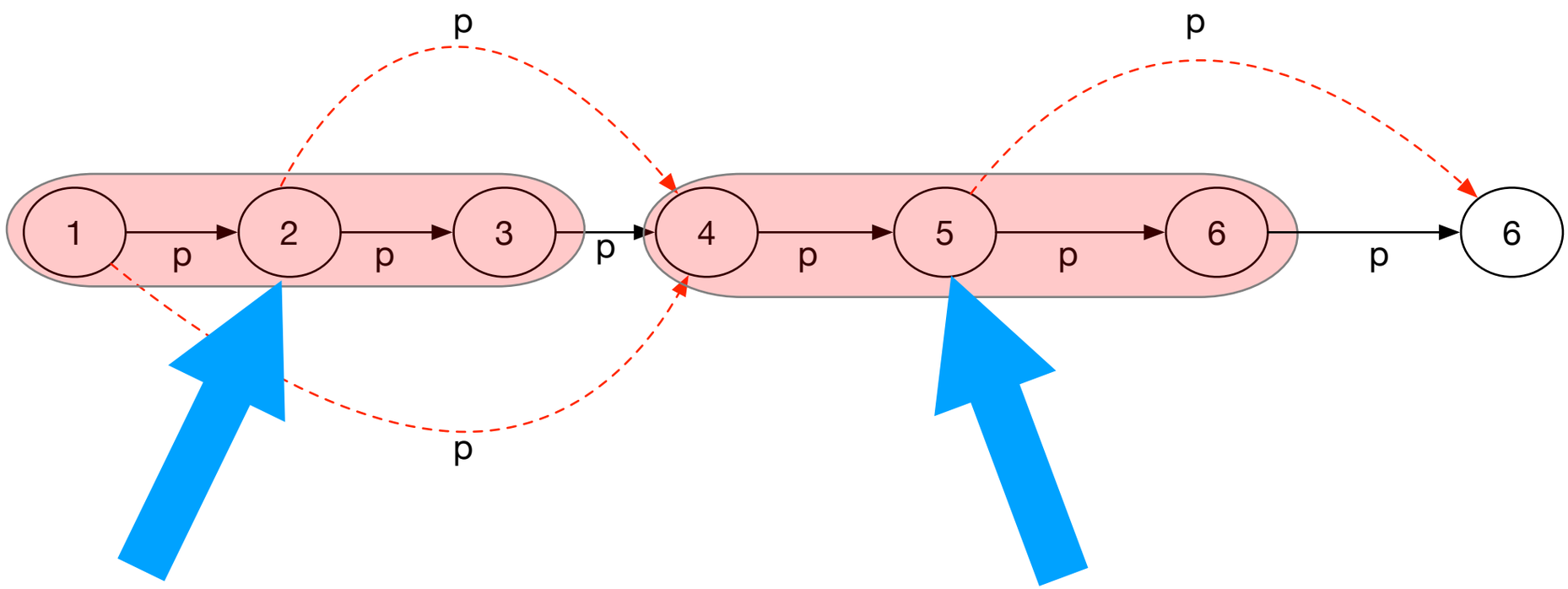


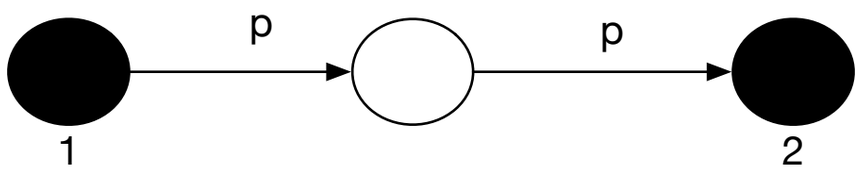
Idea: Remove all direct transitive paths!

Transitive properties

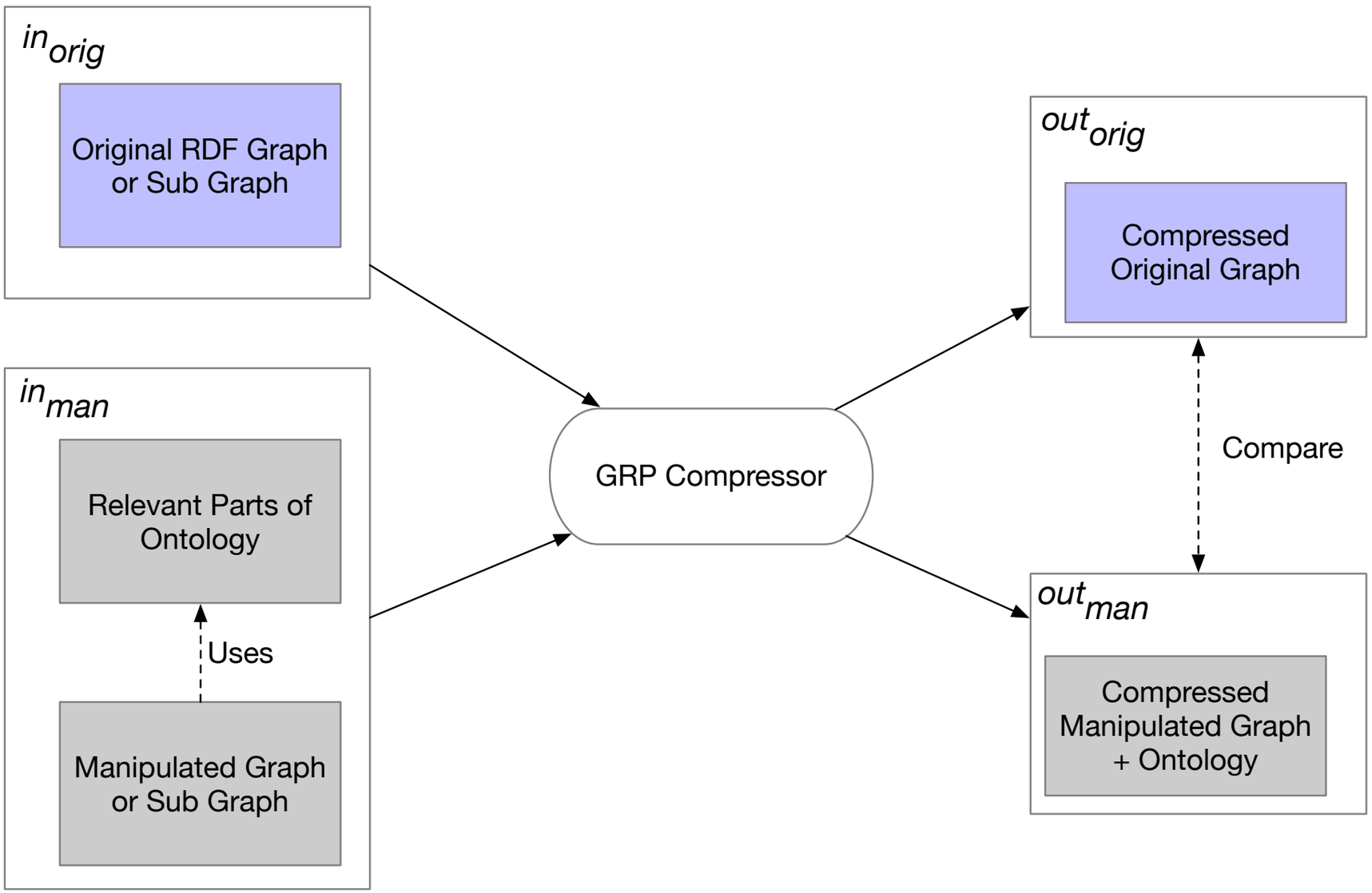


Transitive properties

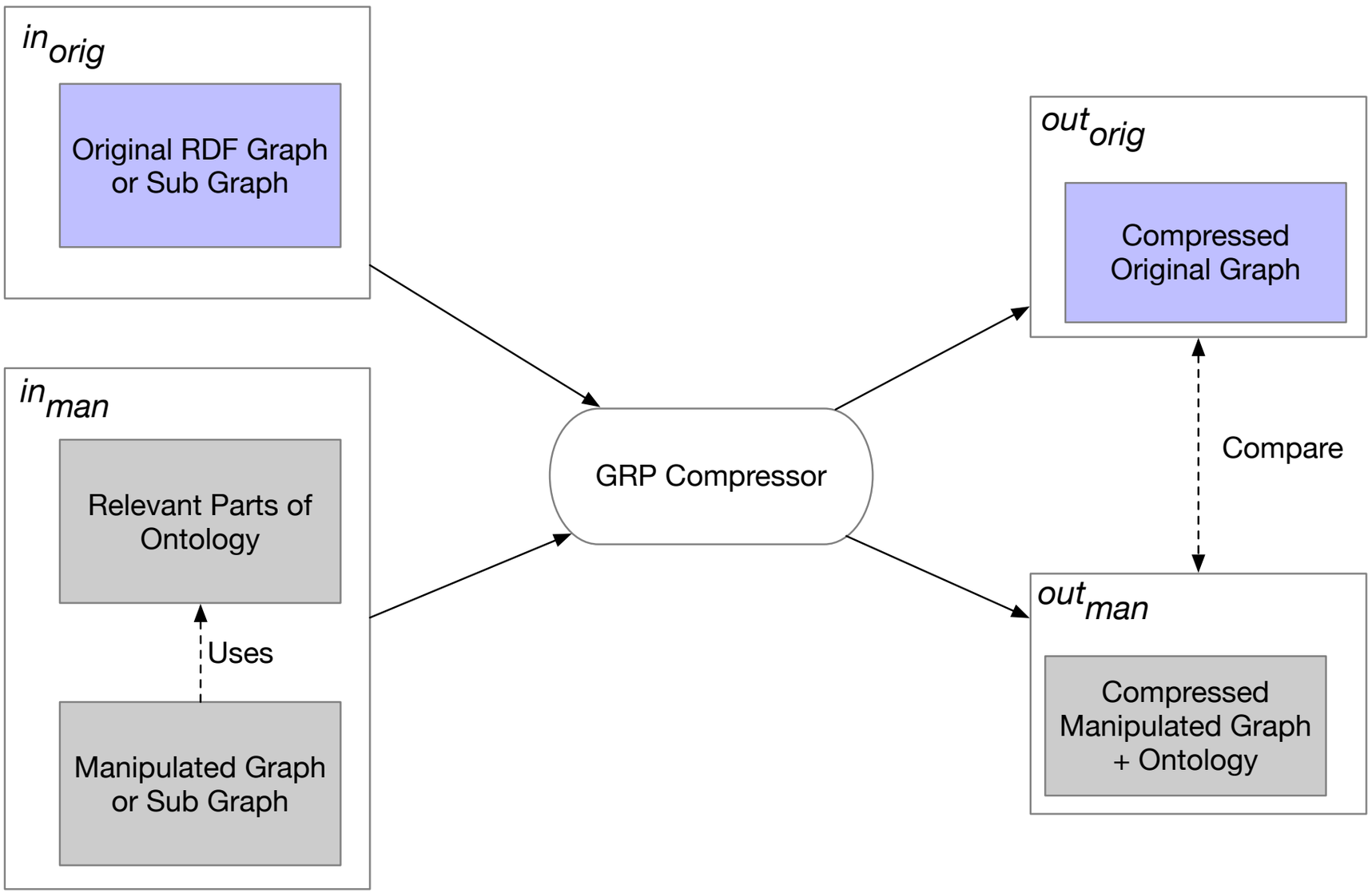




Evaluation

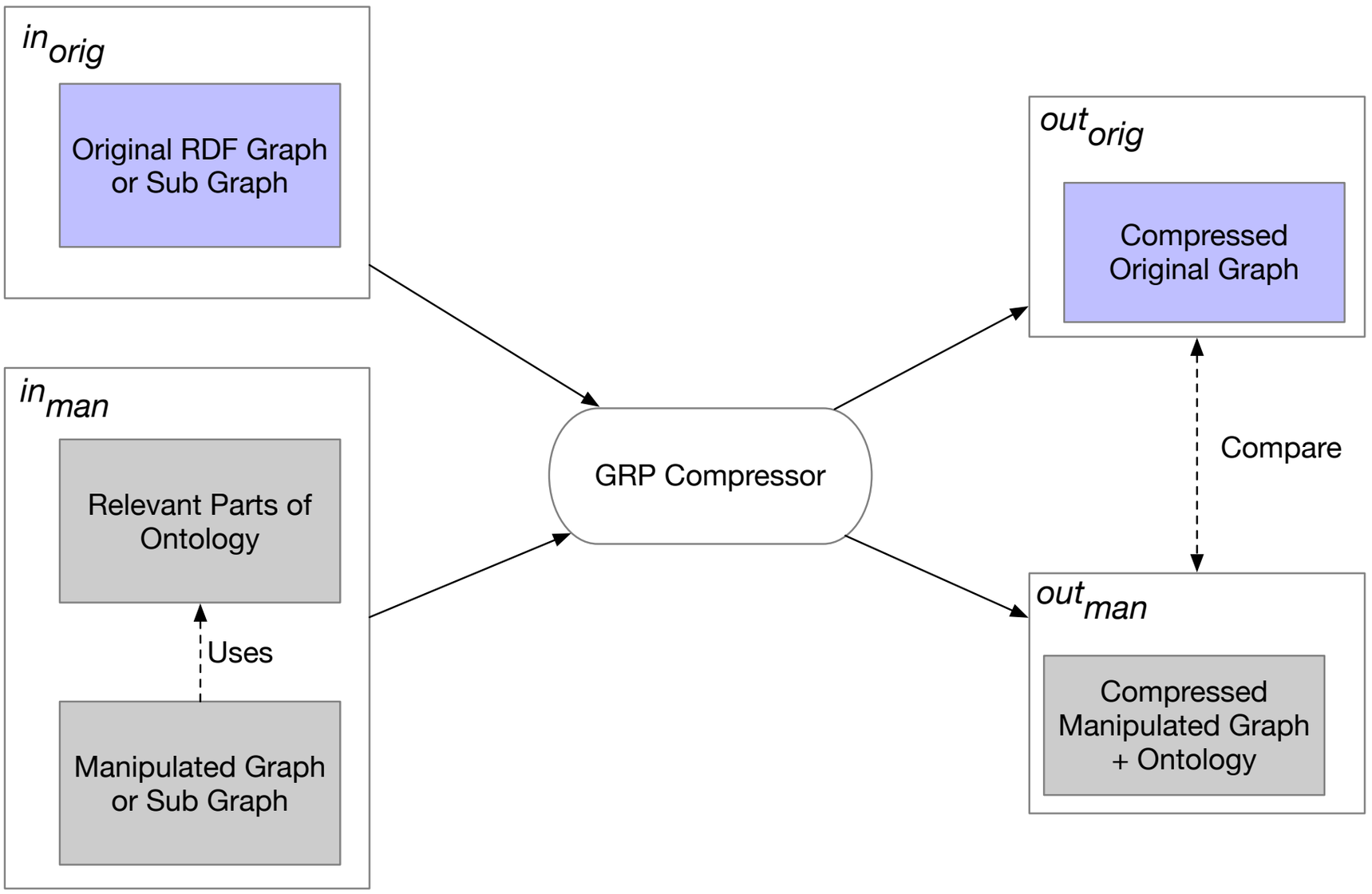

in orig
Original RDF Graph or Sub Graph
in man
Relevant Parts of Ontology
Uses
Manipulated Graph or Sub Graph
GRP Compressor
out orig
Compressed Original Graph
Compare
out man
Compressed Manipulated Graph + Ontology

Evaluation



$$IER = \frac{\textbf{\#edges in } in_{man}}{\textbf{\#edges in } in_{orig}}$$

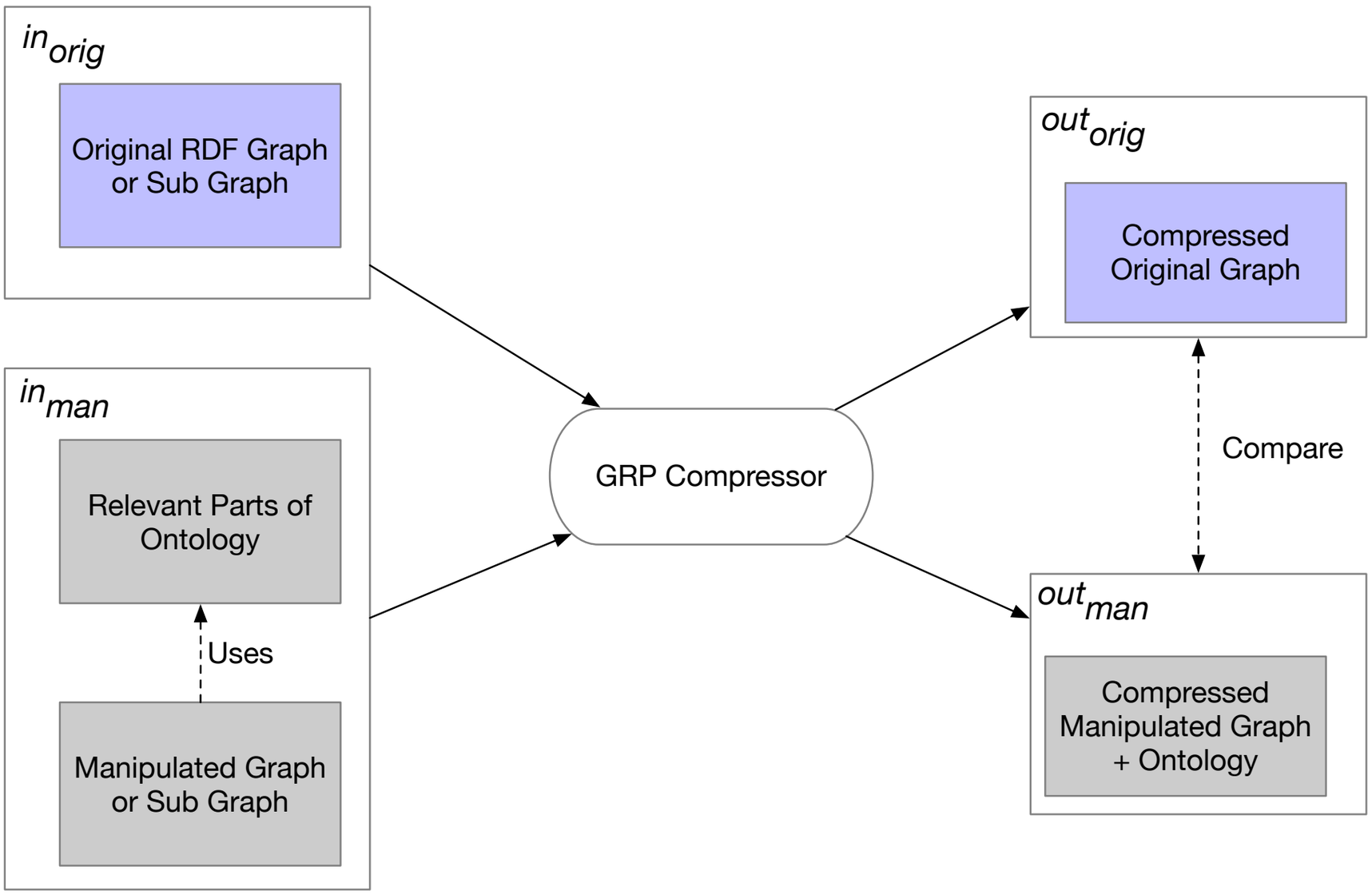
Evaluation



$$IER = \frac{\textbf{\#edges in } in_{man}}{\textbf{\#edges in } in_{orig}}$$

$$OER = \frac{\textbf{\#edges in } out_{man}}{\textbf{\#edges in } out_{orig}}$$

Evaluation

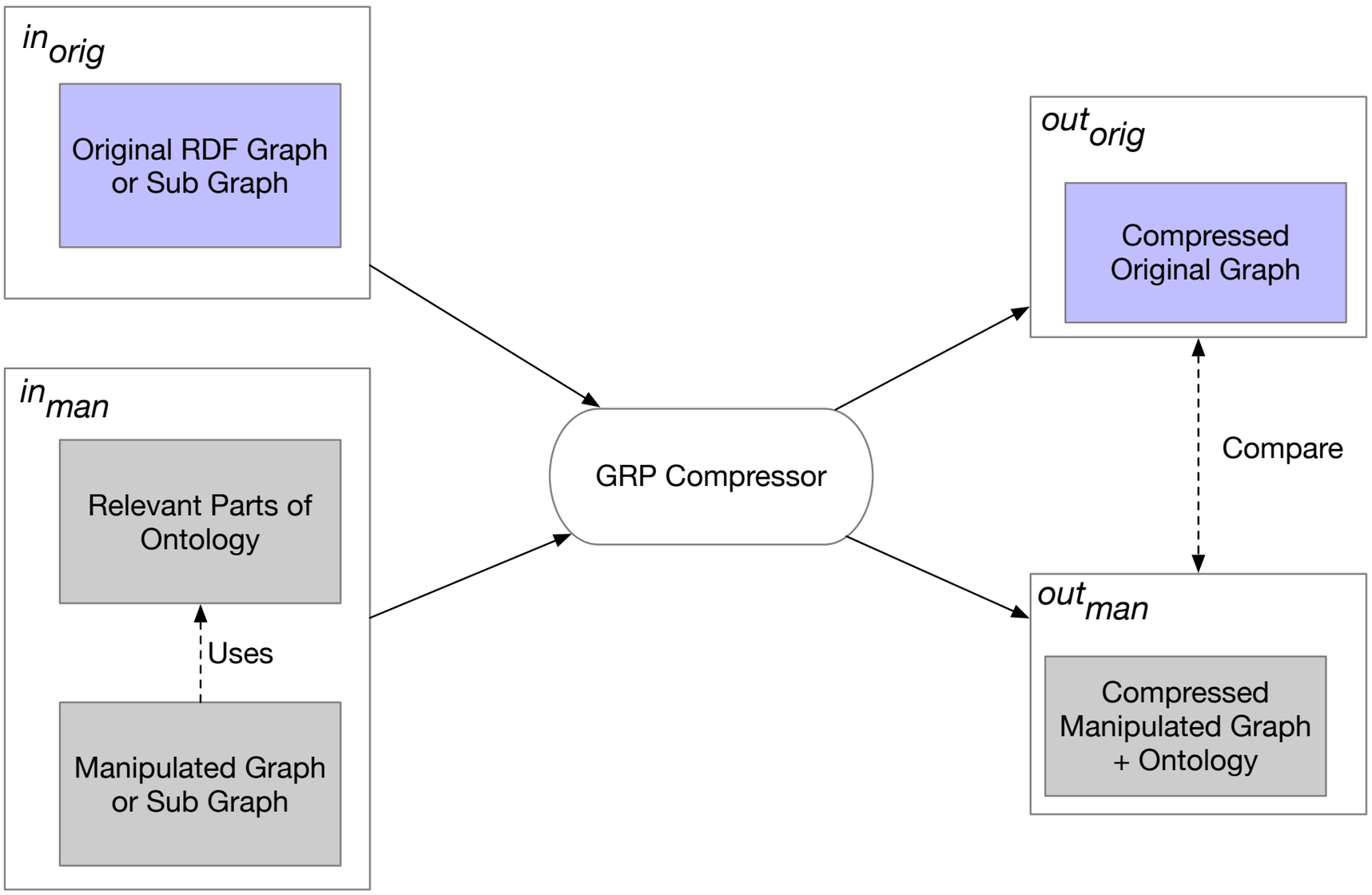


$$IER = \frac{\textbf{\#edges in } in_{man}}{\textbf{\#edges in } in_{orig}}$$

$$OER = \frac{\textbf{\#edges in } out_{man}}{\textbf{\#edges in } out_{orig}}$$

$$SR = \frac{|\, out_{graph_{man}} \,|}{|\, out_{graph_{orig}} \,|}$$

Evaluation



Grammar level:

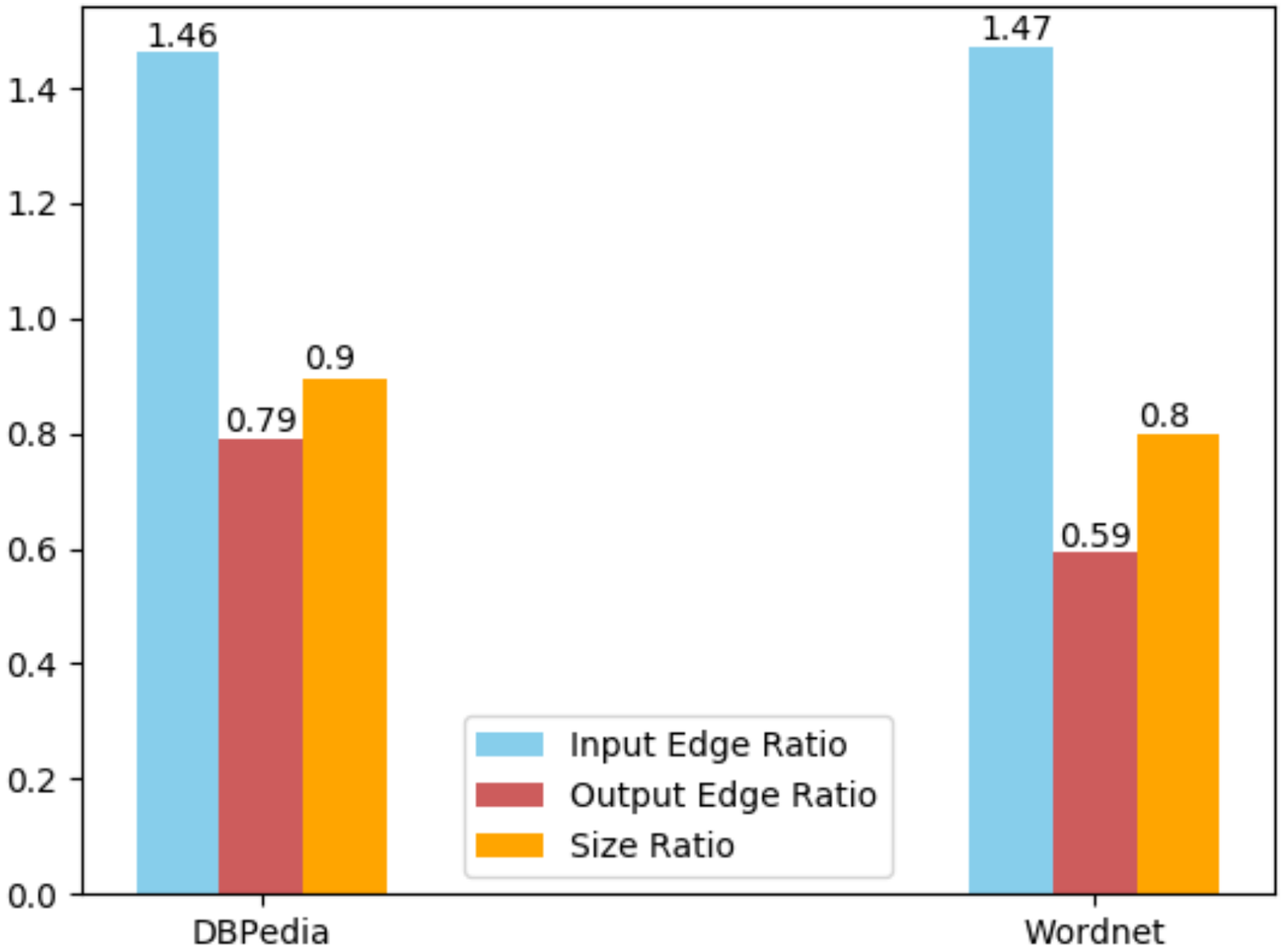
$$IER = \frac{\textbf{\#edges in } in_{man}}{\textbf{\#edges in } in_{orig}}$$

$$OER = \frac{\textbf{\#edges in } out_{man}}{\textbf{\#edges in } out_{orig}}$$

File size level:

$$SR = \frac{|\, out_{graph_{man}} \,|}{|\, out_{graph_{orig}} \,|}$$

Results: adding symmetric properties



Grammar level:

$$IER = \frac{\textbf{\#edges in } in_{man}}{\textbf{\#edges in } in_{orig}}$$

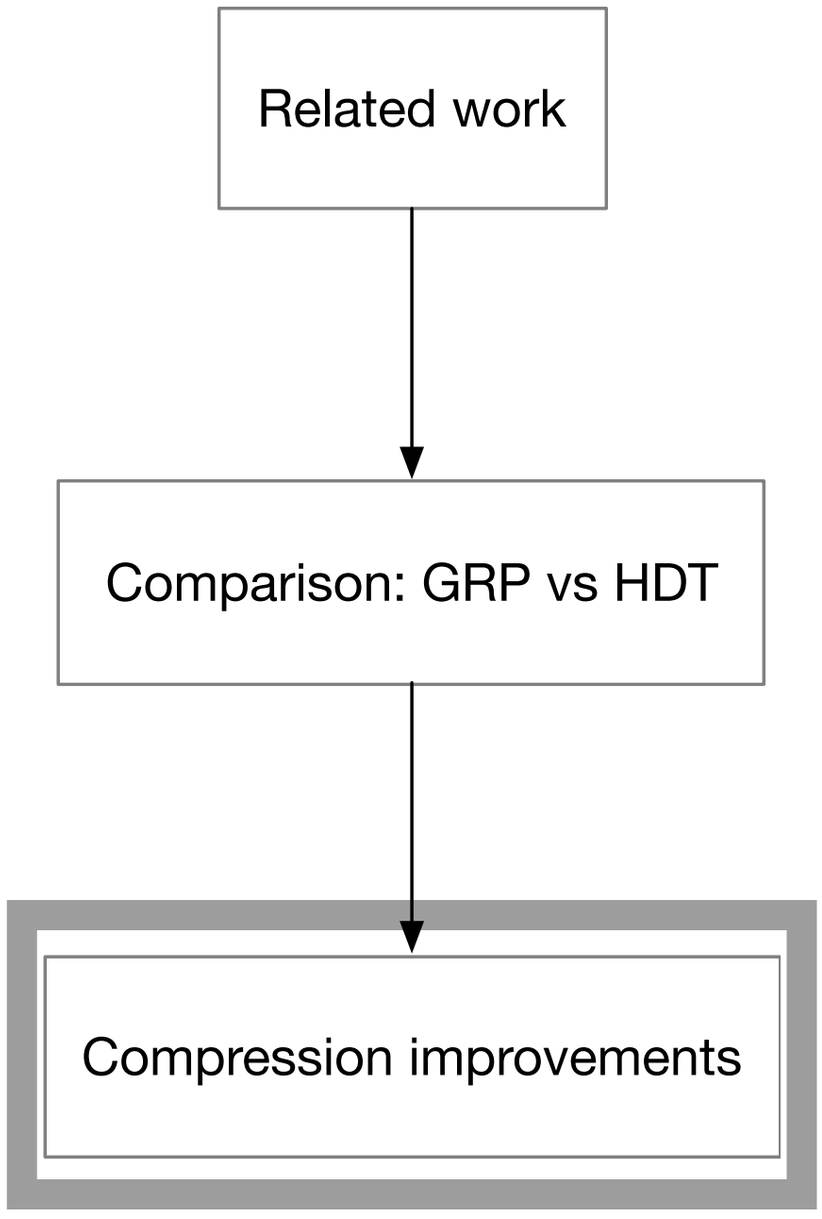
$$OER = \frac{\textbf{\#edges in } out_{man}}{\textbf{\#edges in } out_{orig}}$$

File size level:

$$SR = \frac{|\, out_{graph_{man}} \,|}{|\, out_{graph_{orig}} \,|}$$

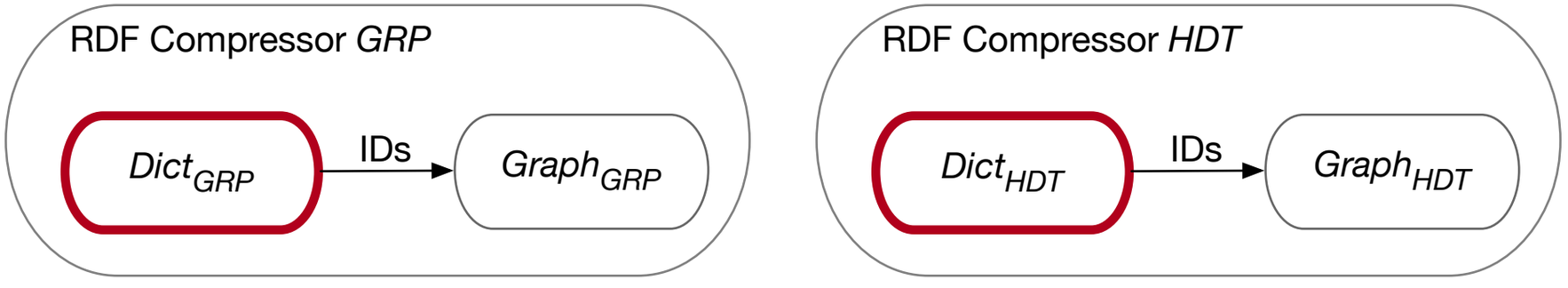
In original graph: only one direction for all triple pairs

Further: Compression improvements



1. Improve graph compression of GRP
2. **Improve dictionary compression of GRP and HDT**
3. Combination of improved graph compression & dictionary

Dictionary Compression Improvements



$$CR_{Dict_C} = \frac{|out_{dict}|}{|in|}$$

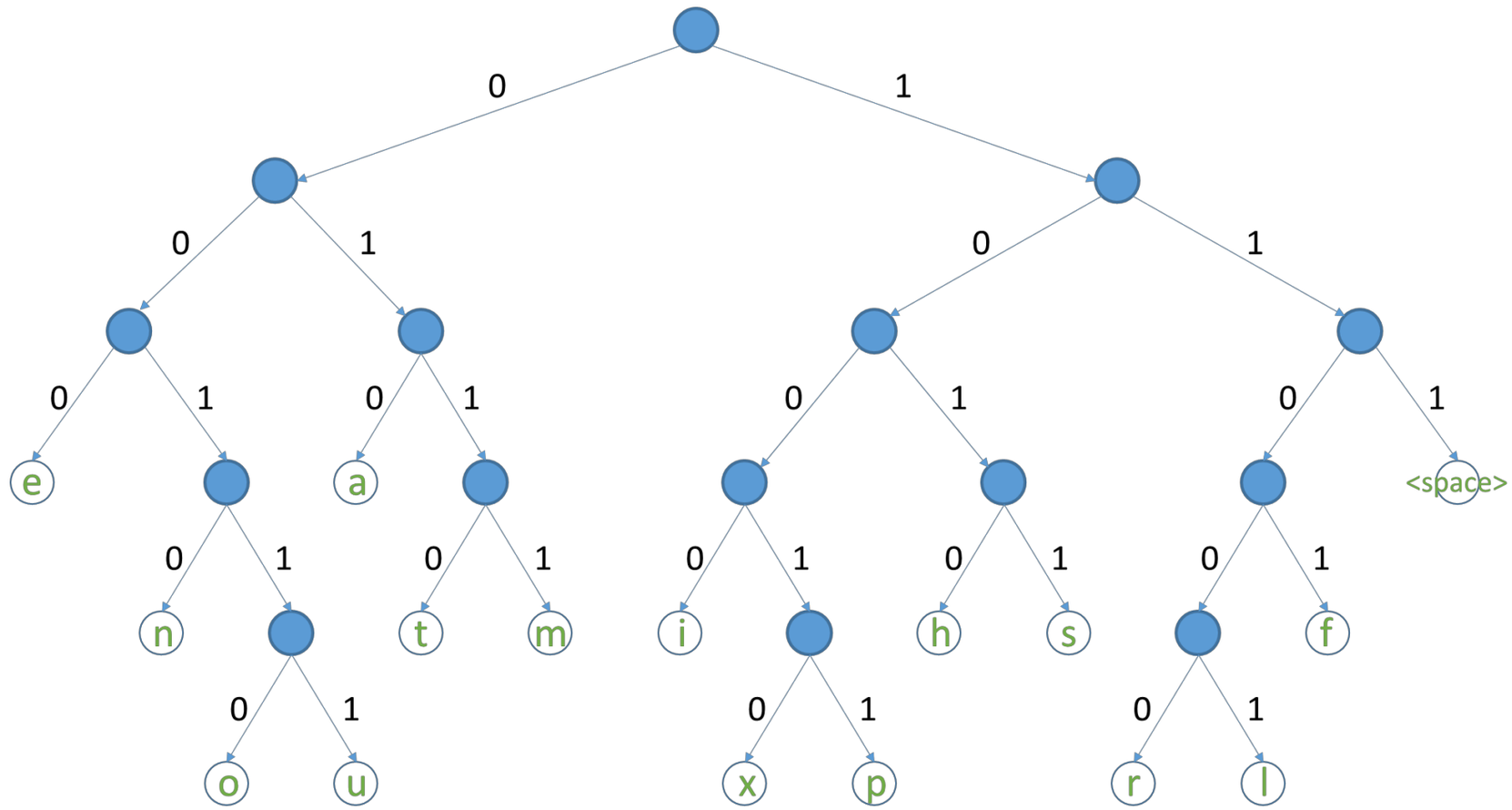
Dictionary Compression Improvements

$$CR_{Dict_C} = \frac{|out_{dict}|}{|in|}$$

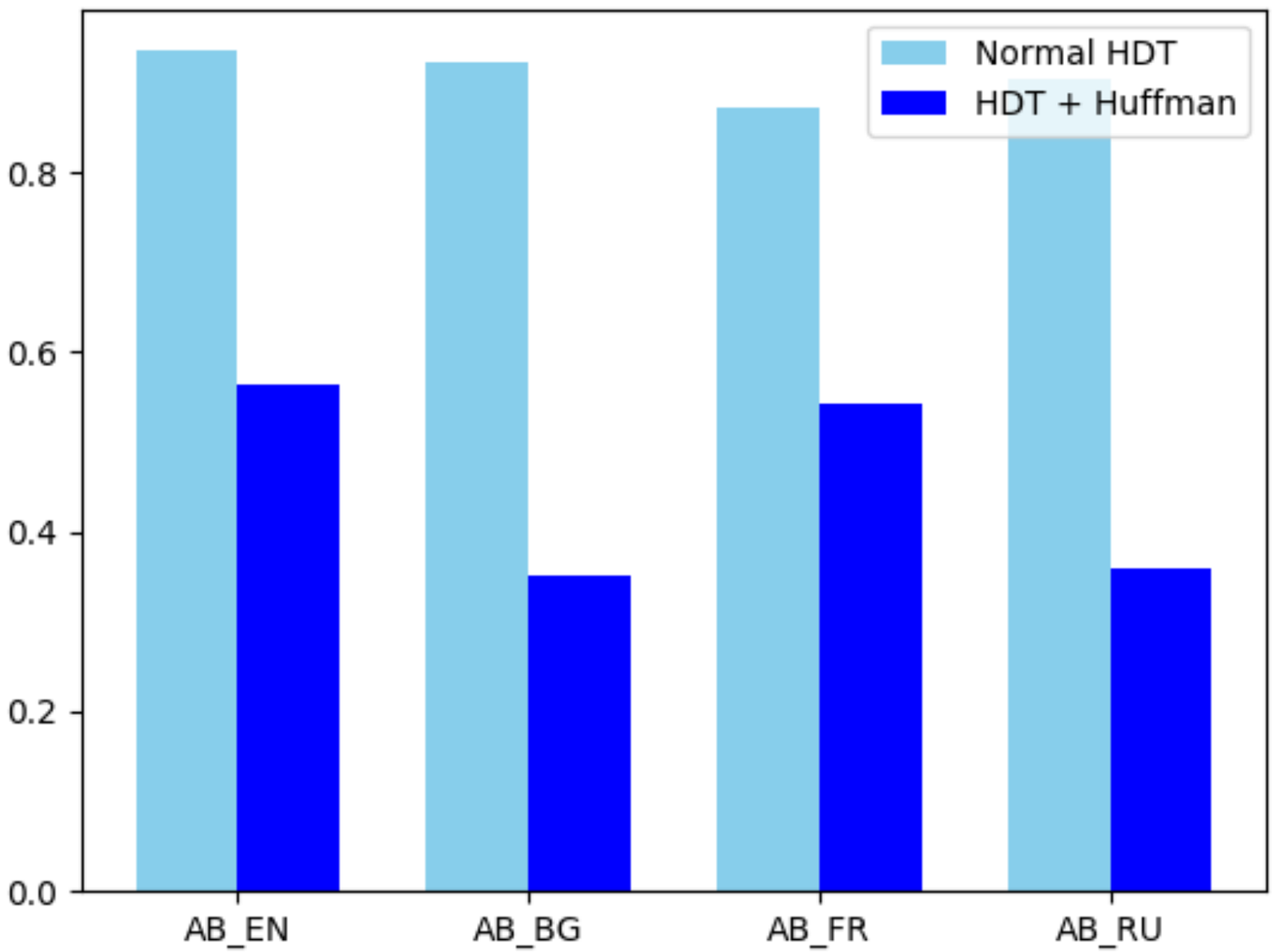
- Already uses prefix-based text compression (good for URIs)
- Not good for **literals** and blank nodes (rarely common prefixes here)

Literals

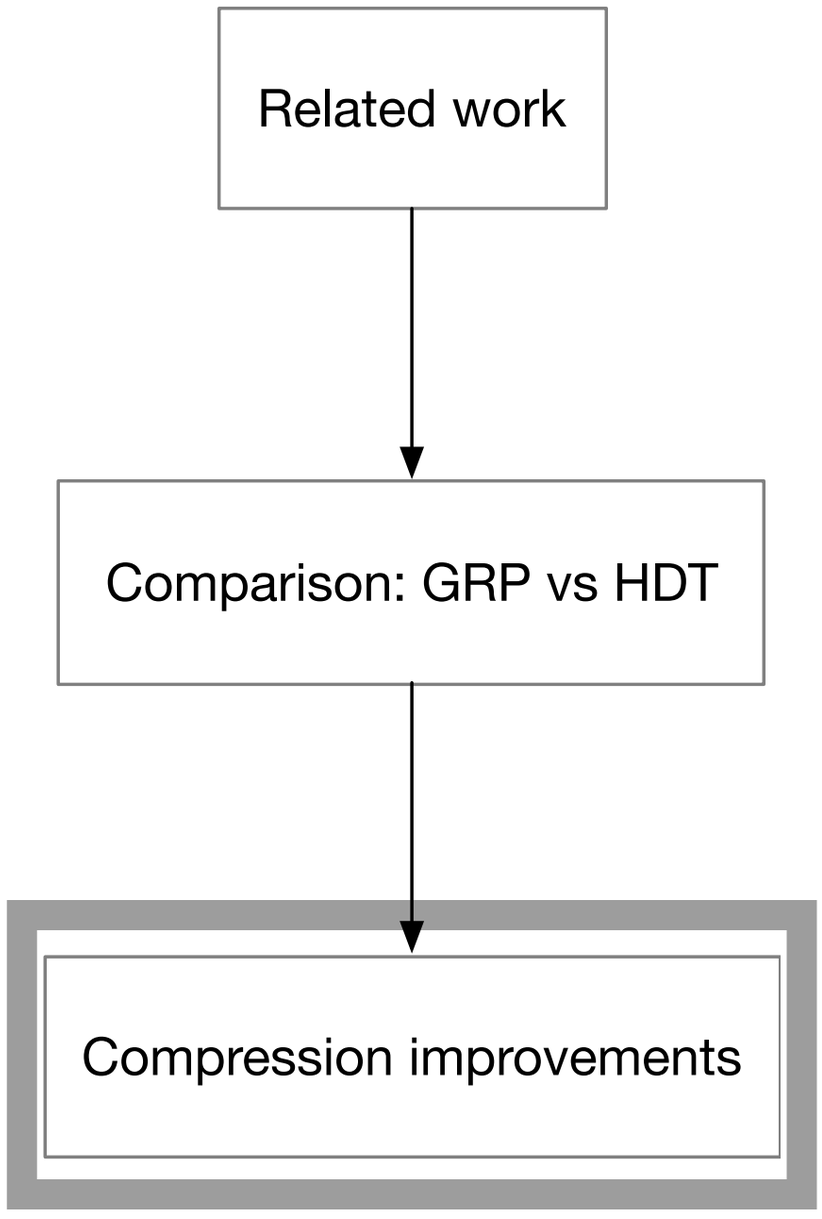
- Contain different data values (e.g. string, numbers)
- Text compression for strings: Huffman
- Compute Huffman Code in advance



Literals: Results for DBPedia Abstracts

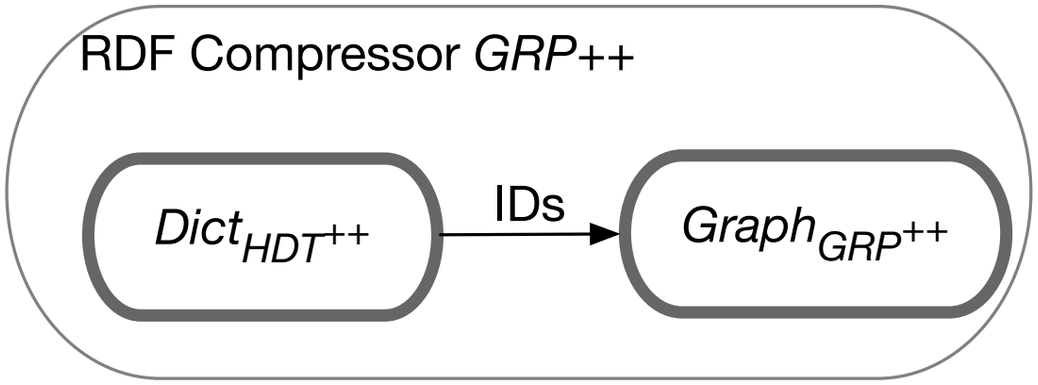


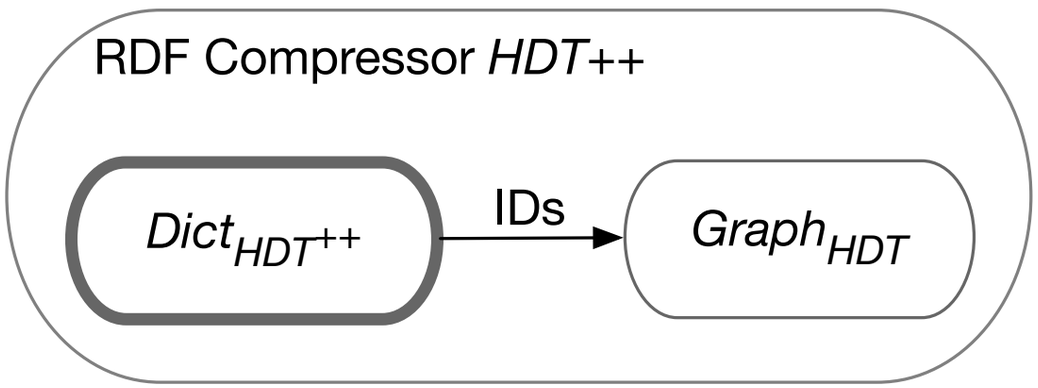
Further: Compression improvements



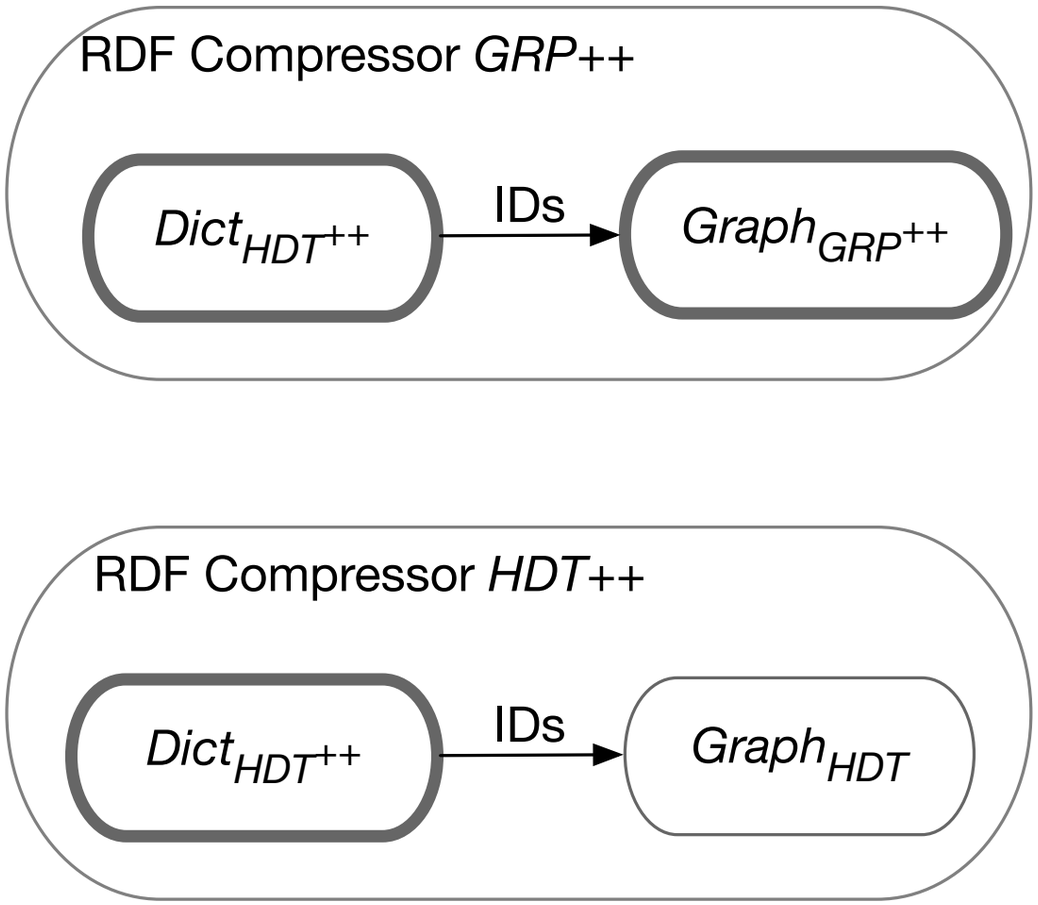
1. Improve graph compression of GRP
2. Improve dictionary compression of GRP and HDT
3. **Combination of improved graph compression & dictionary**

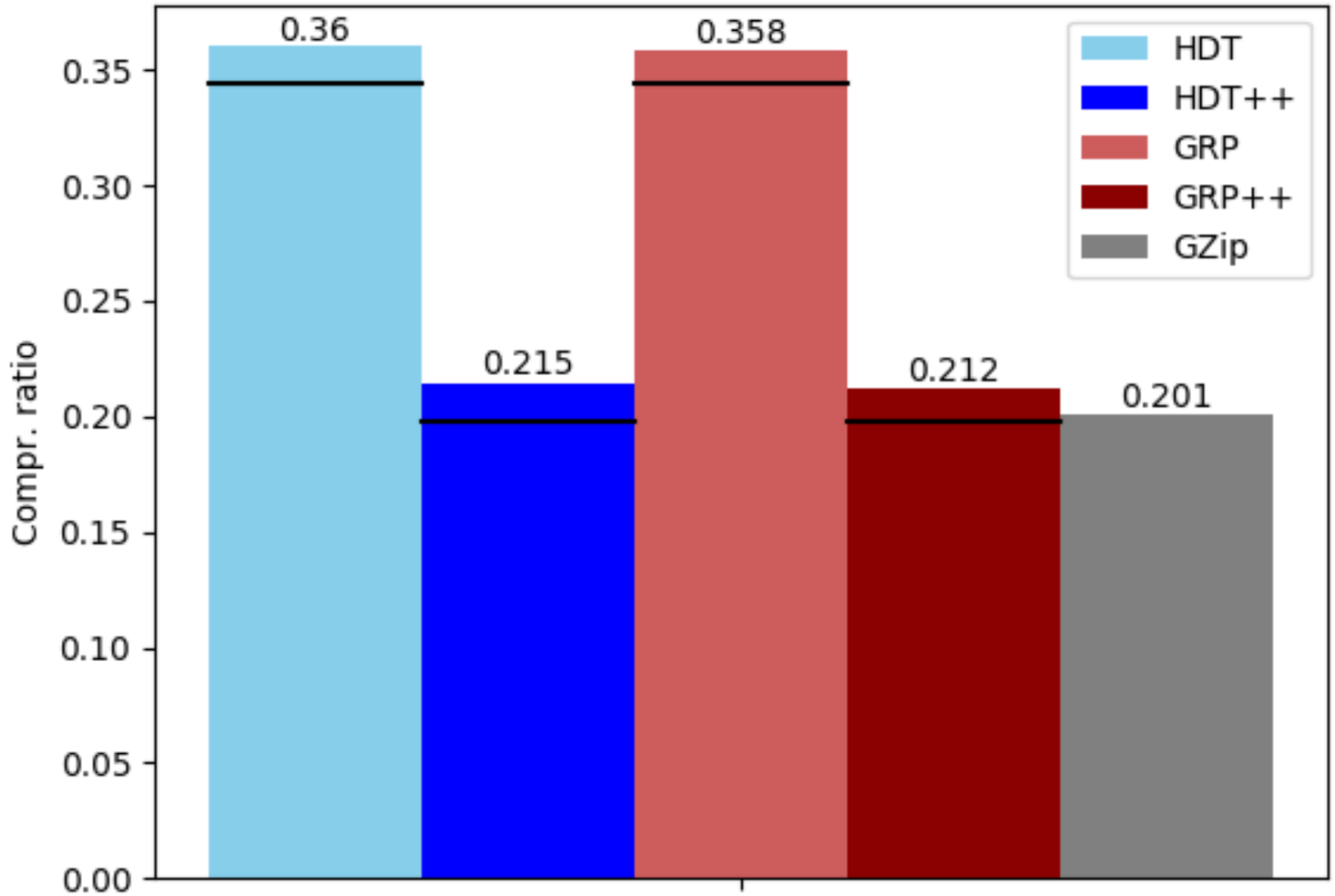
Combined evaluation





Combined evaluation





Conclusion

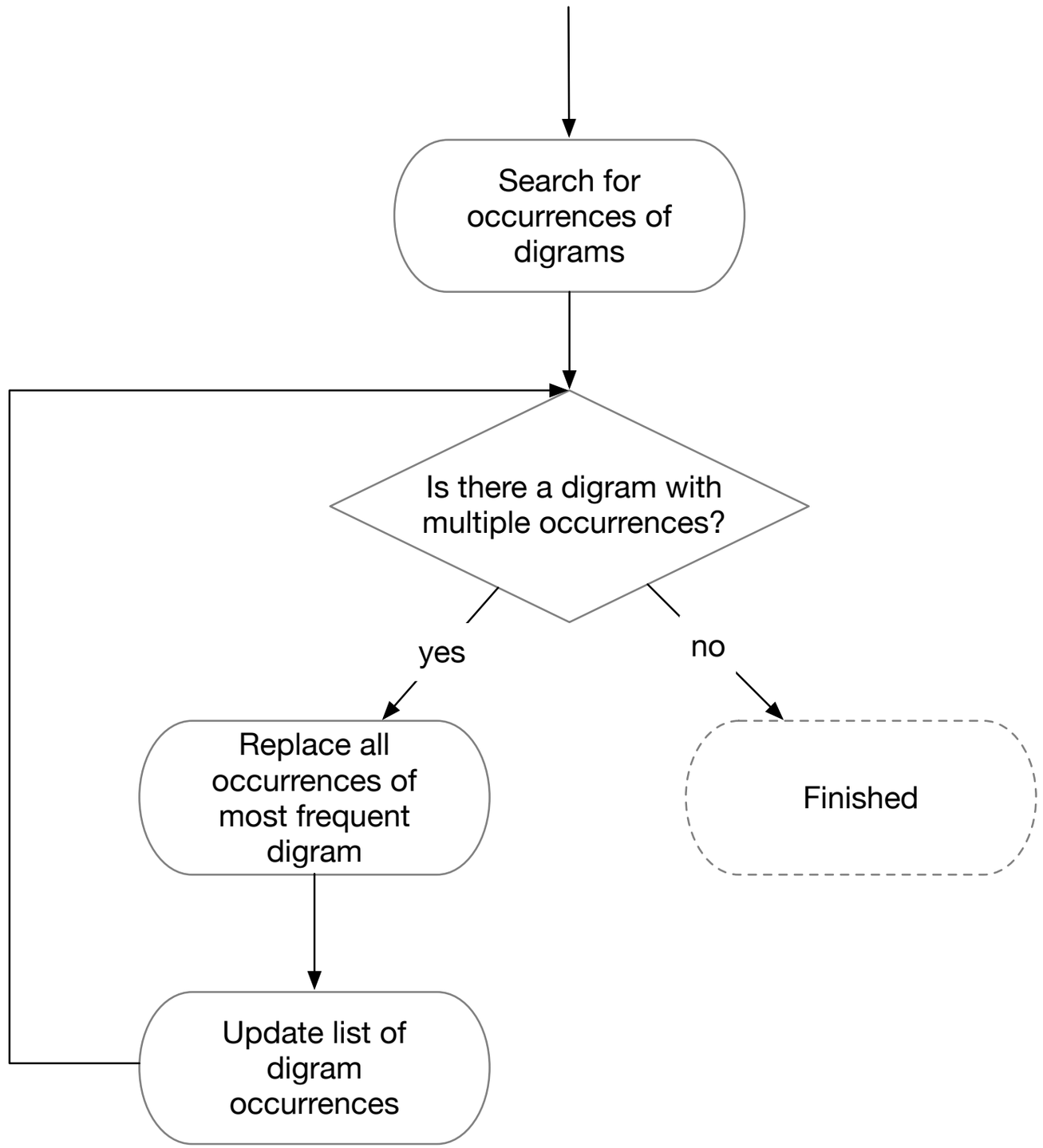
- GRP outperforms HDT in many cases (compression ratio)
- GRP can be improved by applying ontology knowledge (Problem: grammar encoding)
- Much potential for improving dictionary compression
- Dictionary uses much more space than graph

End

Thanks for your attention!
Questions?

Appendix

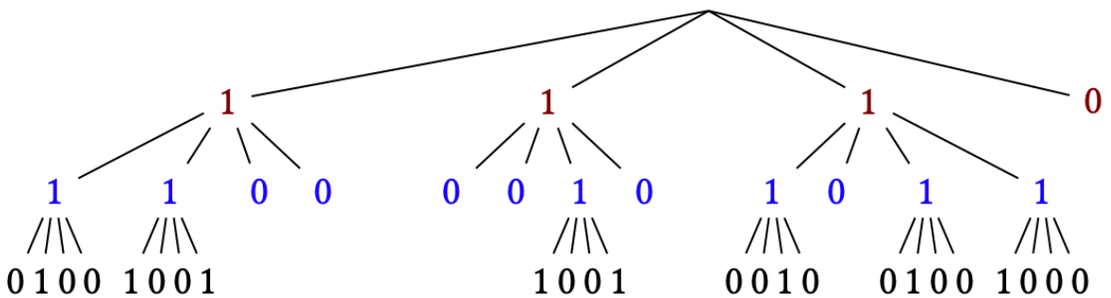
GRP - Algorithm



GRP - Grammar Encoding

	0	1	2	3	4	5	6	
0	0	1	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0
6	0	1	1	0	0	0	0	0
	0	0	0	0	0	0	0	0

(a) Adjacency matrix of a graph.



(b) Conceptual K^2-tree with $k = 2$.

$T_1 = 1110$

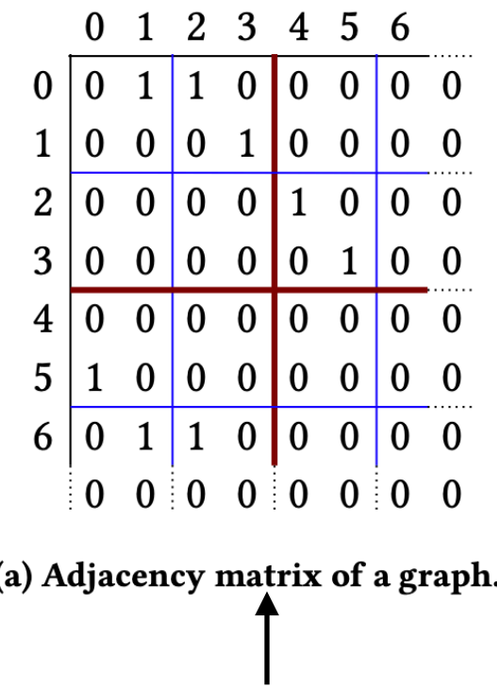
$T_2 = 1100\ 0010\ 1011$

$L = 0100\ 1001\ 1001\ 0010\ 0100\ 1000$

(c) Bit representation of K^2-tree.

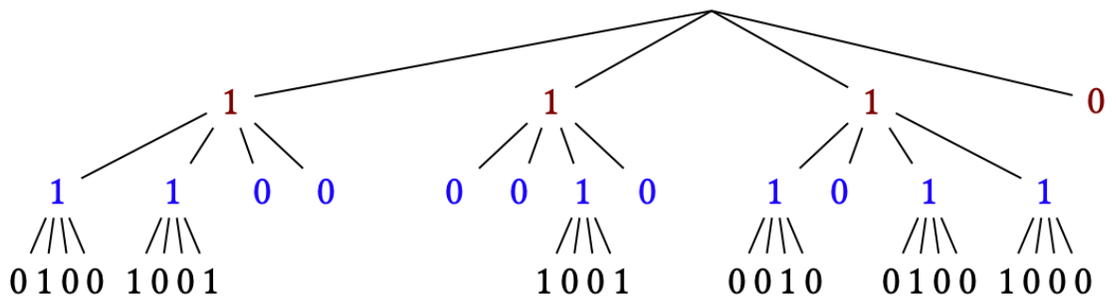
Start rule of the grammar

GRP - Grammar Encoding



	0	1	2	3	4	5	6	
0	0	1	1	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0
6	0	1	1	0	0	0	0	0
	0	0	0	0	0	0	0	0

(a) Adjacency matrix of a graph.



(b) Conceptual K^2-tree with $k = 2$.

$T_1 = 1110$

$T_2 = 1100\ 0010\ 1011$

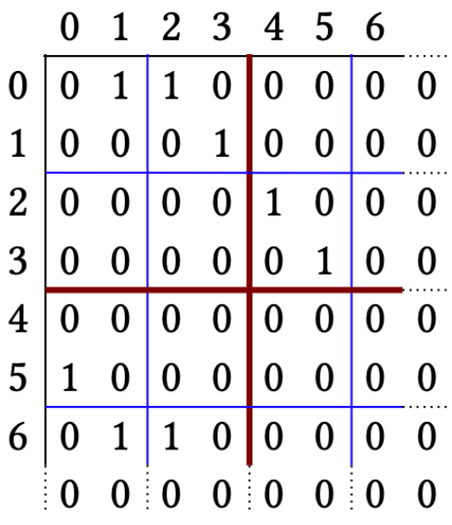
$L = 0100\ 1001\ 1001\ 0010\ 0100\ 1000$

(c) Bit representation of K^2-tree.

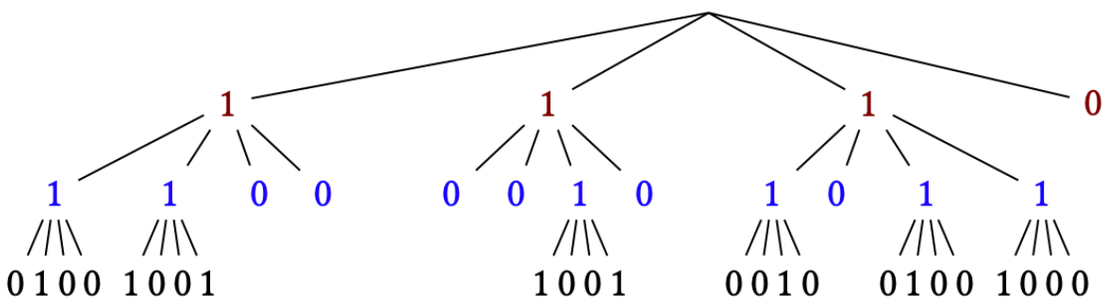
Start rule of the grammar

- Edge labels not stored in matrix => A matrix for each label needed!
- Start rule can contain hyper edges => Order of attached nodes not stored in matrix

GRP - Grammar Encoding



(a) Adjacency matrix of a graph.



(b) Conceptual K^2-tree with $k = 2$.

T_1 = 1110

T_2 = 1100 0010 1011

L = 0100 1001 1001 0010 0100 1000

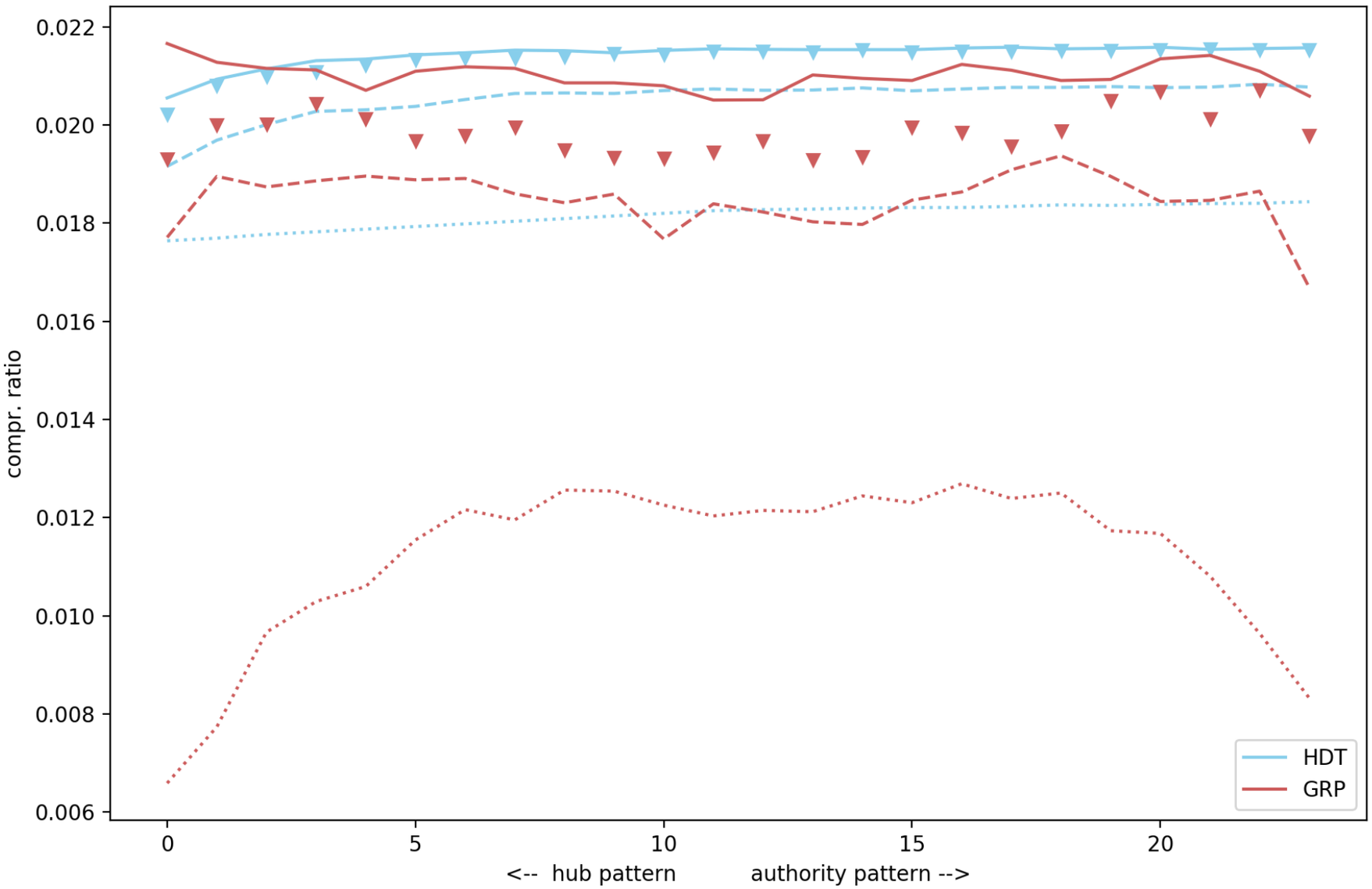
(c) Bit representation of K^2-tree.

Start rule of the grammar

- Edge labels not stored in matrix => A matrix for each label needed!
- Start rule can contain hyper edges => Order of attached nodes not stored in matrix

- Other rules are encoded differently (smaller size)

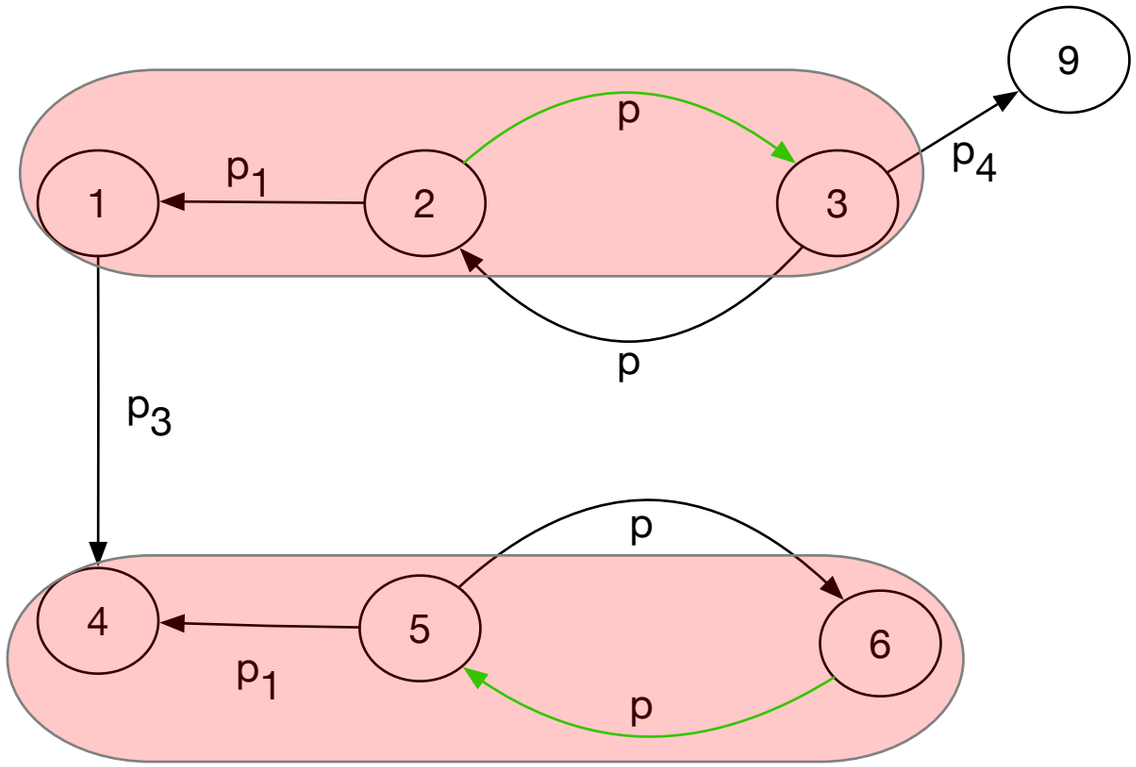
Evaluation results: More distinct properties

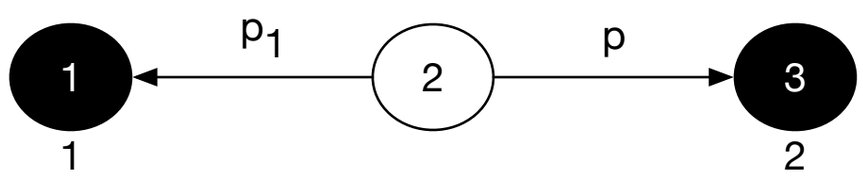


Iteration	Line Style
1	··
2	– –
3	▽▽
4	—

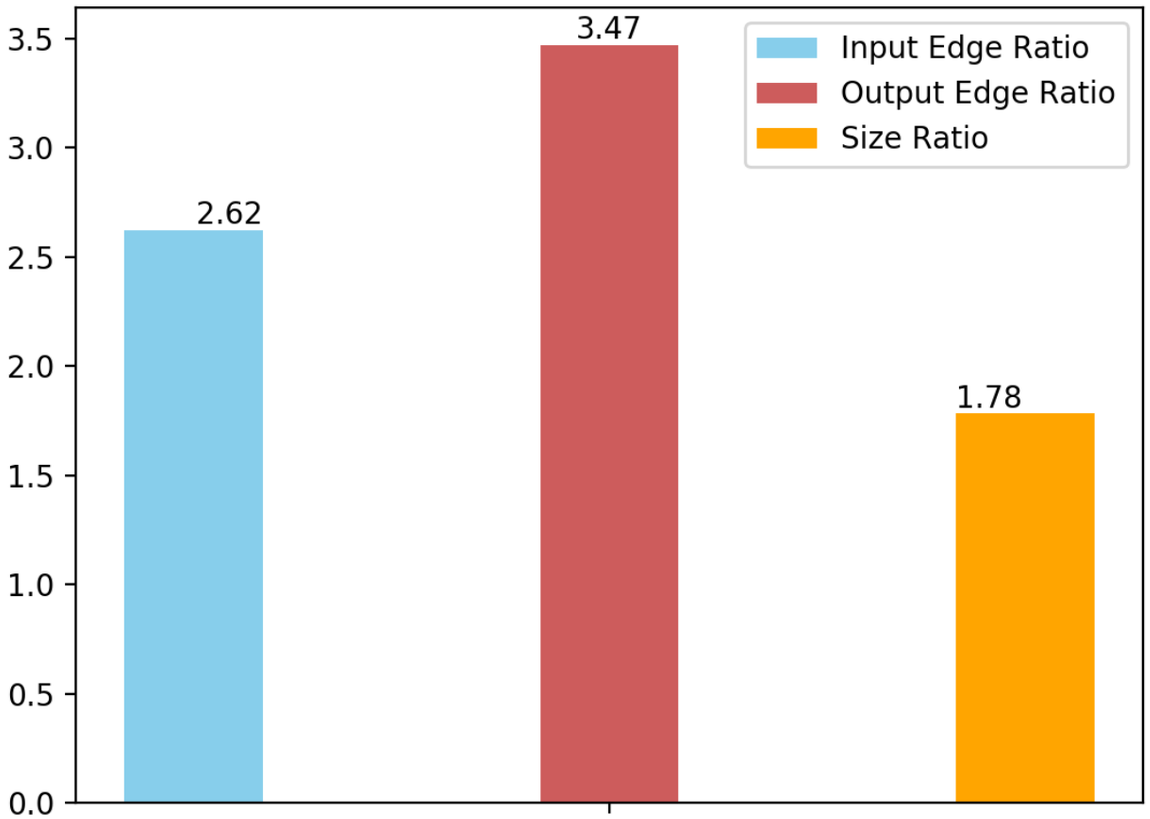
without dictionary

Symmetric properties

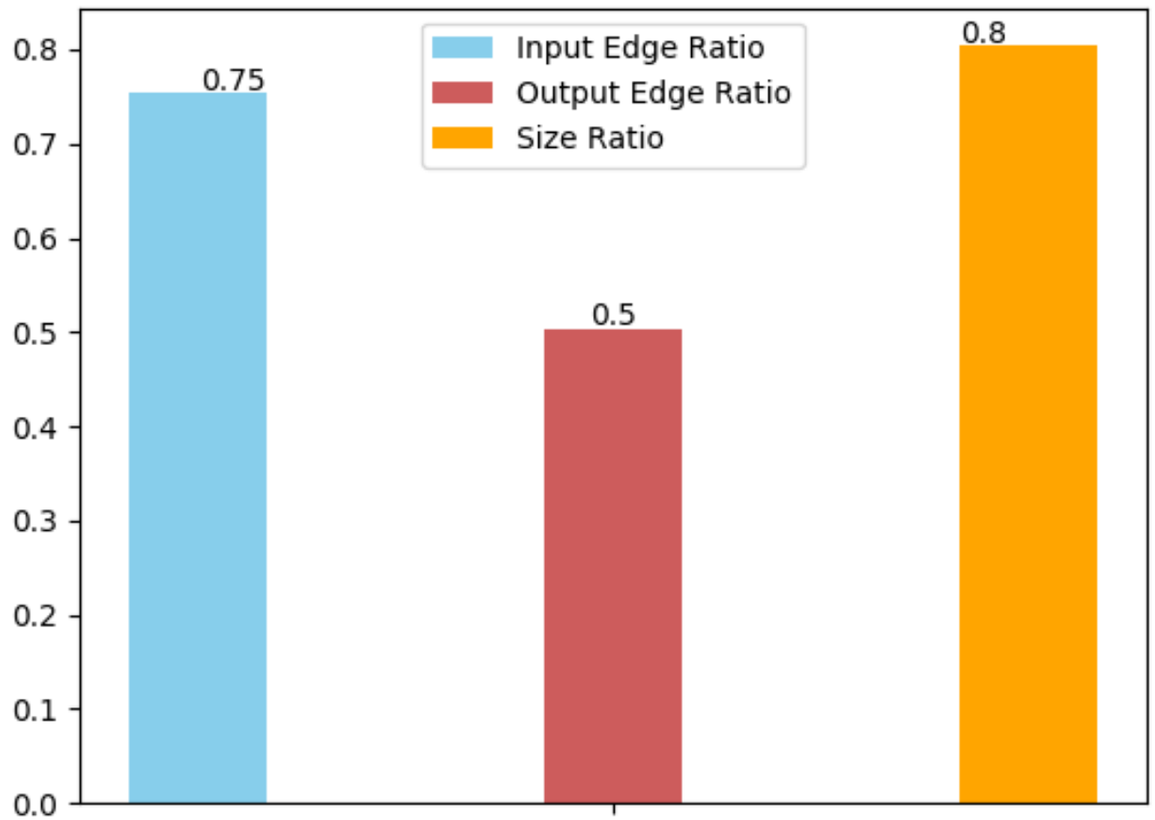




Results: transitive properties

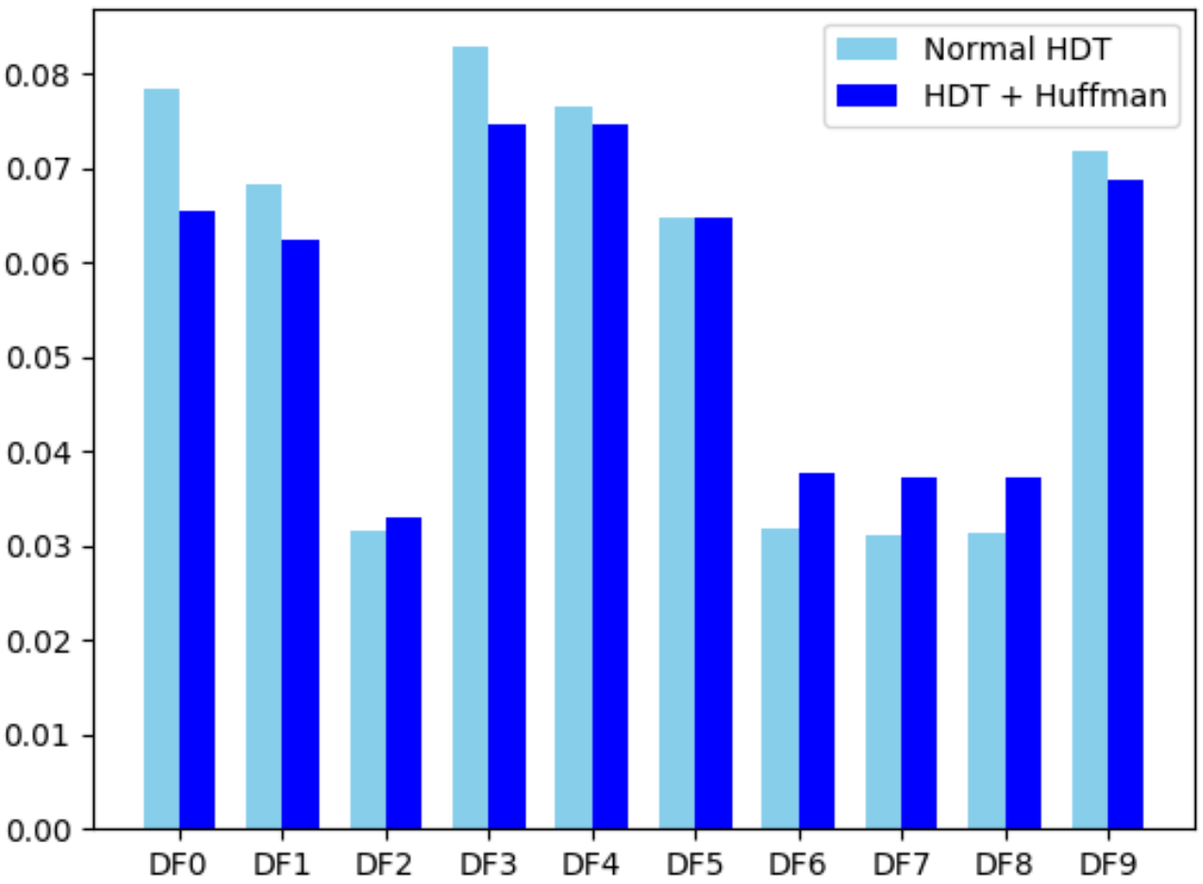


Add direct transitive paths

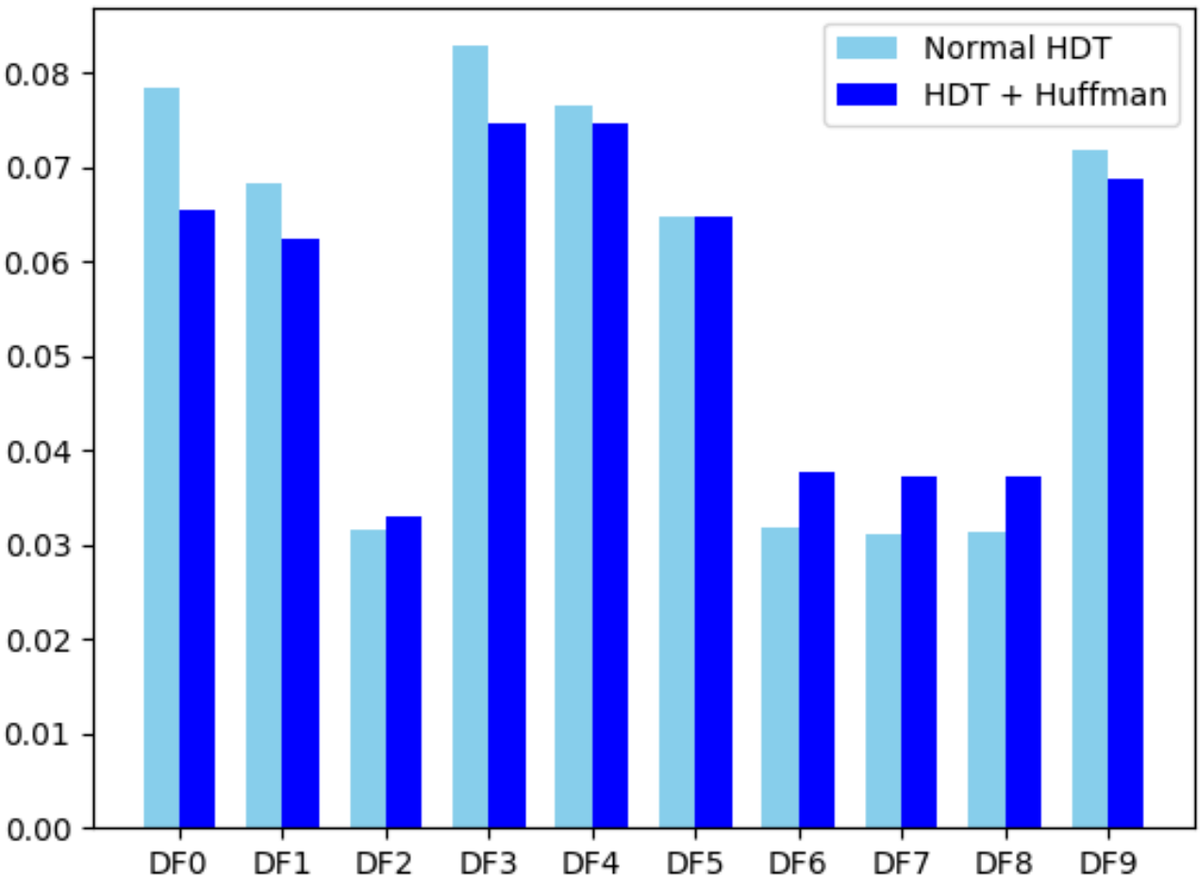


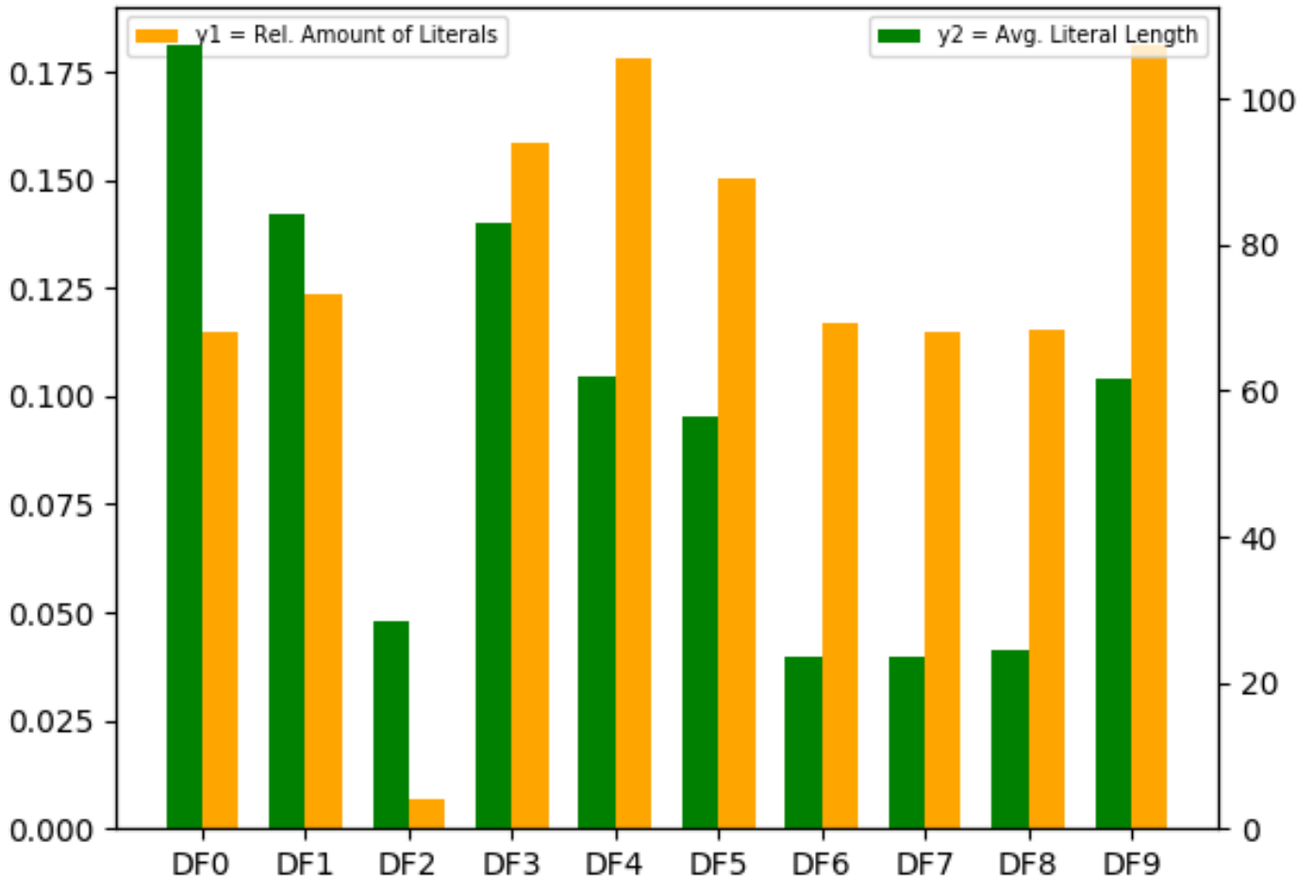
Remove direct transitive paths

Literals: Results for Scholar Data



Literals: Results for Scholar Data





$$\textbf{Relative Literal Amount} = \frac{\#literals}{\#triples}$$

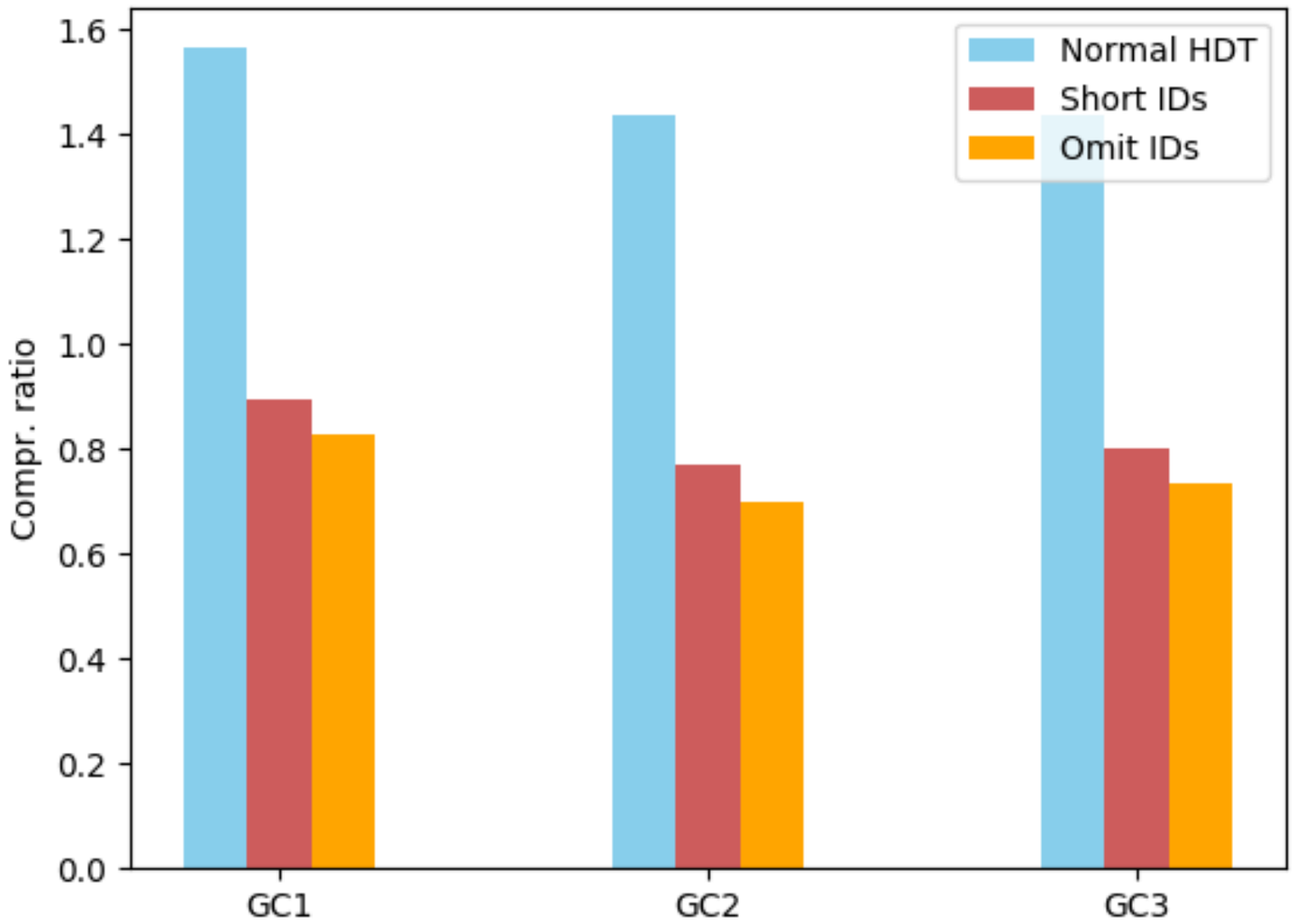
Blank Node IDs

- No semantic meaning
- Only for referencing node across multiple triples
- Dict_{HDT} normally uses arbitrary & long strings as IDs
  => not good for prefix-based compression

Blank Node IDs

- No semantic meaning
- Only for referencing node across multiple triples
- Dict_{HDT} normally uses arbitrary & long strings as IDs
  => not good for prefix-based compression
- Two approaches:
  1. Use shorter IDs (numbers)
  2. Not store IDs in compressed data

Results: Blank Node IDs (Geo Coordinates Data)



Normal HDT produces compression ratio > 1