KOGNAC: Efficient Encoding of Large Knowledge Graphs

Jacopo Urbani,[a] Sourav Dutta,[b] Sairam Gurajada,[b] and Gerhard Weikum[b]
[a]VU University Amsterdam, The Netherlands
[b]Max Planck Institute for Informatics, Germany
jacopo@cs.vu.nl,{sdutta,gurajada,weikum}@mpi-inf.mpg.de

Abstract

Many Web applications require efficient querying of large Knowledge Graphs (KGs). We propose KOGNAC, a dictionary-encoding algorithm designed to improve SPARQL querying with a judicious combination of statistical and semantic techniques. In KOGNAC, frequent terms are detected with a frequency approximation algorithm and encoded to maximise compression. Infrequent terms are semantically grouped into ontological classes and encoded to increase data locality. We evaluated KOGNAC in combination with state-of-the-art RDF engines, and observed that it significantly improves SPARQL querying on KGs with up to 1B edges.

1 Introduction

The advent of natural-language-based queries and entity-centric search has led to the enormous growth and applicability of *Knowledge Graphs* (KG) to model known relationships between entity-pairs. Large KGs have not only been built in academic projects like DBpedia [Bizer *et al.*, 2009],but are also used by leading organizations like Google, Microsoft, etc., to support user-centric Internet services and mission-critical data analytics.

KGs are generally represented using the RDF data model [Klyne and Carroll, 2006], in which the KG corresponds to a finite set of subject-predicate-object (SPO) triples whose terms can be URIs, blank nodes, or literal values [Klyne and Carroll, 2006]. Since many Web applications rely on RDF-style KGs during their processing, efficient and scalable querying on huge KGs with billions of RDF triples have necessitated intelligent KG representation.

In concept, KGs can be managed using a variety of platforms, like RDF engines [Yuan *et al.*, 2013; Gurajada *et al.*, 2014; Neumann and Weikum, 2008], relational stores [Sidirourgos *et al.*, 2008], or graph database systems [Robinson *et al.*, 2015]. In this context, the storage of RDF terms in raw format is both space and process inefficient since these are typically long strings. As such, all existing approaches *encode* the RDF terms typically by mapping them to fix-length integer IDs, with the original strings retrieved only during execution.

Objectives. Modern KGs are typically queried using the W3C SPARQL language [Harris *et al.*, 2013]. Currently, the impact of different ID mappings on advanced SPARQL operations (like query joins, index compression, etc.) is less well studied. Ideally, the encoding of RDF terms into numerical IDs should: i) Consider the *skew* in the term frequencies in the KG, and assign smaller IDs to frequent terms in order to facilitate efficient down-stream compression (by the storage engine). ii) For more advanced query access patterns, particularly for join operations, *data locality* should be increased as much as possible by the encoding. That is, terms that are often accessed together should have close ID assignment in order to further reduce memory and index access [Harbi *et al.*, 2015]. iii) It is often crucial to quickly load billions of triples, for example, when a KG is required as background knowledge for new analytic applications, or for append-only bulk update operations. Thus, the encoding process should support parallelism as much as possible for better scale-up.

Problem Statement. Current RDF engines generally employ four types of encodings: *order or hash-based*, *syntactic*, or based on *coordinates*. Order-based approaches assign consecutive IDs for new incoming triples in the order of appearance. Hash-based procedures use term hashes as IDs. Syntactic encoding assigns IDs to terms based on their lexicographic order. Coordinate-based techniques stored the terms in special data structures and use memory coordinates as IDs.

Interestingly, we observe that none of the existing approaches performs well along all three dimensions of our desiderata. Assigning consecutive identifiers leads to good compression, but does not improve locality for joins. Hash-based encoding allows parallel loading, but has poor locality and is sub-optimal in exploiting skew. The last two disadvantages are also shared by methods which use memory coordinates as IDs. Syntactic encoding provides a compromise along the three objectives, but is not robust enough to handle the cases where term similarity cannot be extracted directly from the syntax. The problem addressed in this paper is to encode RDF terms in large KGs such that all three desiderata, namely better compression, query performance, and loading time, are well satisfied.

Contribution. We present KOGNAC (**K**n**O**wledge **G**raph e**N**oding **A**nd **C**ompression), an efficient algorithm for KG encoding based on a judicious combination of statistical and semantic techniques. Our encoding procedure detects

skewness in term frequency distribution with a approximation streaming technique, and subsequently encodes frequent terms differently in order to facilitate high down-stream compression. To improve *data locality* for join access patterns, KOGNAC computes semantic relatedness between terms by hierarchically grouping them into ontological classes, and mapping terms in the same group to consecutive IDs.

KOGNAC has the advantage that it is independent from RDF application details, since its output is a plain mapping from strings to IDs. To evaluate its efficiency, we integrated it with four RDF systems – RDF-3X [Neumann and Weikum, 2008], TripleBit [Yuan *et al.*, 2013], MonetDB [Sidirourgos *et al.*, 2008], and TriAD [Gurajada *et al.*, 2014] – and observed significant improvements in query performance on metrics like runtime, RAM usage, and disk I/O.

A longer version of this paper, with more details and experiments, is available online at [Urbani *et al.*, 2016].

2 Encoding KGs: State Of The Art

Typically, applications query KGs using SPARQL [Harris *et al.*, 2013] – a W3C declarative language. The core execution of SPARQL queries corresponds to finding all graph isomorphisms between the KG and the graphs defined in the queries.

RDF Encoding. SPARQL engines, e.g., TripleBit [Yuan *et al.*, 2013], TriAD [Gurajada *et al.*, 2014], Virtuoso [Erling and Mikhailov, 2009], etc., use *dictionary encoding* to assign numeric IDs to terms based on their *appearance ordering*, i.e., simply using consecutive or pseudo-random numbers for incoming triples. The 4Store engine [Harris *et al.*, 2009] uses a string-hashing based ID assignment that disregards any possible co-relation among terms. Both approaches do not consider term frequencies leading to sub-optimal encoding with frequent terms possibly assigned to larger IDs. Further, sophisticated partitioning methods in TriAD renders such encoding prohibitively compute expensive [Harbi *et al.*, 2015].

RDF-3X [Neumann and Weikum, 2008], one of the fastest single-machine RDF storage engines, pre-sorts the SPO triples lexicographically and then assigns consecutive integer IDs. A similar approach is also followed by [Urbani *et al.*, 2013], while [Cheng *et al.*, 2014] proposes a combination of appearance order with hashing to improve partitioning. In contrast to our work, these approaches strongly leverage the string similarity heuristics to cluster the elements. These heuristics break when the semantic similarity does not follow the lexicographic ordering. Such dissimilarity occurs frequently via subdomain usage in URIs, or may even be imposed explicitly by political decisions (e.g., Wikidata [Vrandečić and Krötzsch, 2014] uses meaningless strings to avoid an English bias).

Some relational engines (e.g., MonetDB [Sidirourgos *et al.*, 2008]) can optionally use dedicated data structures for the storage of strings (mainly using variants of Tries). In this context, a particular variant of Trie with term prefix overlap was proposed in [Gallego *et al.*, 2013] to capture syntactic similarity. In these cases, the coordinates to the data structure (e.g., memory addresses) are used as numerical IDs. These IDs are typically long, and the induced locality reflects the physical storage of the strings rather than the semantics in the KG. Older versions of MonetDB followed this approach, but it was later abandoned.

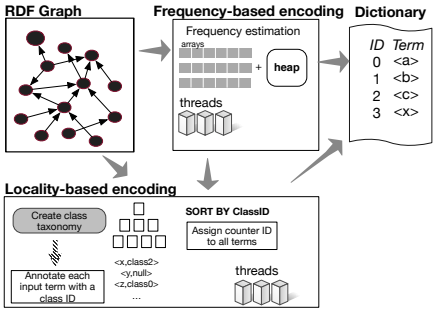


Figure 1: High level overview of KOGNAC.

Semantic Relateness. There is a rich literature on semantic relatedness based on the lexicographic features [Zhang *et al.*, 2013], or on domain-dependant data like Wikipedia [Gabrilovich and Markovitch, 2007], Wordnet [Budanitsky and Hirst, 2006], biomedical data [Pedersen *et al.*, 2007], and spatial [Hecht *et al.*, 2012]. In our case, we cannot make assumptions about the domain of the input and the strings may be completely random, so lexicographic features are not applicable.

In general, semantic relatedness functions cannot be directly applied to our problem of graph encoding. For instance, [Leal, 2013] defines semantic relatedness among two nodes as a function of combining the path length and the number of different paths between two nodes. In our context, it would be too expensive to compute relatedness for many (or even any possible) pairs of nodes. Furthermore, the high sparsity in the graph results in very low relatedness coefficients in almost all cases. [Curé *et al.*, 2015] describes how ontological taxonomies can be exploited to speed up reasoning via intelligent ID encoding. In spirit, this approach is similar to our approach for encoding infrequent terms. However, [Curé *et al.*, 2015] focuses on improving reasoning efficiency rather than the semantic relatedness. Furthermore, [Curé *et al.*, 2015] does not consider data skewness, as we do.

Finally, clustering methods based on the graph structure (e.g., METIS [Karypis and Kumar, 1998], or graph-coloring approach of [Bornea *et al.*, 2013]) are infeasible at our scale [Gurajada *et al.*, 2014], and often require a conversion to an undirected single-label graph disregarding entirely the semantics in the KG. In contrast, the goal of KOGNAC is to leverage precisely this semantics to improve the encoding.

3 The KOGNAC Algorithm

Algorithmic Overview. KOGNAC performs a crucial distinction between (few) *frequent* terms from the (many) remaining *infrequent ones*. For the frequent terms, it applies a frequency-based encoding, which is highly effective in terms of compression due to the skewed distribution of modern KGs [Kotoulas *et al.*, 2010]. For the infrequent ones, it exploits the semantics contained in the KG to encode similar terms together to improve data locality.

Fig. 1 describes at a high level the functioning of KOGNAC.

Let V be the set of terms in a generic RDF graph G. KOGNAC receives G and a threshold value k (used for the top-k frequent elements) as input, and returns a dictionary table $D \subset V \times \mathbb{N}$ that maps every element in V to a unique ID in $\mathbb{N}$.

The dictionary table D is constructed using two different encoding algorithms: *Frequency-based encoding (FBE)*, which encodes only the frequent terms, and *Locality-based encoding (LBE)*, which encodes the infrequent ones. The core computation of FBE corresponds to the execution of an approximated procedure for accurate frequency detection. LBE instead constructs a class taxonomy, groups the terms into these classes, and assigns the IDs accordingly. Both methods support parallelism. FBE is executed before LBE. If D_1 is the output of FBE, and D_2 of LBE, then $D = D_1 \cup D_2$ and $D_1 \cap D_2 = \emptyset$.

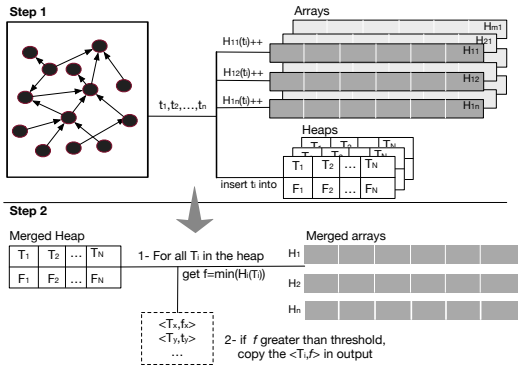


Figure 2: Overview of CM+MG.

4 Frequency-based Encoding (FBE)

The goal of this procedure is to detect the top-k frequent terms and assign them incremental IDs starting from the most to the least frequent term. For our purpose, an exact calculation of the frequencies is not required, and even though it can be easily computed for small KGs, it would be unnecessarily expensive for very large KGs. Sampling provides the reference technique for a fast approximation [Urbani *et al.*, 2013]. Unfortunately, an excessive sampling might lead to false positives and negatives, and increasing the sample size for tolerable error rates might still be too expensive.

To obtain a better approximation, we investigated the applicability of hash-based sketch algorithms [Karp *et al.*, 2003; Charikar *et al.*, 2002]. Sketch algorithms have successfully been deployed in other domains to identify distinct items in streams [Karp *et al.*, 2003], but never to our problem domain.

Sketch Algorithms. We experimented with three state-of-the-art sketch algorithms: *Count-Sketch* [Charikar *et al.*, 2002], *Misra-Gries* [Misra and Gries, 1982], and *Count-Min* [Cormode and Muthukrishnan, 2005]. Count-Sketch, a single-scan algorithm, requires a heap with quadratic space in error tolerability. After many experiments, we concluded that updating such large heap was too expensive for our inputs.

Misra-Gries is similar to Count-Sketch, with the difference that it uses a smaller heap and reports the terms that are at least *k-frequent*, i.e., having a frequency $> \frac{n}{k}$, where n denotes the total number of term occurrences. This method approximates the relative frequencies (i.e., frequency after a term is inserted in the heap). We found that the relative frequencies were not accurate enough to allow a precise ordering of terms, as the count depends on the appearance ordering.

Count-Min uses $n > 1$ hash counter arrays and n hash functions to count the frequencies. In contrast to the previous two methods, it always requires two input scans: First to count the frequencies, and then to extract the actual frequent terms. Count-Min does not use heaps and this makes it in principle faster than the other two. However, for large inputs the cost of second input scan offsets this gain.

Our proposal: CM+MG. Count-Min provides a good estimate of term frequencies, but cannot identify the top-k elements within a single pass. Misra-Gries detects the top-k elements but does not report good frequencies. The disadvantages of the two are complementary. We thus propose a hybrid approach, which we call *CountMin+MisraGries (CM+MG)*, that intelligently combines elements of the two.

An overview of its functioning is reported in Fig. 2. As input it receives the input KG, k, a hash family H with n hash functions, m parallel threads, and a threshold k of popular elements. In our implementation, we selected as default values three hash functions for H, while m is the number of physical cores, and the value k is requested from the user.

CM+MG is executed in two steps: In the first step, CM+MG creates $m * n$ counter arrays and m Misra-Gries heaps of size k. The KG is split into m subsets, and fed to the m threads. Each thread calculates n hash codes for each term occurrence in its partition. The n hash-codes are modulo-ed the array size and the corresponding indices in the arrays are incremented by 1. The terms are also inserted into the heaps.

In the second step, the m heaps are merged in a single heap. Also the arrays are summed into n final arrays. As threshold value for the frequent terms, we select the top-k value in the first array. The algorithm now scans all elements in the merged heap. Instead of using the relative frequency as estimate, CM+MG queries the arrays using the term's hashcodes, and uses the minimum of the returned values. If this value is greater than the threshold, the term is marked as frequent.

Effectiveness of FBE. Our approach works well because KGs are skewed. In order to better characterize the gain we can obtain with our approach, we present a short theoretical analysis on the space efficiency of FBE.

Assume $T = \{t_1, t_2, \ldots, t_n\}$ to be n distinct RDF terms, with term t_i having a frequency of f_i in the input KG. In the worst case, an assignment criterion which is independent from the term frequencies (e.g., an order-based one) will produce an assignment which is close to a fixed-length encoding; that is where all terms will be assigned to IDs of length $\lceil \log_2 n \rceil$ bits. In this case, the total space required to store an encoded KG in the database would be

$$S_{fix} = \sum_{i=1}^{n} f_i \lceil \log_2 n \rceil = F \lceil \log_2 n \rceil \quad [\text{with } F = \sum_{i=1}^{n} f_i] \qquad (1)$$

In KOGNAC, the terms are divided into blocks depending on their frequencies. Here, block $i \in \{1 \ldots b\}$ contains the top 2^i elements which are not in any previous group. Hence, there exists $b = \lceil \log_2 n \rceil$ non-empty blocks for n distinct

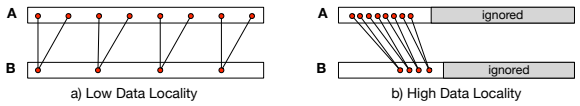


Figure 3: Effect of data locality during SPARQL join.

items in T. An item in block i is encoded using i bits. Since the assignment is not *prefix-free*, in order to properly decode the IDs we need to append to each term some extra data to discriminate it from the different values. This extra data must take at least $\lceil \log_2 b \rceil$ bits as this is the minimum space necessary to identify each of the b blocks. Hence, the total space required for decoding an item in block i takes $(i + \lceil \log_2 b \rceil)$ bits. Assuming, f_j^i to denote the frequency of the j^{th} item in block i, the total encoding space required for KG is,

$$S_{kog} = \lceil \log_2 b \rceil \sum_{i=1}^{b} \sum_{j=1}^{2^i} f_j^i + \sum_{i=1}^{b} i \sum_{j=1}^{2^i} f_j^i \qquad (2)$$

Since modern KGs have a skewed term distribution [Kotoulas *et al.*, 2010], we now assume that the item frequencies is drawn from a Zipfian distribution[1] with parameter $s \geq 2$, such that the frequency of the k^{th} frequent term $f_k \approx \frac{F}{s^k}$. If we apply this distribution in Eq. (2), we obtain

$$S_{kog} = F\lceil \log_2 b \rceil + \sum_{i=1}^{b} \frac{iF}{s^{\sum_{k=1}^{i-1} 2^k}} \sum_{j=1}^{2^i} \frac{1}{s^j}$$

By algebraic manipulations, we have

$$S_{kog} = F\lceil \log_2 b \rceil + \sum_{i=1}^{b} \frac{iF}{s^{2^i-2}} . \frac{1}{s-1} \qquad \text{[for large } i\text{]}$$

$$\approx F\lceil \log_2 b \rceil + F \sum_{i=1}^{b} \frac{i}{s^{2^i}} \approx F\left(\lceil \log_2 b \rceil + \frac{1}{s^2} \right) \qquad (3)$$

If we compare Eq. 3 with Eq. 2, then we see that with KOGNAC we can achieve nearly an exponential theoretical decrease (i.e., $log\ log\ n$ vs. $log\ n$) in the total encoding space required to store the KG. This is the scale of potential improvement that our encoding can offer to the current frequency-independent encoding algorithms.

5 Locality-based Encoding (LBE)

In the long tail of the frequency distribution, a frequency-based encoding no longer pays off. Each infrequent term appears only a few times and this reduces the negative impact of assigning large IDs to them. Moreover, the increased ID space provides a much larger number of disposable IDs: for instance, after the most frequent 2^{24}th ID, all the following $2^{32} - 2^{24}$ IDs will take the same number of bytes.

Data Locality. With LBE, we propose an encoding that is designed to improve *data locality* during the execution of SPARQL queries. Data locality plays a significant role to reduce the cost of index access for advanced operations like relational joins. Consider, as example, Fig. 3, which shows a join between two generic relations A and B. It is common that these relations are indexed (i.e., sorted) to enable merge joins [Neumann and Weikum, 2008]. If the index locations of the join terms are spread around the entire relations, as shown in Fig. 3a, then the join algorithm must process large parts of the indexes. On the contrary, if the join succeeds using only sub-portions of the index, then the join algorithm can save significant computation by ignoring large chunks of the indexes (Fig. 3b).

Encoding and Data-Locality. Since SPARQL engines work (mostly) directly on the encoded data, the elements on which the join operates are precisely the IDs which we should assign during encoding. Therefore, an assignment of close IDs will significantly improve data locality.

Unfortunately, it is not possible to encode the terms so that they are *always* next to each other. We can make one assignment only, and SPARQL queries could request joins on any subset of the relations. Still, we can leverage a heuristics that is surprisingly effective: SPARQL joins tend to materialize between terms which are semantically related.

Following this heuristic, we propose to cluster the terms into the ontological classes they are connected to via the isA relation, and assign consecutive IDs to the members of each cluster. Our approach has several advantages: (i) isA is a common relation in KGs and is domain-independent; (ii) new isA edges can be inferred using other ontological information, e.g., definitions of the domain/range of properties; (iii) classes can be organized in a taxonomy using the *rdfs:subClassOf* relation of the RDF Schema [Brickley and Guha, 2014]. The taxonomy can help us to further sepa-

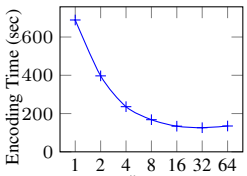
$S := \{\}$; $D_{infreq} := \{\}$; $cID :=$ max ID in D_{freq};
$MAX :=$ constant with number higher than any class ID in T;
for *every triple <s,p,o> in the KG* **do**
  Add to S three pairs: $\langle s, MAX\rangle$,$\langle p, MAX\rangle$,$\langle o, MAX\rangle$;
  If p = 'type', then add $\langle s, id(o,T)\rangle$ to S;
  If p has domain c, then add $\langle s, id(c,T)\rangle$ to S;
  If p has range c, then add $\langle o, id(c,T)\rangle$ to S;
end
Remove from S all pairs $\langle t, c\rangle$ where t is in D_{freq};
for *all pairs* $\langle t_1, c_1\rangle$ *and* $\langle t_2, c_2\rangle$ *in* S **do**
  if $t_1 = t_2$ **then**
    Remove $\langle t_1, c_1\rangle$ from S if $c_1 > c_2$ or remove $\langle t_2, c_2\rangle$ otherwise;
  end
end
while S *is not empty* **do**
  Take out from S one pair $\langle t_1, c_1\rangle$ s.t.
  $\nexists \langle t_2, c_2\rangle \in S : c_2 < c_1 \vee c_1 = c_2 \wedge t_2 < t_1$;
  Increment cID by 1;
  Add to dictionary D_{infreq} the assignment $\langle t_1, cID\rangle$;
end

Algorithm 1: Locality-aware Encoding: *Input:* a KG, the taxonomy T, and the frequent dictionary D_{freq} generated by FBE. *Output:* The infrequent dictionary D_{infreq}.

[1]This distribution is used for heavy-tailed characteristics observed in natural language sources used for KG construction.

Dataset	Encoding Runtime (sec)			
	KOGNAC			RDF3X
	FBE	*LBE*	*Total*	
LUBM1K	429	752	1181	1136
LUBM8K	3863	867	12538	11637
DBPedia	5071	10504	15575	12805
LDBC	612	1127	1739	1696

(a)



(b)

Figure 4: (a) Comparison of KOGNAC vs RDF-3X dictionary encoding times, (b) KOGNAC multi-threading performance analysis on LUBM1K. Experiments on machine M1.

rate instances of subclasses from instances of siblings of the parents (e.g., *Students* should be closer to *Professors* than to *Robots* because the first two are both subclasses of *Persons*).

Algorithm Overview. Our locality-based encoding works as follows: First, we must create the taxonomy of classes. To this end, we extract all triples that define the subclass relation between classes, or mention classes (these share *rdfs:subClassOf* as predicate, and are the objects of triples with the predicates *isA*, *rdfs:domain*, and *rdfs:range*). We create a graph where the classes are vertices and the edges are defined by the *subclassOf* triples. We add the standard class *rdfs:Class* (the class of all classes [Brickley and Guha, 2014]), and add one edge from each vertex to it, to ensure that there are no disconnected components. We remove possible loops in the graph by extracting the tree with the maximum number of edges rooted in *rdfs:Class* using the well-known Tarjan's algorithm [Tarjan, 1977].

We now assign in post-order an incremental class ID to each class in the tree. We indicate with $id(c, T)$ the class ID assigned to the class c contained in the taxonomy T.

After the taxonomy is built, we are ready to encode the terms. This procedure is outlined in Alg. 1. First, we annotate each term with a class ID it is an instance of. The annotation might come from an explicit relation (`isA`) or an implicit one (`domain` and `range`). If a term cannot be associated to any class, we give it a dummy ID (MAX). Then, for each term we maintain only the annotation with the class which has the smallest ID. Finally, we order all annotations first by class ID, and then (lexicographically) by term. We use the same counter used in FBE and assign incremental IDs to the terms with the order defined in the sorted list. In this way, the assignment first considers the semantic type of the term, and then its syntax. Notice that terms which are not mapped to any class (mainly labels), will be encoded only syntactically.

A large part of the computation of LBE can be parallelized. The task of assigning the terms to the smallest class IDs does not require thread synchronization because at this point the taxonomy is a read-only data structure. Therefore, the task can be trivially parallelized using standard input range partitioning and parallel merge sorts. The final assignment is performed sequentially due to the usage of a single counter.

6 Evaluation

We implemented `KOGNAC` in a C++ prototype. We tested it in combination with four SPARQL engines: RDF-3X, TripleBit, TriAD, and MonetDB. We chose them because they represent the state-of-the-art of different type of SPARQL engines: native/centralized (RDF-3X, TripleBit), native/distributed (TriAD), and RDBMS/centralized (MonetDB). These engines also perform different encodings: RDF-3X performs syntactic encoding, TripleBit and TriAD performs an order-based encoding, MonetDB can be loaded with arbitrary encodings (we used a syntactic encoding).

We used two types of machines: *M1*, a dual 8-core 2.4 GHz Intel CPU, 64 GB RAM, and two disks of 4 TB in RAID-0; and *M2*, a 16 quad-core Intel Xeon CPUs of 2.4GHz with 48GB of RAM. As input, we used three RDF graphs in NT format: LUBM [Guo *et al.*, 2005] – a popular benchmark tool, LDBC [Angles *et al.*, 2014], another, more recent benchmark designed for advanced SPARQL 1.1 workloads, and DBPedia [Bizer *et al.*, 2009], one of the most popular KGs. We created two LUBM datasets: *LUBM1K* (133M triples, 33M terms), and *LUBM8K* (1B triples, 263M terms). We created a LDBC dataset with 168M triples and 177M terms. The DBPedia dataset contains about 1B triples and 232M terms.

Due to space reasons, for LUBM, we used only five adaptations of benchmark queries which were selected as representative in [Yuan *et al.*, 2013]. For DBPedia, we slightly changed five example queries that are reported in the project's website. For LDBC, we use the official queries in the SPB usecase [Angles *et al.*, 2014]. These queries require SPARQL 1.1 operators which were unsupported by any of our engines. We implemented some of the missing operators in RDF-3X so that we could launch 7 of the 12 SPB queries. Fig. 5 reports the LUBM and DBPedia queries in compressed form. The LDBC queries are freely available at benchmark website www.ldbcouncil.org. Here we use abbreviations to refer to them (e.g. query Q3 is the third query of the official benchmark). The `KOGNAC` code is available at `https://github.com/jrbn/kognac`.

Encoding runtime. As baseline, we selected the syntactic encoding algorithm performed by RDF-3X. We found that syntactic encoding performs better than the others, and RDF-3X implements a highly optimized loading procedure.

First, we measured the (sequential) encoding runtimes of all four datasets and report them in Fig. 4(a). We notice that our approach is slightly slower than the one of RDF-3X. We expected such difference, since we perform a much more complex operation than a simple syntactic encoding. In the worst case, `KOGNAC` is about 20% slower. Considering that loading is a one-time operation whose cost gets amortized over time, we deem it as an acceptable cost, especially in view of the benefit we get at query time. In Fig. 4(b), we measured the `KOGNAC` runtime doubling each time the number of threads from 1 to 64. We see that the runtime steadily decreases until it stabilizes after 8 threads. The runtime left after this point is the one necessary to sequentially assign the IDs to the list of terms and write the dictionary to disk.

In a series of experiments (not shown in this paper), we compared the performance of CM+MG against sampling at 5% (the most popular technique) and Count-Min (the fastest one). We varied the k threshold, and measured the runtime and accuracy of the approximation. We found that $k = 50$

	Q.	# Results	Runtime (sec)		Max RAM (MB)		Disk I/O (MB)	
			KOG	*R3X*	*KOG*	*R3X*	*KOG*	*R3X*
LUBM1K	L1	10	0.22	0.31	4	4	14	19
	L2	10	0.04	0.17	5	6	16	27
	L3	1	88.53	90.83	708	854	53	69
	L4	2528	92.21	98.41	724	883	548	367
	L5	44190	7.45	15.79	1,261	1,588	1,102	2,132
LUBM8K	L1	10	0.09	0.27	4	4	15	21
	L2	10	0.02	0.64	5	7	18	32
	L3	1	700.58	716.55	5,335	6,537	309	486
	L4	2528	717.65	744.10	5,320	6,536	321	811
	L5	351919	75.90	174.16	9,739	12,400	8,832	16,928
DBPedia	D1	449	0.99	3.32	8	15	55	52
	D2	600	0.23	3.17	6	8	29	61
	D3	270	1.60	2.96	6	11	59	49
	D4	68	0.72	1.34	6	6	45	49
	D5	1643	5.05	26.79	29	60	330	263
LDBC	Q2	36	44.59	45.58	1,320	2,053	241	279
	Q3	178	126.48	132.55	524	574	588	624
	Q6	3819127	60.17	71.32	2,157	5,198	1,268	3,953
	Q7	98	5.85	6.76	549	3,663	625	3,675
	Q8	1018	1,847	4,915	2,867	5,934	1,804	3,949
	Q10	14	420.88	4,577	714	3,662	729	3,676
	Q11	114	24.51	89.21	170	204	201	237

Table 1: Query runtime, Max RAM usage, and disk I/O with KOGNAC and RDF-3X encodings on one M1 machine.

was a good threshold value because KGs typically contain only few very frequent terms. Therefore, all experiments in this section should be intended with $k = 50$. In general, CM+MG was the fastest algorithm with k up to 500. In the best case, it was twice as fast as Count-Min (the second best). In the worst case, it was 10% slower. With higher ks, CM+MG became slower because of the heap and Count-Min returns the best runtimes. In terms of accuracy, all three managed to identify the very first top k. However, as we increased k, all methods started to fail: Sampling quickly lost accuracy, Count-Min produced large overestimates (due to hash collisions in the arrays), while CM+MG produced underestimates (due to the limited heap).

SPARQL Query performance. Tab. 1 reports on the execution of SPARQL queries with the mappings produced by KOGNAC and by RDF-3X's syntactic encoding. The table reports cold query runtimes, the maximum amount of RAM used by the system, and the disk I/O. From the table, we see that KOGNAC leads to a significant improvement over all three metrics. All query runtimes are faster, with improvements of up to ten times. The system always uses less main memory and in all but four cases it reads less data from disk.

We tested KOGNAC also with the other three systems, which use a traditional order-based encoding. For conciseness, we report in Tab. 2 only the runtimes of the LUBM queries as they are representatives of the general behaviour. We observe that also here KOGNAC produced better runtimes. For TriAD, there was an improvement in all but two cases, where the runtime was unchanged. For TripleBit, one query failed due to bugs in the system, while another produced a slightly worse runtime. In the remaining cases KOGNAC encodings were beneficial. Finally, with MonetDB we observed an improvement in all cases. In essence, our results show that an intelligent assignment, like the one produced by KOGNAC, has a significant impact on the processing of the KG. Given the encoding runtime, this improvement comes at little cost.

Q.	TriAD		TripleBit		MonetDB	
	KOGNAC	*Standard*	*KOGNAC*	*Standard*	*KOGNAC*	*Syntactic*
L1	0.001	0.001	0.056	0.149	0.820	2.4
L2	0.002	0.002	0.094	n/a	0.943	1.2
L3	0.106	0.631	1.672	1.567	11.1	15.2
L4	2.684	3.090	5.626	6.549	9.5	21.1
L5	2.558	3.067	5.082	6.438	4.1	8.2

Table 2: Impact of KOGNAC on example SPARQL queries using one M2 machine.

LUBM Queries.
@prefix r: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix u: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
L1. { ?x u:subOrganizationOf < http://www.Department0. University0.edu> . ?x r:type u:ResearchGroup .}
L2. { ?x u:worksFor <http://www.Department0.University0.edu> . ?x r:type u:FullProfessor . ?x u:name ?y1 . ?x u:emailAddress ?y2 . ?x u:telephone ?y3 . }
L3. { ?y r:type ub:University . ?x u:memberOf ?z . ?z u:subOrgOf ?y . ?z r:type u:Department . ?x u:undergradDegreeFrom ?y . ?x r:type u:UndergradStudent.}
L4. { ?y r:type u:University . ?z u:subOrgOf ?y . ?z r:type u:Department . ?x u:memberOf ?z . ?x r:type u:GraduateStudent . ?x u:undergradDegreeFrom ?y .}
L5. { ?y r:type u:FullProfessor . ?y u:teacherOf ?z . ?z r:type u:Course . ?x u:advisor ?y . ?x u:takesCourse ?z . }
DBPedia Queries.
@prefix foaf: <http://xmlns.com/foaf/0.1/>, purl: <http://purl.org/dc/terms/>, db: <http://dbpedia.org/resource/>, dbo: <http://dbpedia.org/ontology/>, rs: <http://www.w3.org/2000/01/rdf-schema#>
D1. { ?car purl:subject db:Category:Luxury_vehicles . ?car foaf:name ?name . ?car dbo:manufacturer ?man . ?man foaf:name ?manufacturer }
D2. { ?film purl:subject db:Category:French_films }
D3. { ?g purl:subject db:Category:First-person_shooters . ?g foaf:name ?t }
D4. { ?p dbo:birthPlace db:Berlin . ?p dbo:birthDate ?b . ?p purl:subject db:Category:German_musicians . ?p foaf:name ?n . ?p rs:comment ?d}
D5. { ?per dbo:birthPlace db:Berlin . ?per dbo:birthDate ?birth . ?per foaf:name ?name . ?per dbo:deathDate ?death .}

Figure 5: LUBM and DBPedia queries.

7 Conclusions

We proposed KOGNAC, an algorithm for efficient encoding of RDF terms in large Knowledge Graphs. KOGNAC adopts a combination of estimated frequency-based encoding (for frequent terms) and semantic clustering (for infrequent terms) to encode the graph efficiently and improve data locality in a scalable way. We evaluated the performance of KOGNAC by integrating it into multiple state-of-the-art RDF engines and relational stores. We observed significant improvements regarding query runtimes, a reduction in memory usage, and disk I/O. These results were achieved without altering the architecture or functioning of the RDF engine, but only by rearranging the encodings in an intelligent way.

We identified several directions for future work. More combinations of encoding, or other clustering criteria for terms that cannot be mapped to the taxonomy might further improve the performance. Moreover, it is interesting to study how well our FBE and LBE encodings can deal with updates, or whether they can improve other tasks than SPARQL. For instance, methods for knowledge completion with embeddings might also benefit from our encodings.

To the best of our knowledge, our work is the first that seeks an improvement of KG processing via intelligent dictionary encoding. KOGNAC represents a first step in this direction to further improve the processing of emerging KGs.

Acknowledgments. This work was partially funded by the NWO VENI project 639.021.335.

References

[Angles *et al.*, 2014] R. Angles, P. Boncz, J. Larriba-Pey, I. Fundulaki, T. Neumann, O. Erling, P. Neubauer, N. Martinez-Bazan, V. Kotsev, and I. Toma. The linked data benchmark council: A graph and RDF industry benchmarking effort. *ACM SIGMOD Record*, 43(1):27–31, 2014.

[Bizer *et al.*, 2009] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia-A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.

[Bornea *et al.*, 2013] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee. Building an Efficient RDF Store over a Relational Database. In *SIGMOD*, pages 121–132, 2013.

[Brickley and Guha, 2014] D. Brickley and Ramanathan V. Guha. RDF Vocabulary Description Language 1.1: RDF schema, 2014. W3C Recommended.

[Budanitsky and Hirst, 2006] A. Budanitsky and G. Hirst. Evaluating WordNet-based measures of lexical semantic relatedness. *Comput. Linguist.*, 32(1):13–47, 2006.

[Charikar *et al.*, 2002] M. Charikar, K. Chen, and M. Farach-Colton. Finding Frequent Items in Data Streams. In *ICALP*, pages 693–703, 2002.

[Cheng *et al.*, 2014] L. Cheng, A. Malik, S. Kotoulas, T. Ward, and G. Theodoropoulos. Efficient parallel dictionary encoding for RDF data. In *Proc. 17th International Workshop on the Web and Databases (WebDB'14)*, 2014.

[Cormode and Muthukrishnan, 2005] G. Cormode and S. Muthukrishnan. An improved data stream summary: the Count-Min Sketch and its applications. *J. of Algorithms*, 55(1):58–75, 2005.

[Curé *et al.*, 2015] O. Curé, H. Naacke, T. Randriamalala, and B. Amann. LiteMat: A Scalable, Cost-efficient Inference Encoding Scheme for Large RDF Graphs. In *Big Data*, pages 1823–1830, 2015.

[Erling and Mikhailov, 2009] O. Erling and I. Mikhailov. Virtuoso: RDF Support in a Native RDBMS. In *Semantic Web Information Management*, pages 501–519, 2009.

[Gabrilovich and Markovitch, 2007] E. Gabrilovich and S. Markovitch. Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis. In *IJCAI*, volume 7, pages 1606–1611, 2007.

[Gallego *et al.*, 2013] M. A. Gallego, O. Corcho, J. D. Fernández, M. A. Martínez-Prieto, and M. C. Suárez-Figueroa. *CAEPIA*, chapter Compressing Semantic Metadata for Efficient Multimedia Retrieval, pages 12–21. Springer, 2013.

[Guo *et al.*, 2005] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182, 2005.

[Gurajada *et al.*, 2014] S. Gurajada, S. Seufert, I. Miliaraki, and M. Theobald. TriAD: A Distributed Shared-nothing RDF Engine based on Asynchronous Message Passing. In *SIGMOD*, pages 289–300, 2014.

[Harbi *et al.*, 2015] R. Harbi, I. Abdelaziz, P. Kalnis, and N. Mamoulis. Evaluating SPARQL queries on massive RDF datasets. *PVLDB*, 8(12):1848–1851, 2015.

[Harris *et al.*, 2009] S. Harris, N. Lamb, and N. Shadbolt. 4store: The Design and Implementation of a Clustered RDF Store. In *SSWS*, pages 94–109, 2009.

[Harris *et al.*, 2013] S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language. *W3C Recommendation*, 21, 2013.

[Hecht *et al.*, 2012] B. Hecht, S. H. Carton, M. Quaderi, J. Schöning, M. Raubal, D. Gergle, and D. Downey. Explanatory Semantic Relatedness and Explicit Spatialization for Exploratory Search. In *SIGIR*, pages 415–424, 2012.

[Karp *et al.*, 2003] R. Karp, S. Shenker, and C. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28(1):51–55, 2003.

[Karypis and Kumar, 1998] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Compututing*, 20(1):359–392, 1998.

[Klyne and Carroll, 2006] G. Klyne and J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, 2006. W3C.

[Kotoulas *et al.*, 2010] S. Kotoulas, E. Oren, and F. Van Harmelen. Mind the Data Skew: Distributed Inferencing by Speeddating in Elastic Regions. In *WWW*, pages 531–540, 2010.

[Leal, 2013] J.P Leal. Using proximity to compute semantic relatedness in RDF graphs. *Computer Science and Information Systems*, 10(4):1727–1746, 2013.

[Misra and Gries, 1982] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982.

[Neumann and Weikum, 2008] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *PVLDB*, 1(1):647–659, 2008.

[Pedersen *et al.*, 2007] T. Pedersen, Serguei V. S. Pakhomov, S. Patwardhan, and Christopher G. Chute. Measures of semantic similarity and relatedness in the Biomedical domain. *Journal of Biomedical Informatics*, 40(3):288–299, 2007.

[Robinson *et al.*, 2015] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases: New Opportunities for Connected Data*. O'Reilly, 2015.

[Sidirourgos *et al.*, 2008] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-Store Support for RDF Data Management: Not all swans are white. *PVLDB*, 1(2):1553–1563, 2008.

[Tarjan, 1977] R. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[Urbani *et al.*, 2013] J. Urbani, J. Maassen, N. Drost, F. Seinstra, and H. Bal. Scalable RDF data compression with MapReduce. *Concurrency and Computation: Practice and Experience*, 25(1):24–39, 2013.

[Urbani *et al.*, 2016] J. Urbani, S. Dutta, S. Gurajada, and G. Weikum. KOGNAC: Efficient Encoding of Large Knowledge Graphs (Tech. Report), 2016. http://arxiv.org/abs/1604.04795.

[Vrandečić and Krötzsch, 2014] D. Vrandečić and M. Krötzsch. Wikidata: A free collaborative knowledge base. *Commun. ACM*, 57(10), 2014.

[Yuan *et al.*, 2013] P. Yuan, P. Liu, B. Wu, H. Jin, W. Zhang, and L. Liu. TripleBit: A fast and compact system for large scale RDF data. *PVLDB*, 6(7):517–528, 2013.

[Zhang *et al.*, 2013] Z. Zhang, A. Gentile, and F. Ciravegna. Recent advances in methods of lexical semantic relatedness – A survey. *Natural Language Engineering*, 19(04):411–479, 2013.