Information Sciences 509 (2020) 87–103

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

HAPE: A programmable big knowledge graph platform

Ruqian LU[a,b,c,d,*], Chaoqun FEI[a,b,e], Chuanqing WANG[c,d,e], Shunfeng GAO[a,b,f], Han QIU[g], Songmao ZHANG[c,d], Cungen CAO[a,b]

[a] ICT, Chinese Academy of Sciences, Beijing 100190, China
[b] Key Lab of IIP, Chinese Academy of Sciences, Beijing 100190, China
[c] AMSS, Chinese Academy of Sciences, Beijing 100190, China
[d] Key Lab of MADIS, Chinese Academy of Sciences, Beijing 100190, China
[e] University of Chinese Academy of Sciences, Beijing 100190, China
[f] Jiangsu University of Science and Technology, Zhenjiang 210003, China
[g] Telecom Paris, Paris 75013, France

ARTICLE INFO

Article history:
Received 8 May 2019
Revised 18 August 2019
Accepted 22 August 2019
Available online 23 August 2019

Keywords:
Big knowledge
Big knowledge system
Big knowledge graph
Knowledge graph browser
Knowledge graph operating system
Knowledge scripting language
Big knowledge security

ABSTRACT

Heaven Ape (HAPE) is an integrated big knowledge graph platform supporting the construction, management, and operation of large to massive scale knowledge graphs. Its current version described in this paper is a prototype, which consists of three parts: a big knowledge graph knowledge base, a knowledge graph browser on the client side, and a knowledge graph operating system on the server side. The platform is programmed in two high level scripting languages: JavaScript for programming the client side functions and Python for the server side functions. For making the programming more suitable for big knowledge processing and more friendly to knowledge programmers, we have developed two versions of knowledge scripting languages, namely K-script-c and K-script-s, for performing very high level knowledge programming of client resp. server side functions. HAPE borrows ideas from some well-known knowledge graph processing techniques and also invents some new ones as our creation. As an experiment, we transformed a major part of the DBpedia knowledge base and reconstructed it as a big knowledge graph. It works well in some application tests and provides acceptable efficiency.

1. Introduction

Big data might have been one of the hottest technical terms since the beginning of the 21st century. It was Michael Cox and David Ellsworth who have coined the term “big data” in their paper published at a conference on visualization in 1997 and called it “the interesting challenge for computer system” [5]. Only one year later, in 1998, the first paper with the term “big data” in the title was published [44]. This time the growth of the Internet was taken as evidence. Three years later, in 2001, Doug Laney proposed the 3V (Volume, Velocity, Variety) characteristics of big data in a published short note [25]. Further developments include 4V [15] that adds Veracity to 3V and subsequently 5V [45] that adds Value to 4V. Besides, Xindong Wu et al. proposed the HACE (Heterogeneous, Autonomy, Complex, Evolving) model for characterizing big data in 2016 [49].

* Corresponding author at: ICT, Chinese Academy of Sciences, Beijing 100190, China.
E-mail addresses: rqlu@math.ac.cn (R. LU), feichaoqun@ict.ac.cn (C. FEI), wangchuanqing15@mails.ucas.ac.cn (C. WANG), gaoshunfeng@stu.just.edu.cn (S. GAO), han.qiu@telecom-paris.fr (H. QIU), smzhang@math.ac.cn (S. ZHANG), cgcao@ict.ac.cn (C. CAO).

https://doi.org/10.1016/j.ins.2019.08.051
0020-0255/© 2019 Elsevier Inc. All rights reserved.

Almost all big data researches recognized the significance of inventing new strategies and techniques for mining knowledge from big data. It was pointed out in [33] that knowledge acquisition has experienced a drastic change in the big data era. The main source of knowledge acquisition is no more the domain experts as in the time of Feigenbaum [11]. Nowadays new knowledge comes mainly from big data mining. Furthermore, the knowledge mined from big data should be different from that mined from usual data. One wants it to be a brand new kind of knowledge, both quantitatively and qualitatively [48]. Finally, in the second decade of this century, people have found the term "big knowledge (BK)" for it [1,20,27,33,37]. However, although quite a few authors have emphasized the significance of BK research, it was difficult to find a published paper or talk discussing the essential properties of BK, e.g., something like the 3V, 4V, 5V, and HACE models for big data. It was only in 2018 that the first paper in this line [29] was published where BK, Big Knowledge System (BKS) and Big Knowledge Engineering (BKE) were discussed in depth together and the 10MC (Massive Characteristics) model was proposed. It is the basis and start point of our discussion in this paper, where we will report the implementation of the first BKS according to the 10MC principles in form of a big knowledge graph (BKG) platform. Let us first recall some basic definitions from [29].

Definition 1. BK is a massive set of structured knowledge elements, where a knowledge element may be a concept, an entity, a datum, a rule or any other computer operable information element. The most popular properties of BK are the five MC characteristics [29] described roughly as follows. The estimated quantities of each characteristic are obtained after investigating some practical BK examples. In particular, the concrete quantities for some real-world BKG systems are given later in Section 2.

- MC1: Massive knowledge elements, represented as concepts, entities, rules, facts, etc., where the core components are concepts. Without concepts there will be no knowledge.
- MC2: Massive well-connectedness between knowledge elements. Without connectedness, there will be no reasoning at all. In particular, well-connectedness should be enhanced to well-structuring of knowledge.
- MC3: Massive clean data resources which are raw or semi-raw data to be upgraded to knowledge.
- MC4: Massive cases which are facts and evidence of the applicability of the BK.
- MC5: Massive confidence which means that the reliability of knowledge is high.

The lower bounds of BK characteristics are estimated according to some typical BK examples, and they are estimated as 0.1M for the number of concepts, 1M for relations, 1M for clean data resources, 10M for a number of visits (cases), and finally 80% for a measure of knowledge confidence (reliability). Note that MC1, MC2, and MC5 are necessary for BK whilst MC1-5 make the characteristics complete.

Definition 2. BKS is some BK supported by the following sixth criterion.

- MC6: Massive capabilities, where capabilities refer to knowledge processing techniques including algorithms, tools, and routines.

Definition 3. Advanced BKS fulfills four additional criteria as follows.

- MC7: Massive Cumulativeness. It means that the knowledge content undergoes a continuous and uninterrupted growth and renewing process.
- MC8: Massive Concerns. It means that the knowledge is not limited to any special set of knowledge domains. It collects any kind of knowledge. Knowledge often exists in forms of different truth values, preciseness, representation forms, etc. For keeping and benefiting them in a most suitable way, we introduce the idea of possible worlds, which is well-known in modal logic semantics [31]. For a BKS, any knowledge element may belong to different possible worlds at the same time and have different values and representations there.
- MC9: Massive Consistency. It means that the whole BK can be covered by a set of possible worlds, where each possible world is logically consistent and at least one possible world satisfies MC1, MC2, and MC5.
- MC10: Massive Completeness. It requires that there is a massive subset of possible worlds fulfilling the (knowledge) completeness criteria.

Due to the limited space, in this paper, we only check the first five MCs, namely only the characteristics for BK *per se*.

2. Big knowledge graph – testbed of big knowledge systems

In the second decade of this century, a new form of knowledge representation and organization has appeared in computer science and artificial intelligence community. This is the knowledge graph (KG). It became well known to the public since Google purchased Mendawei including its knowledge system Freebase in 2010, and published its own Knowledge Graph later in 2012. Since then many KGs have been developed and published. To mention a few, DBpedia, YAGO, NELL, and OpenCyc are all large, real-world KGs covering commonsense knowledge domains.

Since BK exists in a variety of different forms, the standards for estimating MCs also vary in different situations. In the case of KGs, the lower bounds of BKG characteristics are shown in Table 1, where the MC1 and MC2's lower bounds for

Table 1
The basic statistics of some well-known KGs [29].

KG	Concept (100K)	Entity (10M)	Fact (1M/1B)	Precision (90%)
Freebase (E)	2000 (2013) [18]	44M (2013) [18]	3.1B (2017) [9]	99%(2017) [9]
Google Knowledge Graph (E)	N/A	570M (2012) [2]	70B (2016) [22]	88%(2012) [38]
Knowledge Vault (E)	N/A	45M (2014) [8]	1.6B (2014) [8]	90%(2014, ca. 270M estimated) [8]
Probase (C)	5.4M (2016) [42]	N/A	20,757,545 (2016) [42]	92.8%(2012) [47]
DBpedia	N/A	4.58M (2014) [7]	3B (2014) [7]	99%(2017) [9]
OpenKN (E)	10,000 (2016) [23]	20M (2014) [23]	2.2B (2014) [23]	N/A
Xlore	856,146 (2013) [41]	7,854,301 (2013) [41]	N/A	90.48%(F1-score) (2013) [41]
YAGO2 (C)	350,000 [21]	10M (2015) [21]	120M (2015) [21]	95%(2011) [21]
YAGO3	N/A	45M (2015) [30]	540M (2015) [30]	99%(2017) [9]
Wikidata	N/A	15M (2014) [40]	43M (2014) [40]	99%(2017) [9]
OpenCyc	6000 (2008) [6]	N/A	60,000 (2008) [6]	nearly 100%
OpenCyc2	47,000 (2009) [6]	N/A	306,000 (2009) [6]	nearly 100%
OpenCyc3 [C]	177,000 (2010) [6]	N/A	1,505,000 (2010) [6]	nearly 100%
OpenCyc4 [C]	239,000 (2012) [6]	N/A	2,093,000 (2012) [6]	nearly 100%

concept-based BKG are estimated as 100 K resp. 1M, while they are 10M resp. 1B for entity-based BKG. The lower bound of MC5 is estimated as 90% by random testing.

Definition 4. A KG whose knowledge content and supporting facilities meet the conditions of BK defined in the head line of Table 1 is called a BKG.

Based on Table 1, we can see that Freebase, Google Knowledge Graph, Knowledge Vault, and OpenKN are entity-based BKGs whereas Probase, YAGO2, OpenCyc3 and OpenCyc 4 are eligible to be called concept-based BKGs. Note that although DBpedia, which is entity-based (only 4.58 M entities), does not reach the lower bound of MC1, it still fits the lower bound of MC2 (3B). In addition, DBpedia has much better accessibility than many other KGs. Therefore we decided to take DBpedia as the knowledge source in our experiment.

3. Previous works on knowledge graph platforms

Metaphactory [19] is a KG management platform which was primarily designed towards large enterprises and organizations. It provides three functionalities including KG data management, end-user oriented interaction and rapid application development according to the diverse needs of different categories of KG users within the organization. Those functionalities are very useful to KG management. GNOSS [16] provides plentiful KG management and metadata management functions including ontology management, thesaurus and taxonomy management, linked data management, semantic content management, presentation layer personalization, multilingual management, etc. The above two platforms have shown what a KG management system should have, and give some hints to the design of our Heaven Ape (HAPE) platform.

In enterprises, KG and KG management also have received deep attention. Dan Woods [46] discusses the necessity of Enterprise Knowledge Graph (EKG) in business. He describes the power of KG and how KG works in a corporation. Michael Galkin et al. [13,14] consider that EKG can provide a new data integration paradigm by combining large-scale data processing with Semantic Web technologies. They even think EKG is the first step towards the next generation of Enterprise Information Systems. They describe the features necessary for implementing EKG. Moreover, they give an assessment framework and use it to compare 10 of the most prominent EKG solutions. The assessment dimensions of this framework involve KG curation, exploration, search, security, etc.

There are other works related to KG curation and management. James P et al. [32] introduce a reusable framework for developing KGs including knowledge curation, interaction, and inference. Based on this framework, KG builders can update the KG on their demand. Zhanfang Zhao et al. [50] summarize the architectures of the well-known KGs such as DBpedia, YAGO, etc. and divide KG construction approaches into two groups namely top-down and bottom-up ones. They both affirm the importance of knowledge extraction, knowledge construction in the KG development. Furthermore, they also think knowledge management can be beneficial to continuous quality improvement, which coincides with our concern on knowledge management.

Despite a variety of concerns on KG and KG management, there are still serious insufficiencies which make the research, management, and service of BKG unsatisfactory in many aspects. The knowledge base of each BKG consists of a huge quantity of Resource Description Framework (RDF) triplets. There lack higher level structures to organize group operations on knowledge. Almost all knowledge services nowadays are only triplets oriented. This provides only limited services to users with few question patterns. It lacks a user-friendly interactive knowledge discovery mechanism for browsing the BKG knowledge in an in-depth way. It cannot provide a dynamic survey of some specific knowledge according to the user request. It cannot report to the user the major information changes of the BKG since his/her last visit.

The research of BKG will produce lots of brand new techniques to be studied by the AI and computer science professionals. We expect that the above insufficiencies of current BKG will be resolved step by step in the coming future. The goal of our HAPE project is to design and implement a general-purpose BKG platform which is open to the public and is

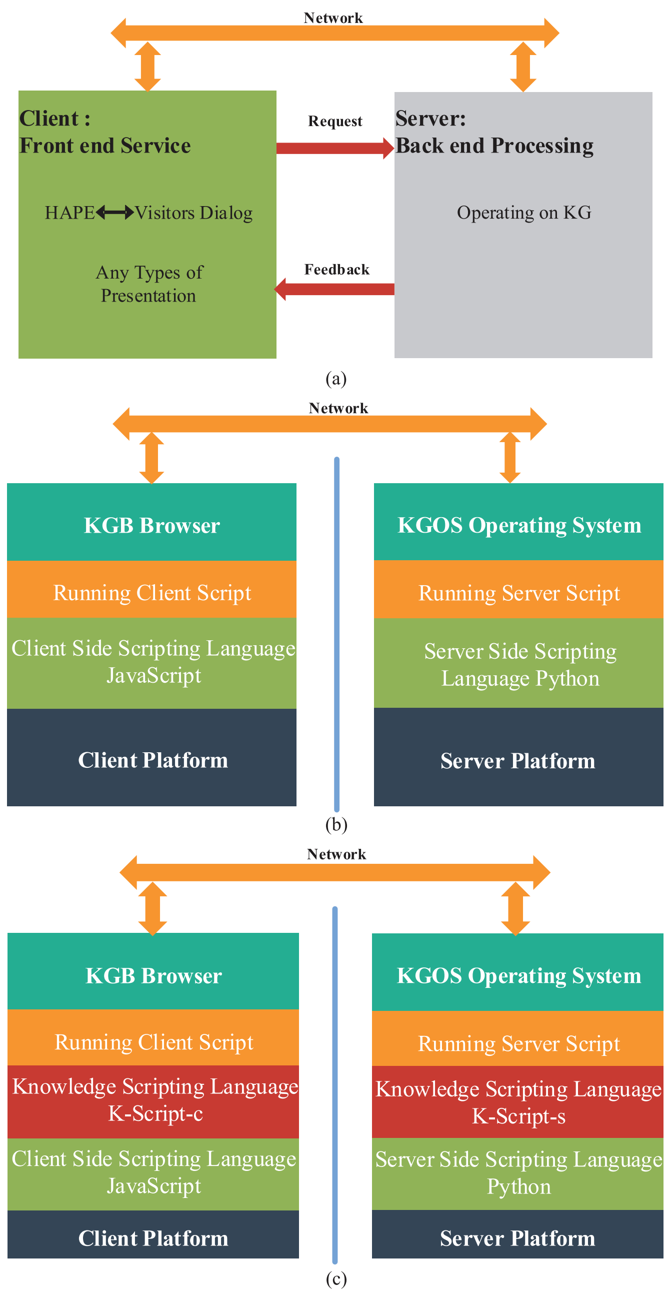

Fig. 1. HAPE system architecture: (a) the client server structure of HAPE; (b) high level script programming of HAPE; (c) high level knowledge programming of HAPE.

suitable for doing various experiments on BK in the form of BKG. For that purpose, we extract a large module of DBpedia and transform it into HAPE architecture as shown in Section 9. In doing so, some new techniques are developed to support our experiments.

4. HAPE architecture – knowledge graph as big knowledge system

The architecture of HAPE is a platform consisting of three parts: the client side, which provides various kinds of services to the visitors and interacts directly with them; the server side, which provides all kinds of knowledge management and processing, which directly affect HAPE's third part – the KG's knowledge base, as shown in Fig. 1 (a). More precisely, HAPE's client side is a browser operating on KG's knowledge base, while its server side is HAPE's operating system (OS) operating on the whole HAPE platform. Both of the browser and the OS are running according to their own scripts produced by scripting

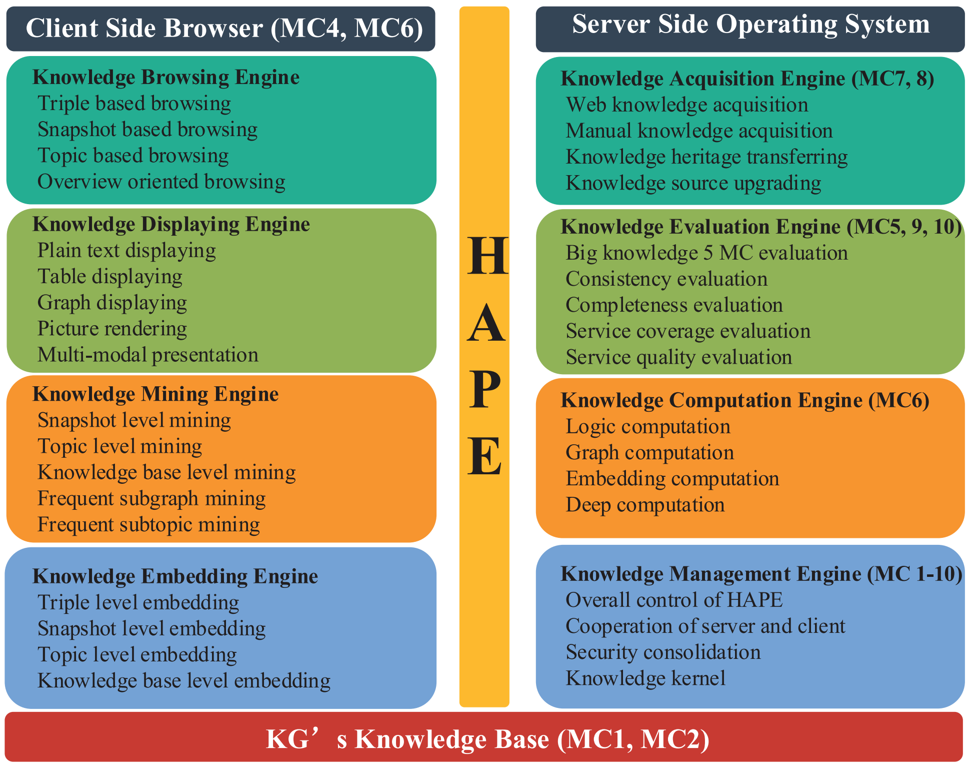
Client Side Browser (MC4, MC6)
Server Side Operating System
Knowledge Browsing Engine
Triple based browsing
Snapshot based browsing
Topic based browsing
Overview oriented browsing
Knowledge Acquisition Engine (MC7, 8)
Web knowledge acquisition
Manual knowledge acquisition
Knowledge heritage transferring
Knowledge source upgrading
Knowledge Displaying Engine
Plain text displaying
Table displaying
Graph displaying
Picture rendering
Multi-modal presentation
Knowledge Evaluation Engine (MC5, 9, 10)
Big knowledge 5 MC evaluation
Consistency evaluation
Completeness evaluation
Service coverage evaluation
Service quality evaluation
HAPE
Knowledge Mining Engine
Snapshot level mining
Topic level mining
Knowledge base level mining
Frequent subgraph mining
Frequent subtopic mining
Knowledge Computation Engine (MC6)
Logic computation
Graph computation
Embedding computation
Deep computation
Knowledge Embedding Engine
Triple level embedding
Snapshot level embedding
Topic level embedding
Knowledge base level embedding
Knowledge Management Engine (MC 1-10)
Overall control of HAPE
Cooperation of server and client
Security consolidation
Knowledge kernel
KG’s Knowledge Base (MC1, MC2)

Fig. 2. Functional components of HAPE.

language programs. The languages used are JavaScript and Python for client and server side programming, respectively, as shown in Fig. 1 (b). There are three advantages to adopting high level scripting languages. Firstly, HAPE’s runtime functions can be programmed in high level languages; secondly, these runtime functions can be transported to other KG platforms easily; and thirdly, managers or even visitors can modify the program codes to get new functions.

Since JavaScript and Python are not designed specifically for knowledge programming, we have designed and implemented extensions of these two languages, called k-script-c and k-script-s, respectively. Both of them contain program idioms designed specifically for knowledge programming. Translators are provided for transforming these idioms into JavaScript and Python statements. This is illustrated by Fig. 1 (c) and the details are given in Section 8.

As for the concrete functions of the client and server side, each of them consists of four major engines, where the browsing engine, displaying engine, mining engine and embedding engine belong to the client side, whereas the acquisition engine, evaluation engine, computation engine, and management engine belong to the server side. These functions are shown in Fig. 2 and we will discuss them in subsequent sections. Note that in Fig. 2 each module is attached with one or more MC markings. This is because we have constructed HAPE platform according to the 10 MC principles of BK, and the markings show the particular MCs that each module of HAPE’s architecture is relevant to.

5. Knowledge structure of HAPE

In this section, we come to the knowledge base of HAPE. Usually the knowledge base of a KG consists of a large set of knowledge elements that are concepts, entities, etc. Though these elements are connected by facts or rules, they are wired into a complicated network which is very difficult to be divided into meaningful modules. This makes knowledge operations such as inference, search, mining, prediction, etc. very difficult. This says that the knowledge elements of KG are badly structured. The situation becomes even worse when we are facing BKG, where the number of knowledge elements may reach as high as several billions. A rational approach out of this bottleneck is to make the immense knowledge base a well-structured organization of knowledge modules. These modules should have different types/patterns according to their respective ways of structuring. Only predefined operations can be applied to corresponding modules and their patterns. Semantically each module pattern is considered as an abstract data type which usually contains variables. Modules whose all variables are instantiated are called module instances.

HAPE maintains five different kinds of knowledge structures, described as follows, and most of them are newly introduced.

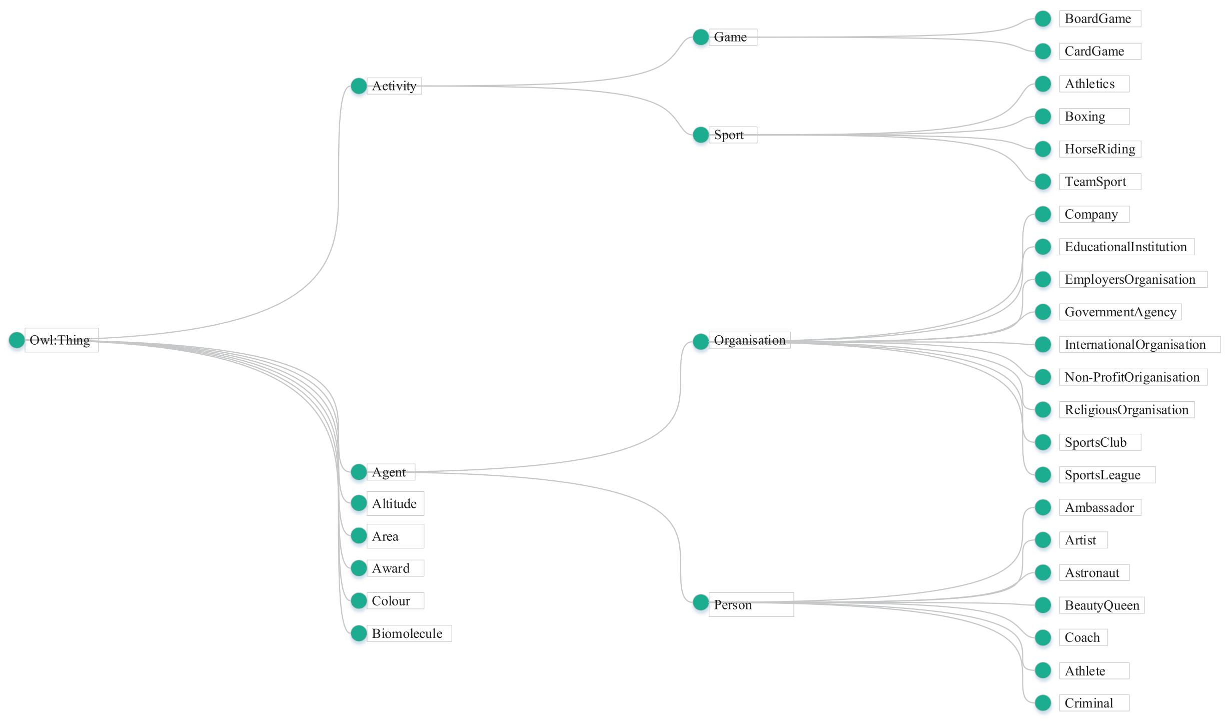

Fig. 3. A taxonomy of DBpedia concepts.

5.1. Triplet pattern- the smallest module pattern

In this paper, a triplet always refers to a RDF triplet (h-term, relation, t-term). A triplet pattern is a triplet where at least one of the h-term or t-term is a variable. It is also called a v-triplet in this paper.

Example: Given a set of triplets (Zhang_Yimou, directorOf, Red_Sorghum), (William_Wyler, directorOf, Rome_holiday), (Cary_Fukunaga, directorOf, Jane_Eyre), we can find that (X, directorOf, Y) is a triplet pattern, representing “who directs what”.

5.2. Concept taxonomy

Syntactically, a taxonomy is a hierarchy of concepts where the leaves may be concepts or entities. Semantically, the concepts of BKG are grouped in inheritance classes which form a partial ordering. BKG extracts concepts from knowledge resources and organizes them in taxonomies to ease the reasoning. Fig. 3 shows that a domain related terminology is organized in taxonomies.

$$Syntax : Taxonomy ::= Concept|Entity|Empty|Concept : [Taxonomy]_{2-n} \tag{1}$$

5.3. Snapshot

The main source of knowledge for DBpedia comes from Wikipedia articles, where an infobox is often attached playing the role of a short summary of the articles’ content. Though infoboxes are useful and significant, they do not provide readers with the knowledge wanted in all cases. Firstly, there are domains about which the Wiki-articles do not have infoboxes, e.g. in the domain of science (e.g. Wiki-Mathematics) and culture (e.g. Wiki-Impressionism). Secondly, infoboxes occur solely in part of Wiki-articles. Since only very important and classical subjects (e.g. the Second World War) are worth to be written down as a Wiki-article, the overwhelming majority of subjects of the same sort have not been considered. So we think why not make infobox an independent form of knowledge representation, such that a much larger quantity of explicit or implicit knowledge pieces can be mined from a huge RDF knowledge base? This idea has led us to the definition of a new knowledge representation form called snapshots, specified as follows:

Definition 5. Each RDF knowledge base G can be considered as a network whose nodes (edges) are the entities (relations) of triplets of G, and no two nodes correspond to a single entity while different edges can correspond to the same relation. A connecting path between two nodes has a length equal to the number of relations on that path. Distance between two nodes is minimal length among all paths connecting them.

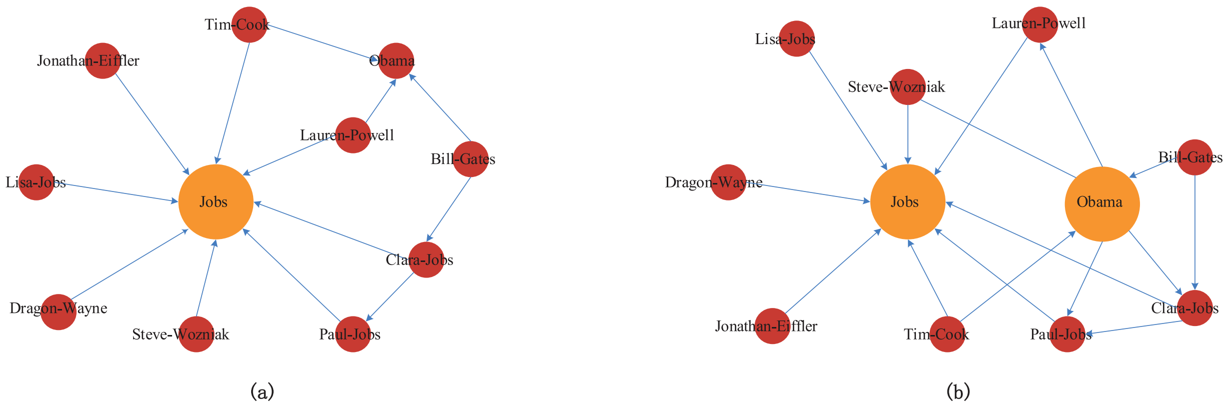

Fig. 4. Snapshot examples: (a) snapshot of a single entity; (b) snapshot of two entities.

Definition 6. Given an RDF knowledge base G, a finite set E of entities from G and two positive integers N and M, a snapshot consisting of E is a connected subnet S of G, including all nodes having a distance no more than N from at least one element of E, and all edges connecting two nodes of E with a distance no more than M.

Examples of snapshots are shown in Fig. 4. Simply speaking, a snapshot of an entity e is a subnet of G describing its most important features. A snapshot of an entity set E is defined as the union of snapshots of all entities in E, plus the additional subnet describing the relative features among entities of E. This very useful function is part of our HAPE knowledge graph browser (KGB). All snapshots are dynamically produced summaries of features of some object or objects during visitors' interaction with KGB. Different from template-based infobox, no template is needed for playing with snapshots.

5.4. Topic and topic pattern

Since several years ago, a research direction called topic modeling has attracted many researchers' attention, in particular, in the circle of natural language processing and information retrieval. In this research direction, topics are groups of words used to characterize and classify natural language texts. Topics show the main interests and viewpoints of the authors. Approaches for finding topics in articles are stochastical and are called topic modeling.

In this paper, we also use the terms topic and topic modeling, but in another context, namely KG, which is very different from the context of Natural Language (NL) texts.

Definition 7. A topic pattern P is a discourse unit represented as an entity relationship model consisting of a set of RDF triplets where some of which are v-triplets.

Definition 8. Given a topic pattern P consisting of n (>0) v-triplets and m triplets, a topic (instance) of P is the result of substituting all variables in P with entities, i.e., $m+n$ triplets, where variables with same names should be substituted with same entities.

There are many differences between topic modeling in natural language processing (NLP) literature and topic-based knowledge mining in this paper. Firstly, a KG contains general knowledge. It is not towards special opinions of particular NL texts. Secondly, KG is represented in regular RDF forms rather than NL texts. Thirdly, KG's topic modeling attempts to identify all frequent topic patterns while NL's topic modeling concentrates on finding out the central points of view of the authors. Fourthly, therefore, KG's topic modeling is enumerative while NL's is stochastic. Lastly, while in NLP literature a topic is represented by a group of words (unstructured), a topic pattern in this paper is organized as a labeled graph. An example of topic pattern is shown in Fig. 5 (a).

Note that a v-triplet can be considered as a special case of a topic pattern (one-dimensional pattern). Further note that the same triplet may be a component of several topic patterns. With the introduction of topics, we can generalize almost all operations on triplets to those on topics.

5.5. Possible world

The possible world is the loosest knowledge structure of HAPE. When the knowledge base of a KG is very large, it will be difficult to assure high confidence (MC5) since the cost of evaluation would be very high even if it is feasible. Further, any attempt to evaluate the massive consistency (MC9) and massive completeness (MC10) would be even more difficult and labor expensive than checking MC5. Regarding MC9, people have been involved in similar problems since a long time ago. For example, Lenat [26] has divided the large knowledge base Cyc in a set of micro theories to assure the knowledge consistency within each micro theory but given up the consistency between different micro theories. We divide

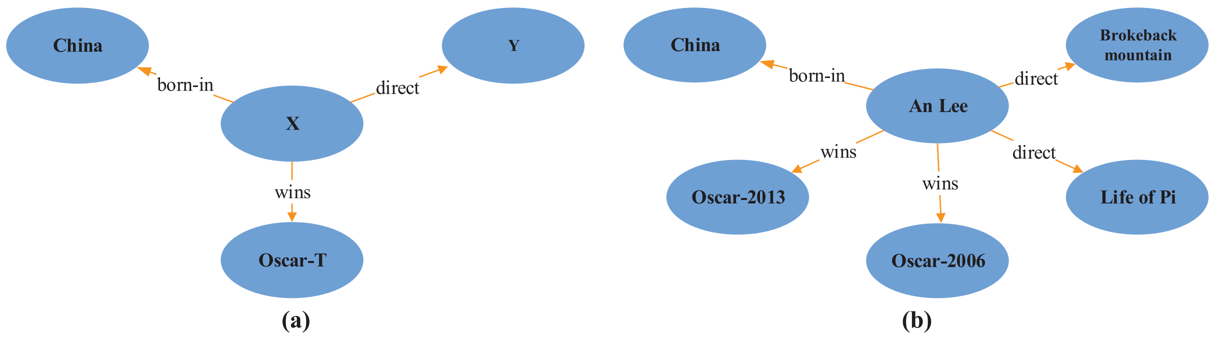

Fig. 5. (a) A topic pattern; (b) a subnet of a KG.

BKG's knowledge pool in different possible worlds in a way somewhat different from Lenat's approach. In BKG, the possible worlds are classified according to one or more criteria such as the following:

Different theories of knowledge: These theories involve different logics such as binary logic, multi-valued logic, fuzzy logic, probabilistic logic, etc.

Different qualities of knowledge: The quality standards include preciseness, trustworthiness, reliability, timeliness, etc.

Different structuredness of knowledge: Structured and unstructured knowledge may be stored separately, where differently structured knowledge may be stored in different possible worlds. Some possible structuring strategies are presented in different subsections above.

6. HAPE browser and the client side functions

The main component of HAPE's client side is the KGB which provides users with services regarding the KG like those provided by common Internet browsers (Explorer, Chrome, etc.) regarding World Wide Web (WWW), and even more. To the best of our knowledge, this is the first browser developed for KGs. There are several reasons for developing such browsers. Firstly, a KGB provides much better services by transplanting WWW browsers' search functions to KG. Everyone who has visited nowadays KGs will have the feeling that only very simple and straightforward search routines are available, which are not comparable with those provided by WWW browsers. Secondly, a KGB provides different ways of acquiring knowledge and information from the KG, which will be listed later in this section. In particular, knowledge operations in HAPE are not limited to triplets, but can also be applied to more complex knowledge structures. Thirdly, besides providing information search services, a KGB presents a rich set of user operation alternatives, e.g. displaying a video, broadcasting a piece of music, running an algorithm, answering a question, generating a survey about some news, etc. The services can be interactive. Lastly but not less importantly, a KGB serves as a firewall against hackers' intrusion. Due to the existence of this firewall, KG visitors cannot access KG's information arbitrarily. Any request from outside will be reviewed and evaluated, and only safe request will be accepted. Furthermore, visitors of KG are classified in different classes and levels where different visitors may have different degrees of visiting rights. That means that different visitors may find different knowledge contents and views by visiting the same KGB and inputting the same keywords.

In the following subsections, we list several major client side functions with details.

6.1. Knowledge browsing

Current KG usually provides functions for browsing HyperText Markup Language (HTML) and RDF knowledge resources, sometimes also infoboxes. Besides these functions, HAPE provides also high level utilities for browsing structured knowledge types like topics and taxonomies which are a generalization of infoboxes.

After user's keywords are inputted, they will be sent from the client side to the server side. The server side search engine will be then awaked to search whole or part of the KG to find the wanted knowledge. The server side search engine will find modules and knowledge elements fitting the keywords and list them in a sequence of results to be displayed to the user. The search can go further if the result obtained so far includes chaining to other knowledge modules or elements. When the search is successful the result will be sent back to the client side. It is then the client side's job to display the result to the user. Note that while displaying the list of search results all those items will be shielded if the visitor is not eligible to access them.

6.2. Knowledge displaying

Any result or response to the user, no matter generated by the client side or server side, will be rendered in user-friendly form and then displayed to the user. The rendering functions include research result ordering, screen design, knowledge structure (see Section 5) construction and operation, etc.

Table 2
Possible solutions of topic prediction and topic mining.

Four solutions	Decision
Ang Lee directs Brokeback Mountain and wins Oscar-2006	Correct
Ang Lee directs Brokeback Mountain and wins Oscar-2013	Wrong
Ang Lee directs Life of Pi and wins Oscar-2006	Wrong
Ang Lee directs Life of Pi and wins Oscar-2013	Correct

But this is not yet all. HAPE's service is presented in a multi-modal way. It may have to draw a picture, to show an image, or to broadcast a piece of music. What's even more is that HAPE may allow a moderate function of HAPE/user interaction in limited natural language. Although this part of service is now still in development, we expect it will be a powerful tool for populating HAPE in broader society. Figs. 3 and 4 are two examples of knowledge displaying.

6.3. Topic oriented knowledge mining

6.3.1. Different kinds of topic mining

Given a KG *G* and a topic pattern *P*, topic mining can be done in the following way: **Topic prediction** is a possible instance (possible topic) of *P* in *G*, where a possible instance of a topic pattern *P* is an instance of *P* where each v-triplet of *P* has been instantiated by a triplet of *G* such that all variables of the former have been instantiated by entities of the latter. Note that variables with the same name in different v-triplets should be instantiated with the same entity.

Why do we say that the instance we get above is only a possible one? It is because of the existence of ambiguity. Assume the pattern is {(X, is-a, Y), (Z, likes, W)}. Then {(Gauss, is-a, mathematician), (mathematician, likes, number)} is an instance. But {(Gauss, is-a, mathematician), (musician, likes, piano)} is only a possible instance since it lacks meaning.

However, despite these troubles, topic prediction is a basic operation, based on which alone many more complicated operations can be implemented, as exemplified by the following derived operations.

Topic instantiation: *P*'s all instances (topics) in *G*.

Topic completion: Given in addition a partly matched topic instance *I*, find in *G* all instances of *P*, which can make *I* a complete instance of *P*.

Solution: Take *I* (instead of *P*) as topic pattern and do topic mining.

Topic fusion: Given in addition a set of partly matched topic instances {*I*}, find in *G* at least one instance of *P*, which can make as much as possible part instances of {*I*} a combined instance of *P*.

Solution: Starting from the pattern *P*, construct a partial ordering of all partly matched instances by always selecting the fewer instantiated copies first, followed by their more instantiated copies. In this way, one gets a bundle of top down paths. Those instantiated copies which are endpoints of most top down paths are the solutions.

6.3.2. Topic prediction implementation

SPARQL[1] is a specialized language for RDF. It supports Basic Graph Pattern (BGP) query. A BGP is a set of triple patterns where a triple pattern is a v-triplet. Our preliminary implementation idea for Topic prediction is transferring the Topic prediction task to BGP query tasks. Please see the example below.

Example: Assume the topic pattern *P* is given in Fig. 5 (a). Further, assume the relevant knowledge is a subnet of the KG shown in Fig. 5 (b). Table 2 lists four possible solutions by mapping *P* to the net where some are true and some are wrong. Using topic prediction we will get one of its four results (in a non-deterministic way). Using topic mining we will get all its solutions (true and false). This shows that using a simple topic pattern to search for solutions in a large KG does not always provide rational solutions.

Although BGP can handle the exact query about a topic pattern, it does not always work well. For example, given a topic pattern (X, steal, bike), BGP can give the exact answers about who steals a bike, yet all of the related triples must have been stored in the KG. If we want to know who may steal a wallet, such knowledge is normally not stored in KG, and BGP will fail to answer. Now the embedding-based methods [3,28,43] may give a good solution, as such methods can give an approximate answer by inferring that the people stealing bike may steal a wallet, too.

6.4. Topic oriented knowledge embedding

There are two basic techniques used for implementing functions of Section 6.3. For obtaining exact and precise results we make use of SPARQL, while for calculating approximate or heuristic results we resort to embedding technique. One of the contributions of this paper is the generalization of triple based embedding to different levels, including at triple level, snapshot level, topic level, and knowledge base level (possible world level). One example of the second approach is the following basic idea for embedding topic mining (see Section 6.3).

[1] SPARQL Query Language for RDF, https://www.w3.org/TR/rdf-sparql-query/.

Basic idea: Assume a KG G and a topic pattern P is given. Let R be the set of all relations appearing in P. For any variable v in some v-triplet vt of P, randomly replace it with some entity in some triplet tp of G, where vt and tp have the same relation r. We get an instantiation T of P by doing this for all variables of the v-triplets. If there is any relation r in R, which does not appear in KG G, then the algorithm fails. Otherwise, we find and use a head resp. tail entity of a triplet of G to instantiate the head resp. tail variable in a v-triplet. In this way we may obtain a set T of different instantiations of P. Calculate their embedding and take the best approximate ones as solutions (topic instances mined from G). The solution is those T whose members have minimal loss function values. For more details see Section 7.3.2.

7. HAPE operating system and the server side functions

The main component of HAPE's server side is the knowledge graph operating system (KGOS) which performs almost all functions necessary for constructing, managing and developing BKG. These functions include four main modules: the acquisition engine, the evaluation engine, the computation engine, and the management engine, plus a kernel of KGOS in addition.

7.1. The knowledge acquisition engine

When we started the design of HAPE, we were aware that HAPE should uninterruptedly extract data and knowledge from all kinds of knowledge sources (see MC7 and MC8 above), where a major source is undoubtedly the World Wide Web. An even broader knowledge source is the Linked Open Data.[2] However, since we are just at the stage of building the HAPE framework, we do not have enough time and vigor to develop a working search engine at this moment. Instead, we have to concentrate our effort on borrowing existing knowledge from other BKG. In particular, we are attracted by DBpedia, a BKG which fulfills the MC7 and MC8 principles.

The quantity of triples in DBpedia is in billion-scale. Searching on such scale data timely and accurately is not an easy thing. Thanks to the application of inverted index technology, searching on billion-scale web pages becomes possible. We applied the inverted index technology to organize the billion-scale triples in DBpedia. Elasticsearch[3] is a distributed, RESTful-style search and data analytics engine based on open-source search software Apache Lucene.[4] We integrate different forms of DBpedia knowledge into a unified management index. As a result, HAPE can provide real-time search service for user queries.

7.2. The knowledge evaluation engine

7.2.1. Check the five massiveness characters of big knowledge

The first check of knowledge quality of BKG supported by HAPE is to check whether it fulfills the 10 MC criteria. Since knowledge representation varies in different BKSs, we have to find out the most suitable measures to check BK in form of BKG. There are the following possibilities:

- MC1-check: To be checked are the quantities of various types of knowledge elements, including the numbers of entities, triplets (facts) and concepts (predicates);
- MC2-check: To be checked is the balance of connections distribution: Number of connected node pairs/number of all node pairs;
- MC3: To be checked is the ratio of working knowledge/knowledge resource: Check1: quantity of knowledge elements/quantity of clean data resources Check2: quantity of high confidential knowledge/ quantity of remaining knowledge;
- MC4: To be checked is the service availability of BKG (based on logs): Check1: Number of visits (daily and total) by public visitors; Check2: Number of visits (daily and total) by managers; Check3: Number of revisions (daily and total) by managers;
- MC5: To be checked is the degree of confidence; Check1: Total: quantity of correct triplets/ quantity of all triplets; Check2: Partial Random check: : quantity of correct triplets/ quantity of checked triplets.

7.2.2. Data quality evaluation

HAPE also supports the regular data quality evaluation for KG. Evaluation aspects include preciseness, reliability, confidence, consistency, completeness, timeliness, etc. All of these aspects can be refined into smaller dimensions, as shown in [10] which gives a comprehensive survey for evaluation dimensions. We list some as follows.

Consistency:

Check1: Consistency of statements w.r.t. class constraints

Check2: Consistency of statements w.r.t. relation constraints

[2] The Linked Open Data Cloud, https://lod-cloud.net/.

[3] Elasticsearch, https://www.elastic.co/cn/products/elasticsearch.

[4] Apache Lucene, https://lucene.apache.org/.

Completeness:
Check1:Schema Completeness
Check2: Column Completeness
Timeliness:
Check1: Timeliness frequency of the KG
Check2: Specification of the modification date of statements

7.3. The knowledge computation engine

HAPE owns two kinds of computation: graph computation and deep computation. For the former, we introduce snapshot computation as an example. For the latter, we present an algorithm for embedding computation which is a very hot topic now in literature.

7.3.1. Graph computation

We consider any knowledge structure (triplet, taxonomy, snapshot, topic) of the KG or part of it as a graph. Various graph operations can be performed on it. All operations produce user wanted views of a KG, as a whole or in part. Due to the limitation of space, we only give the details of snapshot generation.

Snapshot aims to generate a knowledge subgraph with respect to given entities. The work in [4] proposed a task for finding the connected subgraph containing given entities from KG under a given diameter constraint. The difference between [4] and snapshot is that snapshot not only requires finding a connected subgraph of given entities but also needs to find the most important information associated with given entities. A detailed description of snapshot is given in Definition 6. It depends on two parameters N and M. Algorithm 1 implements its simpler version with only a single parameter N. The essence of this problem is a generalized breadth first search (BFS) which has multiple start points.

Algorithm 1: Multiple trees BFS for directed graph.

```
1: Let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of $n$ entities inspected in snapshot;
2: Until $E$ = empty do:
3: Clear the queue;
4: Take one element $e_i$ from $E$ and do:
5: Block1 Begin
6: Push $e_i = \{e_i(0)\}$ in the queue and mark it as visited, where 0 is the index number of $e_i$;
7: While at least one of the index numbers of one of $E$'s elements in queue is less than $n+1$ do:
8:     Pop off the queue's head element $h$;
9:     For each of its unvisited directed neighbor $f$ do:
10:        Block2 Begin
11:            For all indices $e_i(k_i)$ in $h$ put $e_i(k_i+1)$ in $f$;
12:            Add index $f(0)$ in $f$ and mark $f$ as visited;
13:            If $f$ is in $E$ then remove it from $E$;
14:            Put $f$ at the tail of the queue;
15:        Block2 End;
16: End of While;
17: Block1 End;
18: End of Until
```

In this context, a KG is a directed graph in standard sense whose nodes are entities and edges are relations. A triple (subject, relation, object) is considered as a directed edge from subject node to object node, with relation as edge name. Note that no hyper graph (i.e. with self-loop edges or multiple edges between two neighboring nodes) is allowed.

The basic idea of Algorithm 1 is as follows. It performs the search procedure as if it were the usual BFS by maintaining a queue with the only (but important) difference that each element in the queue not only represents a node of the graph. It represents a search path from the starting node until the current node including the distance from each node on the path to the current node. For example, if the search path walked from e_1 through e_2 to e_3 then the queue element e_3 is represented as $e_3 = \{e_1(2), e_2(1), e_3(0)\}$. Each $e_j(k)$ in the parentheses is called an index, k is called the index number.

7.3.2. Embedding computation

We limit our discussion to a simpler case, namely the v-triplets have either at head or at tail place a variable, but not both. Assume the topic pattern P has m v-triplets, where head elements of j triplets and tail elements of m-j triplets are variables, which are denoted with Eq. (2).

$$P_1 = \bigcup_{1 \le i \le j} \{(X_i, r_i, t_i)\},$$

$$
\begin{aligned}
P_2 &= \bigcup_{j+1 \le i \le m} \{(h_i, r_i, X_i)\}, \\
P &= P_1 \cup P_2
\end{aligned}
\tag{2}
$$

This means in the simpler case we only allow one variable in each v-triplet. Mining a topic P is to find all best fit instantiations T_i of P where the head resp. tail variables of P's triplets are replaced with head resp. tail entities of appropriate triplets of knowledge graph G. By embedding $(T_i + G)$ in vector space one can find out those topic instances with minimal loss of preciseness. More precisely, each instance T of P is represented as:

$$
\begin{aligned}
T_1 &= \bigcup_{1 \le i \le j} \{(h_i, r_i, t_i) | \exists (h_i, r_i, s_i) \in G\}, \\
T_2 &= \bigcup_{j+1 \le i \le m} \{(h_i, r_i, t_i) | \exists (s_i, r_i, t_i) \in G\}, \\
T &= T_1 \cup T_2
\end{aligned}
\tag{3}
$$

There are different ways to do the embedding. For the current case we follow the TransE [3] approach and map all elements of the triples to the same vector space. Let f be the vector embedding function, Los be the loss function, then

$$
\begin{aligned}
Los((h, r, t)) &= \|f(h) + f(r) - f(t)\|, \\
Los(T) &= \sum_{(h,r,t) \in T} Los((h, r, t))
\end{aligned}
\tag{4}
$$

In this regard we request the best K solutions for a predefined positive integer K, where the best solution means that solution with minimal loss value. The second best solution means the best solution after neglecting the overall best solution, etc.

7.4. The knowledge management engine

This part of KGOS performs a series of important responsibilities: 1. Take control over the running of all functions of the server side. 2. In particular, control the life cycle (see [29]) of the platform evolution. 3. Regularize the cooperation of server and client sides. 4. Take responsibility of platform security which will be discussed in detail in Section 10. 5. Allocate storage to different information and function units according to criterions like the possible worlds and clean data resources (for examples see Section 9.2), and keep its update dynamically. 6. Host and control all meta information, including metadata and meta knowledge.

The most extraordinary function of knowledge management in KGOS is the data storage function. KGOS manages three copies of the same knowledge store of the KG: the first one in Resource Description Framework Schema (RDFS) form, the second one in simplified symbolic form and the third one in digital form, more exactly in form of an embedded subset of a low dimensional Euclidian space. HAPE keeps the first form because it is the standard, the second form because it saves space and time and is thus good for doing reasoning, and the third form because it spares time when doing embedded reasoning. In fact, many of HAPE's operations are performed based on this representation.

Last but not least, we list the functions of the KGOS kernel. It is very different from a kernel of the currently available OSs. The kernel of KGOS is knowledge-centered. It does not collect all the important things about KGOS. It only cares about how to protect HAPE's knowledge storage from any kind of sabotage or disaster and how to enhance its capability of user service. Therefore it also enjoys the name "knowledge kernel". Its functions include, but are not limited to resist to invalid visits of KG, check newly acquired data and knowledge to avoid rubbish, virus and Trojan horse, protect sensitive data from information leakage, operate runtime examination routines and take care of security devices such as sandboxes.

8. Scripting programming languages

To support the strong request of runtime support of KGB and KGOS, we decided to make use of scripting languages to design and implement both KGB and KGOS, where we use JavaScript for the basis of client side programming and Python for the server side. Currently, all programs of KGB resp. KGOS are written in JavaScript resp. Python, where the former has been well known for writing scripts since long time ago, whereas the latter is currently a hot technique among application programmers.

Python is an interpreted and general-purpose scripting language [34], which has the characteristics of simple and easy to learn, strong readability, a large number of ready-made tool libraries and frameworks, etc. JavaScript is a dynamic scripting language [12]. It can also respond directly to the input of clients without a Web service for faster running. Moreover, JavaScript is cross-platform such that it only relies on the browser without depending on the OS.

Although Python and JavaScript have significant advantages for writing both server-side and client-side functions, neither of them can be used for knowledge-oriented programming directly. Our HAPE is a platform for knowledge manipulation and management. In most cases, we need a language used for programming knowledge directly.

Currently, a variety of RDF programming languages have been proposed, such as Neno [36], Ripple [39], FABL [17], Adenine [35], etc., which take unique advantages of the Semantic Web infrastructure to encode both data and process information within RDF. In addition, there are also well-known RDF programming tools, such as rdflib [24] and RDFJS,[5] etc. These tools first query the RDF by means of the SPARQL query language and then process the information obtained to generate the final result. However, such languages and tools are insufficient to meet the requirement of KGB and KGOS. Hence, we have designed and implemented two extensions of JavaScript and Python respectively. They are K-script-c, the language for programming client side functions, which is a superset of JavaScript; and K-script-s, the language for programming server side functions, which is a superset of Python, where the letter K stands for knowledge.

When designing K-script-s and K-script-c, we mainly consider four aspects:

1. Compatibility of syntax. It means that K-script-s and K-script-c have no conflicts in their syntax.
2. Consistency of semantics. It means that there is no ambiguity in the interpretation of K-script-s and K-script-c statements.
3. Completeness of facilities. K-script-s and K-script-c not only collect all delicate knowledge processing facilities from current RDF manipulation languages but also introduced those facilities for processing high level knowledge structures.
4. Coincidence with the application. K-script-s and K-script-c have different facilities and powers towards their application on server side resp. client side.

We take the implementation of K-script-c's high level knowledge structure processing as an example to provide some details.

8.1. Part of K-script-c functions for knowledge processing

As a knowledge-oriented scripting language, K-script-c has rich functions for knowledge programming and it is an extensible scripting language. At present, we have implemented the functions of Triplet, Topic, and Taxonomy, which are three types of structured objects of K-script-c.

The main functions of K-script-c on Triplet include: get the left (right) item of the Triplet, get the value of a variable, etc. Several examples are given as follows:

Example 8.1: For an instance of a Triplet,

Triplet Director = (Zhang_Yimou, birthPlace, X)

where X is a variable, Director.*getVarValue*( ) returns “China”.

The main functions of K-script-c on Topic include: get all properties of the Topic, get the value of the property; get the domain of the property, etc.

Example 8.2: For an instance of a Topic,

Topic Rise_of_Star = [name: identifier, born: date, studied: concept, supervisor: identifier, stared: phrase, awarded: phrase]

where Rise_of_Star is the name of this Topic, Rise_of_Star.*getAllRelation*( ) returns [identifier, date, concept, identifier, phrase, phrase].

The main functions of K-script-c on Taxonomy include: get the leaves of the node, get the path of two nodes, update child nodes information, etc.

Example 8.3: For an instance of a Taxonomy,

Taxonomy root:[node1,node2:[node21, node22]]

root.*getLeavesOf*( ) returns [node1, node21, node22], and node1.*pathLengthTo*(node21) returns 3.

8.2. Some hints on K-script-c implementation

In K-script-c, Triplet, Topic, and Taxonomy are the basic units in knowledge processing. Considering that all of them are subject to multiple operations, when converting them to the corresponding JavaScript code, we convert them to class objects of the corresponding scripting language. More precisely, we encapsulate the functions of Triplet, Topic, and Taxonomy into TripletClass, TopicClass, and TaxonomyClass, respectively, where the last three are implemented as class objects in JavaScript.

The conversion rules for Triplet and Topic are as follows:

1. Use an intermediate variable (denoted as tmpVariable) to represent the contents of Triplet/Topic;
2. Use the variable name in the original code of the Triplet (Topic) as the variable name of the instantiated class TripletClass (TopicClass), and passing tmpVariable as a parameter of the class at the same time.

The conversion rules for Taxonomy are as follows:

1. Generate a class for each node in Taxonomy in the order they were created from left to right. If the node is the root node, it originates from the TaxonomyClass, otherwise, it extends its parent node;
2. According to the order of the generated classes in step 1, instantiate each class with corresponding parameters; and
3. According to the reverse order of class instantiation in step 2, for each instantiated node, invoke its *update*(*children*) function and bind the node name with its invoked result.

[5] Rdf javascript libraries, https://rdf.js.org/.

Table 3
Organization of DBpedia-BK's HKB.

Entities	Relations	Triplets	Taxonomies
4236105	2727	64,533,020	685

Table 4
Organization of DBpedia-BK's SKB.

DBpedia- standard	DBpedia- simplified	DBpedia- embedded	Yago	Opencyc	Freebase
25/03/2019 8.5G	06/04/2019 2.6G	11/04/2019 17.6G	12/04/2019 91.6G	12/04/2019 0.2G	14/04/2019 396G

9. From DBpedia to DBpedia-BK: a HAPE based BKG

We tested the HAPE server side functions on a server whose configuration consists of two E5-2620 V4 CPU, 96GB memory and Centos 7.0 OS. All server side functions are written in Python 3.5. The client side functions are written in JavaScript and tested in Google Chrome 73.0 version. We deployed a SPARQL endpoint based on Virtuoso 7.x[6] and developed our applications based on Django[7] which is a Python-based free and open-source web framework. For dealing with various tasks, we stored KG in different forms including relational database PostgreSQL 10.1,[8] graph database Neo4j 3.4.4,[9] with full-text search engine Elasticsearch 6.6.2.

9.1. A short introduction to DBpedia

DBpedia is a widely used KG extracted from Wikipedia by community effort. It has been the most prominent KG in the LOD cloud. Since the first public release in 2007, DBpedia has been updated nearly once a year and used extensively in the Semantic Web research community.

Although DBpedia has vast and wide coverage of knowledge content, it is not well structured. Only the part defined in DBpedia Ontology has a clear hierarchy. In the English version of DBpedia 2014, DBpedia Ontology covers 685 classes, described by more than 2700 different properties. It contains about 4.23 million entities. We used this part of data, and extended it by adding all triplets in form of (X, rdf:Type, owl:Thing) where X's rdf:Type relation with owl: Thing is not explicitly declared. The detailed data distribution is showed in Table 3.

9.2. Knowledge organization of DBpedia-BK

How to organize knowledge is a critical issue in particular because of the massiveness of knowledge. A badly organized knowledge base will be very ineffective and even hardly useful. The organization strategy should not be based on a single computer, and due to the huge size of BKG distributed configurations should be taken into consideration.

The transported part of DBpedia, DBpedia-BK, is organized as follows, where it is divided into four large components:

The main knowledge body HKB: It is a set of structured knowledge modules as described in Section 3, stored in different knowledge areas, and covering the knowledge content of MC1-2.

In practice, RDF knowledge is represented in different forms of serialization. To name a few, they include RDF/XML, N-Triples, Turtle, RDFa, JSON-LD, etc. These forms are usually not convenient for use in knowledge operations. HAPE keeps both a standard copy of RDF knowledge and a simplified copy of it to ease various kinds of knowledge operations. This simplified copy is purely symbolic without Web address marking. Besides, the HAPE manager keeps a digital copy of the same RDF knowledge, which is generated through a specific embedding used by the manager to do daily monitoring and maintenance of HAPE's knowledge base. This digital copy is not provided to HAPE visitors since their needs for knowledge embedding may be very different. Only a set of coded embedding algorithms are available such that the visitors are free to choose any of them for use, e.g. in case of triplet embedding one can choose TransE [3], TransH [43] or TransR [28] at will. HAPE provides routines of all these three algorithms, but no embedded knowledge bases are based on them.

The current HKB knowledge store of DBpedia-BK is shown in Table 3. The source of these data has been given in Section 9.1. The taxonomies are inherited from the DBpedia ontology hierarchy, a total of 685.

The supporting knowledge body SKB: It includes all kinds of row data (unprocessed source data and inputted knowledge modules) and semi-processed (multiple times cleaned or mined) ones, including clean data resource (MC3).

The current SKB knowledge store (data quantity in bytes) of DBpedia-BK is shown in Table 4, where sources, dates, and quantity of inputted knowledge are recorded. Note that although the main part of HAPE's knowledge came from DBpedia,

[6] Virtuoso 7.x, https://shop.openlinksw.com/.
[7] Django, https://www.djangoproject.com/.
[8] Postgresql 10.1, https://www.postgresql.org/.
[9] Neo4j 3.4.4, https://neo4j.com/.

Table 5
Evaluation of BKG quality in five massiveness characteristics.

	MC1	MC2	MC3	MC4	MC5
Threshold	Entities 4,236,105	Facts 64,533,020	KR/WK 38.11	Visits 11	CONF N/A
Statistics	Concepts 1788	Connectedness 96.41%	RF/OWN 2.18	APPs N/A	Rdm. Conf. 0.998

including their different forms DBpedia-standard, DBpedia-simplified and DBpedia-embedded, we have already started collecting knowledge from other sources such as Yago, Opencyc, and Freebase. Currently, this part of knowledge forms the clean data resource of HAPE. The fusion of knowledge from different sources is being considered but has not yet started. DBpedia-embedded is stored as a trained model file, where the training model is TransR and the vector dimension is set to 50. The last line of Table 4 shows the capacity of DBpedia-BK's SKB.

The meta knowledge body MKB: It includes all information for managing, benefiting and developing HAPE, such as (data, knowledge, knowledge structure) indexing tables, knowledge test and evaluation records (MC5, 7–10), knowledge revision and version change records, records of application cases of the HAPE (MC4), including those of small applications (such as daily HAPE visits) and large ones (for solving complicated problems). It includes also a depository of BKG processing tools, which represent mainly a network of algorithms and APPs calling each other.

The working space WS: It is a space for any kinds of data and knowledge processing, including different kinds of buffers.

We cannot yet report the statistical data of the current MKB knowledge store of DBpedia-BK. Since HAPE is a brand new KG platform for experimental motivations, there are not yet real data in this table. However, the input/output/registration routines have all been programmed and tested.

It is easy to see that while the three components listed above are only loosely connected with each other, the internal connections within each component are tight and complicated. This reminds us that they can be located on different computing devices connected by a network.

9.3. Evaluation of BKE quality

HAPE checks the quality of its BKG from various aspects. In this paper, we only present those data certifying/disproving the eligibility of DBpedia-BK being a kind of BK. Table 5 shows the main results of evaluation, where facts mean triplets, connectedness means the number of connected entity pairs divided by that of all pairs, KR/WK means quantity of knowledge resources divided by quantity of working knowledge, RF/OWN means quantity of referred external knowledge divided by all in-house knowledge, APPs means number of user visits, and CONF/ Rdm. CONF means overall (randomly calculated) confidence.

Our statistical concepts include classes and the ObjectProperties in DBpedia Ontology. ObjectProperties can express conceptual meaning, hence we consider them as concepts. For calculating the connectedness of entities, we randomly selected 10,000 pairs of entities from our KG and tested whether they were connected by paths. The calculation of KR/WK and RF/OWN in our case is special. Considering the facts in DBpedia-BK as WK (65M) and all facts introduced from Freebase (1.9B), Opencyc4 (2.09M) and Yago3 (120M) as KR (2.022B), we get KR/WK = 31.11. On the other hand, considering the external knowledge sources DBpedia (2.935B), Freebase (1.2B), and Yago3 (420M) as RF, all facts of KR + WK as OWN (2.087B), we get RF/OWN = 2.18.

The visits are still hard to count because our platform is new and have not yet provided service for the public. However, as an initial probe, we have developed more than 10 apps on our platform.

10. Big knowledge security

It is particularly important to assure the security of BKG. As a general architecture supporting BKG development, management and application, HAPE provides multiple security measures. They form three groups of security techniques.

Group 1, Visitor targeted security:

Registration: This is the first firewall and keeps visitor information in multiple aspects and is authorized to block the visit if information shows that the current visit might be dangerous.

Authorization: Visitors are divided in 4 authorization classes. Each visitor has only limited access according to the class s/he belongs to. Limitations include those referring to the range and quantity of data access and confidence levels.

Group 2, Knowledge targeted security:

Knowledge integrity check: Check each knowledge input with a set of rules for some types of relations.

Knowledge storage distribution: By using the possible world mechanism, knowledge of different confidence levels is stored in different areas of the BKG and protected at different levels.

Knowledge enciphering: Part of the knowledge elements, modules or areas can be enciphered or renamed with an alias.

Knowledge buffering: In case of batching knowledge, the inputted "rough" knowledge will be buffered in some sandbox. It will only be merged into the main store of BKG after it has undergone a security and reliability check.

Group 3, System targeted security:
Key node protection: Extra enciphering for critical nodes of the knowledge network to avoid network collapse.
Knowledge back-up: Keeping back-up copies for important knowledge areas.
Exception handling: Alerting system manager.

11. Concluding remarks

This paper introduces a programmable knowledge graph platform HAPE, including its design and implementation. Compared with relevant works published in the literature, HAPE has the following innovation and advantages. Firstly, it is a product and continuation of BK research [29], where the concept was first thoroughly studied. In this paper, the BK concept was applied to the case of the massive size KG, which led to the concept of the BKG, for which HAPE is a platform. Secondly, different from many current large KGs where only very limited services are provided to the visitors, this paper proposes a client-server architecture of BKG management, such that the focus is placed on providing a well-formed set of visitor services. Thirdly, in this paper, we summarize the client side functions as a KGB and those of server side a KGOS. These designations can better reflect the essence of our client-server structure. They largely affect the design of functions available on both sides. Fourthly, as an example, a well-designed set of knowledge embedding routines is provided for visitor's use. It means that HAPE's knowledge base is not only used for visitor's browsing but is also used for visitors' research with up-to-date knowledge processing techniques. Fifthly, different from most KGs, where the huge quantity of knowledge is a plain database only consisting of RDF triplets, HAPE introduces high level knowledge structures such as snapshots and topics, which leads to a better structuring of knowledge base and can benefit various types of applications. Sixthly, HAPE adopts currently popular scripting techniques and makes use of languages such as JavaScript and Python to program both sides of the platform resp. such that HAPE is re-programmable at any time, which makes the upgrading of the platform easy. Seventhly, since JavaScript and Python are general-purpose programming languages not necessarily friendly for knowledge programming, we have designed two extensions of them, namely K-script-c and K- script-s, to allow users to write knowledge processing specific instructions. Translation programs from the latter two languages to the former two are provided. This makes the upgrading and version changing of HAPE even easier.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors wish to thank the anonymous reviewers sincerely for their valuable comments and suggestions. This work has been supported by National Key Research and Development Program of China under grant 2016YFB1000902, National Natural Science Foundation of China (Nos. 61232015, 61472412, and 61621003), Tsinghua-Tencent-AMSS Joint Project: WWW Knowledge Structure and its Application.

References

[1] J. Aaron, Turning big data into big knowledge, https://techerati.com/the-stack-archive/big-data/2015/12/07/turning-big-data-into-big-knowledge/ (2015).
[2] A. Akesson, Google my business profiles start ranking in non-branded searches, https://www.venndigital.co.uk/blog/google-my-business-profiles-start-ranking-in-non-branded-searches-78887/ (2012).
[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Advances in Neural Information Processing Systems, 2013, pp. 2787–2795.
[4] G. Cheng, D. Liu, Y. Qu, Efficient algorithms for association finding and frequent association pattern mining, in: International Semantic Web Conference, Springer, 2016, pp. 119–134.
[5] M. Cox, D. Ellsworth, Managing big data for scientific visualization, in: ACM Siggraph, 97, 1997, pp. 21–38.
[6] Cycorp, Opencyc, http://www.baike.com/wiki/OpenCyc.
[7] F. Daniel, B. Volha, B. Christian, Dbpedia version 2014 released, http://wiki.dbpedia.org/news/dbpedia-version-2014-released (2014).
[8] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, W. Zhang, Knowledge vault: a web-scale approach to probabilistic knowledge fusion, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 601–610.
[9] M. Färber, F. Bartscherer, C. Menne, A. Rettinger, Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago, Semant. Web 9 (1) (2018) 77–129.
[10] M. Färber, F. Bartscherer, C. Menne, A. Rettinger, Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago, Semant. Web 9 (1) (2018) 77–129.
[11] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M.J. Strauss, R.N. Wright, Secure multiparty computation of approximations, in: International Colloquium on Automata, Languages, and Programming, Springer, 2001, pp. 927–938.
[12] D. Flanagan, Javascript: The Definitive Guide, O'Reilly Media, Inc., 2006.
[13] M. Galkin, S. Auer, S. Scerri, Enterprise knowledge graphs: a backbone of linked enterprise data, in: 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI), IEEE, 2016, pp. 497–502.
[14] M. Galkin, S. Auer, M.-E. Vidal, S. Scerri, Enterprise knowledge graphs: a semantic approach for knowledge management in the next generation of enterprise information systems, in: ICEIS (2), 2017, pp. 88–98.
[15] J. Gantz, D. Reinsel, The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east, IDC iView: IDC Analyze the future 2007(2012) (2012) 1–16.

[16] GNOSS, Knowledge graph management and metadata management, https://www.gnoss.com/en/semantic-framework/knowledge-graph-management (2018).
[17] C. Goad, Describing computation within Rdf, in: The Emerging Semantic Web, 2001.
[18] Google, Freebase data dumps, http://www.freebase.com (2013).
[19] P. Haase, D.M. Herzig, A. Kozlov, A. Nikolov, J. Trame, Metaphactory: a platform for knowledge graph management, Semant. Web (Preprint) (2019) 1–17.
[20] S. Hershkovitz, An analyst toolkit: From big data to big knowledge, https://www.linkedin.com/pulse/analyst-toolkit-from-big-data-knowledge-dr-shay-hershkovitz/ (2016).
[21] J. Hoffart, F.M. Suchanek, K. Berberich, E. Lewis-Kelham, G. De Melo, G. Weikum, Yago2: exploring and querying world knowledge in time, space, context, and many languages, in: Proceedings of the 20th International Conference Companion on World Wide Web, ACM, 2011, pp. 229–232.
[22] J. Jarvis, Google knowledge graph has more than 70 billion facts, https://twitter.com/jeffjarvis/status/783338071316135936, October 2016 (2016).
[23] Y. Jia, Y. Wang, X. Cheng, X. Jin, J. Guo, Openkn: an open knowledge computational engine for network big data, in: Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, IEEE Press, 2014, pp. 657–664.
[24] D. Krech, Rdflib: a python library for working with rdf, https://github.com/RDFLib/rdflib (2006).
[25] D. Laney, 3d data management: Controlling data volume, velocity and variety, META group research note, 2001.
[26] D.B. Lenat, R.V. Guha, Building Large Knowledge-Based Systems; Representation and Inference in the Cyc Project, Addison-Wesley Longman Publishing Co., Inc., 1989.
[27] J. Liebowitz, How to extract big knowledge from big data? https://www.druckerforum.org/blog/?p=655 (2014).
[28] Y. Lin, Z. Liu, M. Sun, Y. Liu, X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in: Twenty-ninth AAAI Conference on Artificial Intelligence, 2015.
[29] R. Lu, X. Jin, S. Zhang, M. Qiu, X. Wu, A study on big knowledge and its engineering issues, IEEE Trans. Knowl. Data Eng. 31 (9) (2018) 1630–1644.
[30] F. Mahdisoltani, J. Biega, F.M. Suchanek, Yago3: a knowledge base from multilingual wikipedias, in: CIDR, 2013.
[31] D. Makinson, Brian f. chellas. Modal logic. an introduction. cambridge university press, cambridge etc. 1980, xii+ 295 pp, J. Symb. Logic 46 (3) (1981) 670–672.
[32] J.P. McCusker, S.M. Rashid, N. Agu, K.P. Bennett, D.L. McGuinness, The whyis knowledge graph framework in action, in: International Semantic Web Conference, 2018.
[33] K. Murphy, From big data to big knowledge, in: Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management, ACM, 2013, pp. 1917–1918.
[34] T.E. Oliphant, Python for scientific computing, Comput. Sci. Eng. 9 (3) (2007) 10–20.
[35] D. Quan, D. Huynh, D.R. Karger, Haystack: A platform for authoring end user semantic web applications, in: International Semantic Web Conference, Springer, 2003, pp. 738–753.
[36] M.A. Rodriguez, General-purpose computing on a semantic network substrate, in: Emergent Web Intelligence: Advanced Semantic Technologies, Springer, 2010, pp. 57–102.
[37] A. Russell, Turning big data into big knowledge, https://communication.ucdavis.edu/research/research-spotlight/turning-big-data-into-big-knowledge (2014).
[38] B. Schwartz, Googles knowledge graph has an error 20% of the time, https://searchengineland.com/googles-knowledge-graph-errors-126098 (2012).
[39] J. Shinavier, Functional programs as linked data., in: SFSW, 2007.
[40] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledge base, Commun. ACM 57 (10) (2014) 78–85.
[41] Z. Wang, J. Li, Z. Wang, S. Li, M. Li, D. Zhang, Y. Shi, Y. Liu, P. Zhang, J. Tang, Xlore: a large-scale english-chinese bilingual knowledge graph, in: International Semantic Web Conference (Posters & Demos), 1035, 2013, pp. 121–124.
[42] Z. Wang, H. Wang, W.-Y. Ma, Probase, https://www.microsoft.com/en-us/research/project/probase/ (2016).
[43] Z. Wang, J. Zhang, J. Feng, Z. Chen, Knowledge graph embedding by translating on hyperplanes, in: Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.
[44] S.M. Weiss, N. Indurkhya, Predictive Data Mining: A Practical Guide, Morgan Kaufmann, 1998.
[45] M. White, Digital workplaces: vision and reality, Bus. Inf. Rev. 29 (4) (2012) 205–214.
[46] D. Woods, Is the enterprise knowledge graph finally going to make all data usable?, https://www.forbes.com/sites/danwoods/2018/09/19/is-the-enterprise-knowledge-graph-going-to-finally-make-all-data-usable/#ede11517d39a (2018).
[47] W. Wu, H. Li, H. Wang, K.Q. Zhu, Probase: a probabilistic taxonomy for text understanding, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, ACM, 2012, pp. 481–492.
[48] X. Wu, H. Chen, G. Wu, J. Liu, Q. Zheng, X. He, A. Zhou, Z.-Q. Zhao, B. Wei, M. Gao, et al., Knowledge engineering with big data, IEEE Intell. Syst. 30 (5) (2015) 46–55.
[49] X. Wu, J. He, R. Lu, N. Zheng, From big data to big knowledge: Hace+ bigke, Acta Autom. Sin. 42 (2016) 965–982.
[50] Z. Zhao, S.-K. Han, I.-M. So, Architecture of knowledge graph construction technique, Int. J. Pure Appl. Math. 118 (19) (2018) 1869–1883.