# A Scalable and Plug-in Based System to Construct a Production-Level Knowledge Base

Tomoya Yamazaki, Kentaro Nishi, Takuya Makabe, Mei Sasaki,
Chihiro Nishimoto, Hiroki Iwasawa, Masaki Noguchi, and Yukihiro Tagami
{tomoyama, kentnish, tmakabe, mesasaki, cnishimo,
hiwasawa, manoguch, yutagami}@yahoo-corp.jp

Yahoo Japan Corporation

**Abstract.** Knowledge bases play crucial roles in a wide variety of information systems, such as web search engines and intelligent personal assistants. For responding to constantly fluctuating user information demands, we aim to construct a large-scale and well-structured comprehensive knowledge base from the world's evolving data. To maintain the quality of our large knowledge base at the production-level, we carefully design not only to match entities but to incorporate various automatic and manual validation methods because it is difficult to filter out all incorrect facts automatically in practice. In this paper, we propose a novel plugin-based system architecture satisfying the ability to rapidly identify mistakes and the system extensibility. Our constructed knowledge base is already utilized in Japanese Web services, and the number of entities in it keeps growing steadily.

**Keywords:** knowledge base · entity matching · data integration

## 1 Introduction

Knowledge Bases (KBs) have been supportive of many user activities such as browsing, searching or buying experiences on the Web. Well-known search engines, such as Google, Bing and Yahoo! Inc., show not only the search results, called 10 blues links but the information as *entity panels* provided by their own KBs, i.e., Google KG (https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html), Satori (https://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/), and Yahoo! KG [2], respectively. Our *Japanese KB* (*JKB*) is one of the largest KBs in Japanese and has the same uses as them.

KB construction systems differ depending on the application, for example, NOUS [4] is a domain-specific KB construction system, and many machine-learning-based approaches, such as Fonduer [9], have been proposed to maximize the $F_1$ score on the quality of the output knowledge base. P. Suganthan et

DI2KG 2019, August 5, 2019, Anchorage, Alaska. 

al. [5] proposed that industrial systems are considerably differerent to those in academia regarding objectives; therefore, we cannot merely adopt such one-shot KB construction systems. The main objective of our KB construction system is not to maximize $F_1$, but to maintain high precision at all time, while trying to improve recall over time, because displaying incorrect information leads to losing our trust. Since the qualities of KBs include various types of measurement such as the precision and recall of the entity matching, the ratio of invalid or incorrect data, the ratio of missing relation etc., we design that our KB construction system can control its measurement and evaluate each algorithm or software component one by one. Therefore, plugin-based system architectures are critical to satisfying many system requirements such as the ability to fix incorrect data immediately and the interchangeability of each method. This plugin-based system allows to implement various algorithms and also satisfy business requirements, while the system focuses on ensuring our SLAs and scaling computation.

WOO [1] is one of the plugin-based KB construction systems in Yahoo! Inc. and is designed to enable various types of products to synthesize KBs. We describe the procedure of how to output KBs from input data via the WOO as follows: (1) Import various scheme of data and normalize input data to the common format, (2) Match entities by some entity matching algorithms, (3) Assign persistent ID, and (4) Export KB in any output format. In this paper, we extend the WOO regarding maintain high precision from multiple measurement perspectives and explore loose-coupling and plugin-based architectures. In addition to the above four primary functions of WOO, our KB construction systems validates incorrect data and completes missing relations and facts. We carefully design the system architecture such that it is easily extensible.

## 2 Our KB Construction System Architecture

Our KB construction system is designed to handle hundreds of millions of entities covering our wide-ranging domain services (e.g., books, movies, companies, landmarks etc.,) and constructs a large production-level KB every day by using Apache Spark. Our system is mainly composed of twelve components shown in Figure 1, and we roughly divide the roles into two groups whether input data are Web-crawled data or not. We daily collect and update structured, or semi-structured data such as linked open data (LOD, e.g., Wikidata, Wikipedia, DBpedia, or Freebase), content provider (CP) data (e.g., landmark, movie or book data), and Web-crawled data by using our Web crawler. These three types of data have different characteristics regarding their accuracy and the complexity of how to extract key-value information, as shown in Table 1.

Our system imports such structured and semi-structured data, other than Web-crawled data, from the Importer to the Exporter through other ten primary components. As shown in Table 1, extracting key-value information from Web-crawled data is another challenge due to the combination of various input

https://plus.google.com/u/0/109936836907132434202/posts/bu3z2wVqcQc

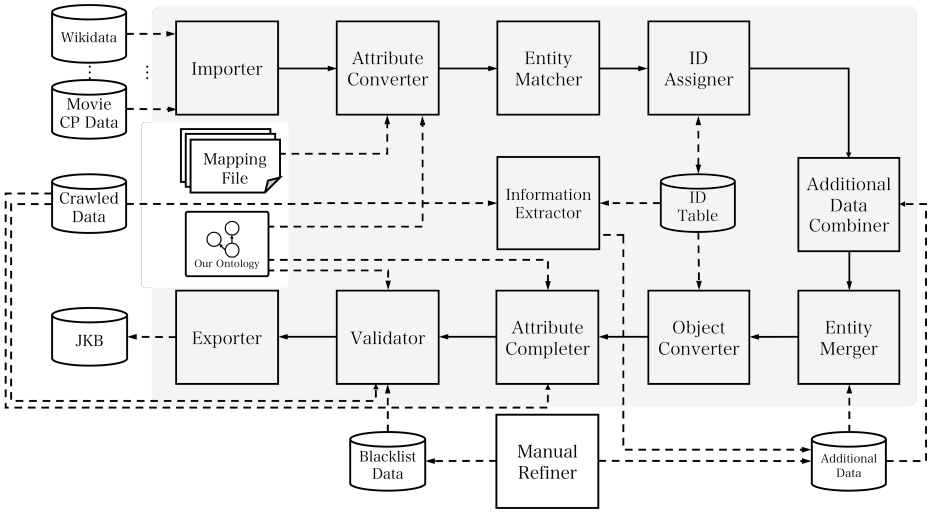


**Fig. 1.** Overview of our KB construction system architecture. Solid arrows, dotted arrows, squared frames, rounded cornered frames, and cylindrical objects represent data flows between our system and the external data, data flows within it, primary software components, input/output data, and preserved data, respectively. Mapping files are for mappings of entity types and relations from input data to these types of our ontology.

documents and variation in extraction targets [3]; thus, we design our information extraction (IE) method separately from the main components and retrieve the IE data as additional data.

### 2.1 The JKB Scheme

We introduce a new scheme for KBs and explain its advantages. The JKB is a set of *entities*. An entity is composed of a unique id, types (e.g., PERSON and WRITTEN WORK), and a set of *Triples*. A Triple is similar to the RDF scheme (https://www.w3.org/TR/rdf11-concepts/) that is composed of subject, predicate, and object. Moreover, triples in the JKB scheme have its certainty score as with YAGO [8], the data type. We can maintain the quality of the JKB by filtering triples following whether the certainty score is higher than a threshold or the data type is consistent to our ontology. Since the JKB scheme is allowed to add arbitrary meta information such as data source, we can easy to debug by tracing the data source.

### 2.2 Primary Functions of Each Component

**Importer** is a data feed component. It supports arbitrary input data scheme, except for Web-crawled data, as shown in Figure 1, and unifies with the JKB

**Table 1.** Comparison of the input data regarding the primary information. We calculate data size from JKB formatted data.

| Data sources | Data size | Data Accuracy | Complexity to use | Update frequency |
|---|---|---|---|---|
| Wikidata | 300 GB | Fair | Low (Json) | About every two weeks |
| DBpedia (Japanese) | 300 GB | Fair | Low (N-Triple) | About once a year |
| Freebase | 700 GB | Fair | Low (N-Triple) | End at June 30, 2015 |
| Wikipedia (Japanese) | 5.6 GB | Good | Medium | About every two weeks |
| CP data (Japanese) | ≈ 500 GB | Excellent | Low (Json, tsv, xml, etc.) | Every day |
| Crawled data | ≈ 1000 GB | Bad | High | Every day |

scheme with the certainty score. We manually assign the certainty score for every data source or predicate taking into account its update frequency or estimated precision.

**Attribute Converter** converts the types and predicates of the input data to our ontology by using the mapping file. For example, we provide a mapping from the person-type in Wikidata such as `https://www.wikidata.org/wiki/Q215627` to the PERSON type in our ontology. We traverse the type hierarchy of each LOD and semi-automatically construct type mappings from LODs to the JKB. We show the semi-automatical steps of constructing person-mappings from the Wikidata to the JKB as follows:

1. We judge whether the mapping from the type in Wikidata (https://www.wikidata.org/wiki/Q215627) to the type in our ontology is correct.
2. If the previous step passes, we automatically collect candidate mappings between all subclasses of the type in Wikidata and the type in our ontology.
3. We sample the mapping results and manually check the correctness of the above candidate mappings.

**Entity Matcher** outputs *entity clusters*, which groups the same entities. The Entity Matcher is mainly composed of three steps, rule-based matchings, graph-based matchings and filtering unnecessary entity clusters and their attributes. First, accurate matchings are conducted by rule-based matchings. Second, we create blocks of candidate entity clusters with weak matching methods (e.g., name matching) to reduce the computation time. We connect edges between related entity clusters of each block with more strict matching methods (e.g., name and birthdate) than blocking matching and extract cliques from each entity cluster to entity cluster graph. Third, the Entity Matcher removes entity clusters whose attribute certainties are zero and unmapped attributes at the Attribute Converter.

**ID Assigner** assigns a unique ID to each entity cluster based on the set of data sources of entities in the cluster. For example, if the ID of an entity cluster composed of two entities derived from "Wikidata ID: 100" and "DBpedia ID: 2000" is "JKB ID: 300", we save two relations as follows: "Wikidata ID: 100" → "JKB ID: 300" and "DBpedia ID: 2000" → "JKB ID: 300".

We preserve the relations between the ID and the set of data sources on Apache HBase (https://hbase.apache.org/), which is the Hadoop database, as the **ID Table**. The ID Assigner ensures that the same entity clusters compared to the past ones inherit to the past ID in the following steps:

1. Get the past JKB IDs of the entities in the given entity cluster by cross-referencing data sources of the entities in the entity cluster and the past ID Table. The past JKB IDs are candidate IDs for the given entity custer.
2. Get all past data sources by cross-referencing the past JKB IDs and the past ID Table.
3. Calculate the ratio of the number of the intersections of past and current data sources to the number of past data sources.
4. If the ratio is greater than 0.5, assign the past ID to the entity cluster; otherwise, assign a new ID and update the ID Table.

The ID is persistent across the time with the ID Assigner.

**Additional Data Combiner** combines entities from the additional data whose scheme is the same as the JKB's; thus, the Additional Data Combiner incorporates these entities into entity clusters from the ID Assigner by using their IDs.

**Entity Merger** merges entities in each entity cluster into one entity. We can define the certainty score to each Triple derived from the data source. Entity Merger merges the same Triples of entities in the entity cluster into one Triple whose certainty score is merged by a monotone increasing function. Since the main objective of our KB construction system is to maintain high precision at all time, the Entity Merger is a simple but powerful component because the Validator can filter unreliable attributes accurately due to the certainty score.

**Object Converter** converts the object of each Triple to the corresponding entity-id by referring the ID Table. If the Triple is derived from linked data and the object is described as the specific identifier, it is easy for the Object Converter to convert the object to the corresponding entity-id by using the ID Table. Since many objects of triples are represented as literals, linking such literal to the corresponding entities is similar to entity-disambiguation problems.

Therefore, we should resolve the entity ambiguity from two viewpoints, (1) the type consistency between the predicate and our ontology and (2) distinguishing different objects with the same name and the same type, such as a person's name. We solve the first problem by comparing our ontology with the range type of the predicate and the second problem by converting the object only when the entity whose pair of the object-type and the object-name is uniquely defined in the JKB.

**Attribute Completer** completes attributes to entities based on our ontology and their URL attributes. First, it completes attributes by using a symmetric property defined in our ontology such as the inverseOf property of owl:inverseOf (https://www.w3.org/TR/owl-ref/#inverseOf-def). Second, it extracts useful information from the entity-related URL, for example, OGP (http://ogp.me/) im-

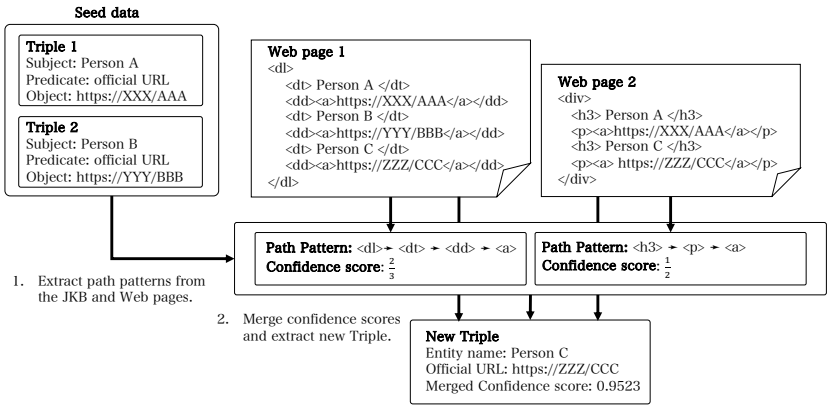


**Fig. 2.** Overview of Information Extractor of Our KB Construction System.

ages are useful attributes of these entities. The Attribute Completer partially addresses a well-known challenge; knowledge base completion [7] by reasoning missing inversed Triples.

**Validator** removes invalid data based on (1) blacklists created via the Manual Refiner, (2) inconsistency between Triples and our ontology, and (3) the results of fact checking by using crawl data (e.g., URLs are deadlink or not). It also rewrites values to standardize definitions of the JKB, for example, the phonetic characters are unified to hiragana (one of the writing systems of Japanese). We describe the details of the component in Section 3.3.

**Exporter** filters and corrects Triples to avoid service-specific issue, such as copyright problems, and outputs the JKB to a well-known format (e.g., JSON or N-Triples).

**Information Extractor** extracts factual information from a large set of Web-crawled data. Figure 2 illustrates an overview of the method of the Information Extractor (IE). There has been extensive work on IE from Web data [3]. We regard each Web page as a Document Object Model (DOM) Tree and collect all DOM path patterns related to a predicate by using the JKB.

The Information Extractor uses simple DOM-based method for extracting information from semi-structured data based on distant supervision [6]. First, we find path patterns from the subject to the object with confidence scores. Path patterns are extracted from DOM trees of a large amount of Web-crawled data and the JKB. Second, we extract new information from the extracted path patterns and merge the confidence score with our defined function in the following.

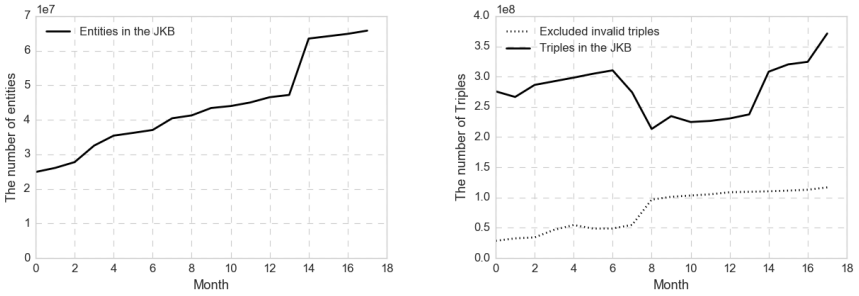


**Fig. 3.** Left is the number of entities in the JKB and right is histories of the number of Triple in the JKB and excluded invalid Triple data over the past 17 months

Given two confidence scores $P_A$ and $P_B$, we define the new associative binary relation between these scores as follows:

$$P_A \oplus P_B = \frac{P_A P_B}{P_A P_B + \alpha(1 - P_A)(1 - P_B)},$$

where $\alpha$ is a hyperparameter for the probability that two paths from two other Web pages are the same and the extracted fact is incorrect. We set, in practice, the hyperparameter $\alpha$ to 0.1 by the preliminary experimental results. The above binary relation satisfies an associative law.

**Manual Refiner** can handle corner cases that are difficult to remove or refine automatically. First, we find the incorrect facts from user feedback and our quantitative evaluation. Second, we examine if these facts are derived from a business requirement or rare cases or not. If these facts are corner cases and required immediate modification, we add them to the blacklist. Third, we create a new Triple with which we cannot import data due to the lack of information, such as images, without any additional information. The quality of JKB improves with the Manual Refiner.

## 3 Quantitative Results

### 3.1 Overall JKB Results

Figure 3 shows the graphs of the number of entities and Triples of the JKB over the past 17 months. The JKB steadily increased the number of entities. We took in a large CP data on the 4 months ago; thus, the graph in Figure 3 shows the sharp growth at that time. Since the quality of validation and filtering methods has been increasing, the number of excluded invalid Triples have risen continuously.

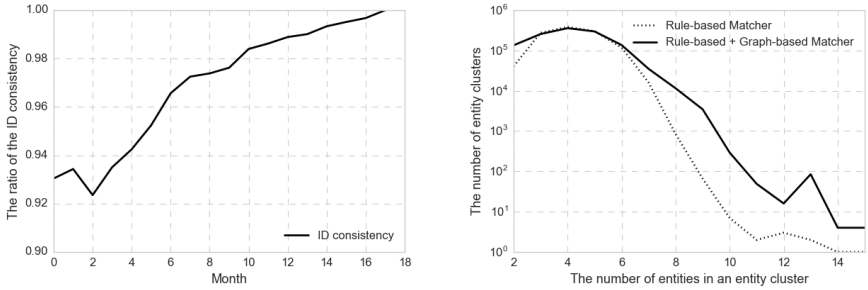


**Fig. 4.** Left is histories of id-consistency ratio between the current and the past JKBs. Right is the number of matched entities after rule-based matcher and graph-based matcher.

### 3.2 Identifiability Results

We first compared two JKBs separated by a week and confirmed that only 0.0004% of entities changed their IDs within a week. We also observed that more than 94% of the entities did not change their IDs as shown in Figure 4. Since we stopped importing some data sources and the Validator filters more and more invalid entities, some entities have deleted from the JKB. That is the reason why there are about 6% of entity-IDs are inconsistent to the current entity-IDs.

Second, we show the number of matched entities. The Entity Matcher uses two algorithms as follows; (1) Rule-based matching matches entities whose Wikipedia URL or some identifiers, such as IMDb ID and ISBN. (2) Graph-based matching matches entities with their types, names, and reliable attributes (e.g., birthdate or coordinates). We show the number of matched entities in Figure 4. Due to the graph-based matching, the number of matched entities increases. Graph-based matching does not match entities derived from the same data sources to improve the precision. For this reason, the precision of matching results is about 99%.

### 3.3 Automatical Validation and Completion Results

The Validator automatically removes many invalid Triples to maintain the quality of the JKB, as shown in the right side of Figure 3. The Validator filters (1) facts whose domain types are inconsistent with the type of objects, (2) facts that are functional (https://www.w3.org/TR/owl2-syntax/#Functional_Data_Properties), (3) facts whose data types do not match objects (e.g., if the data type is URL, the value must start with "http"), and (4) facts whose values do not satisfy the data type format (e.g., date or ISBN). We observed that 97.7, 2.0, 0.2, 0.1% of all validation data are (1), (2), (3), and (4), respectively. Note that a large amount of the invalid data of (1) is mainly derived from unmapped entities from the LOD to our ontology; thus, we can reduce the number of filtering entities, and this validation is one of the reasons that the JKB maintains high accuracy.

The Attribute Completer completes about 1.4% of all facts to the JKB.

## 4 Conclusion

We presented our plugin-based KB construction system that constructs scalable and production-level KB. We described an overview of our plugin-based system architecture and each software component. Our plugin-based KB system allows to implement various entity-matching and validation algorithms and also satisfy business requirements, while the system focuses on ensuring our SLAs and scaling computation. Our constructed knowledge base, JKB, is one of the largest KBs in Japanese and already utilized in Japanese Web services.

## References

1. Bellare, K., Curino, C., Machanavajihala, A., Mika, P., Rahurkar, M., Sane, A.: WOO: A Scalable and Multi-tenant Platform for Continuous Knowledge Base Synthesis. VLDB '13, vol. 6, pp. 1114–1125 (Aug 2013)
2. Blanco, R., Cambazoglu, B.B., Mika, P., Torzec, N.: Entity Recommendations in Web Search. pp. 33–48. ISWC '13 (2013)
3. Chang, C.H., Kayed, M., Girgis, M.R., Shaalan, K.F.: A Survey of Web Information Extraction Systems. vol. 18, pp. 1411–1428 (2006)
4. Choudhury, S., Agarwal, K., Purohit, S., Zhang, B., Pirrung, M., Smith, W., Thomas, M.: NOUS: Construction and Querying of Dynamic Knowledge Graphs. pp. 1563–1565. ICDM '17 (2017)
5. G.C., P.S., Sun, C., K., K.G., Zhang, H., Yang, F., Rampalli, N., Prasad, S., Arcaute, E., Krishnan, G., Deep, R., Raghavendra, V., Doan, A.: Why Big Data Industrial Systems Need Rules and What We Can Do About It. pp. 265–276. SIGMOD '15 (2015)
6. Mintz, M., Bills, S., Snow, R., Jurafsky, D.: Distant Supervision for Relation Extraction Without Labeled Data. pp. 1003–1011. ACL '09 (2009)
7. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning With Neural Tensor Networks for Knowledge Base Completion. NIPS '13
8. Suchanek, F.M., Kasneci, G., Weikum, G.: YAGO: a core of semantic knowledge unifying WordNet and Wikipedia. pp. 697–706. WWW '07 (2007)
9. Wu, S., Hsiao, L., Cheng, X., Hancock, B., Rekatsinas, T., Levis, P., R, C.: Fonduer: Knowledge Base Construction from Richly Formatted Data. SIGMOD '18 (2018)