Natural Language Question/Answering with User Interaction over a Knowledge Base

Huayi Zhan
Xi'an Jiaotong University
huayizhan2012@stu.xjtu.edu.cn

Baivab Sinha
Changhong AI Research
baivabsinha@changhong.com

Wei Jiang
Changhong AI Research
wayne_jiang@sina.com

ABSTRACT

In the demo, we present RecipeFinder, a system for searching the information from knowledge graphs with natural language. The sys-tem has following characteristics: (1) It supports human-computer interaction, to resolve question ambiguity; (2) It provides graphical interface to help users refine questions

CCS CONCEPTS

• **Computing methodologies ~ Artificial intelligence** • Computing methodologies ~ Natural language processing • Computing methodologies ~ Language resources

KEYWORDS

Knowledge Graph, Natural Language Processing, User Interaction

1 Introduction

Knowledge graph have received tremendous attention as more and more real life structured data are available and maintained. However, with the volume of data present and complex methods developed to retrieve the information, the task of developing a user-friendly method for the casual users is challenging. Keyword-based search engines (e.g., Google and Bing) have gained a huge success. Nevertheless, keyword search may not be expressive enough to query structured data since only a few words are used to specify the query intention. In comparison, structured query languages, e.g., SPARQL [4], and GRAPHQL [3], are quite expressive but too complicated for casual end users [10]. Moreover, if the user is not aware of the graph schema, the construction the structured query will prove to be further difficult. Natural language question and answering interface provides a promising approach to accessing knowledge graphs. It offers users a familiar and intuitive way, rather than complicated structured query languages, to describe the query [9] [1] [7].However, it needs to bridge the gap between unstructured natural language questions and structured knowledge graphs. In other words, it is required to understand the natural language question and generate the corresponding structured query, which can fit the knowledge graph. Nevertheless, due to the linguistic variability and ambiguity [6], it is extremely difficult for a computer to understand some questions precisely. Let us consider an example as follows.

AICS 2019, July 12–13, 2019, Wuhan, Hubei, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7150-6/19/07...$15.00
https://doi.org/10.1145/3349341.3349425

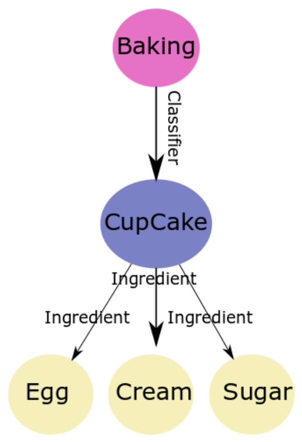


Figure 1: Example Graph

Example 1.1. Consider a fraction of a collaboration network depicted as graph G in Figure 1. There are three different kinds of nodes in the graph G denoted by 3 different colors. The node in Blue, denoted by Nr represents the recipe name with attributes like name of the recipe, its alias or other common name, related information about the recipe regarding its introduction and usefulness, main ingredient list, supplementary ingredient list, directions or steps involved to cook the dish, etc. The node in Pink, denoted by Nc , represents the classifiers used to classify different recipe which is a combination of attributes like what kind of dish it is (breakfast, meal, vegetarian, drinks, cold, hot etc.), different cuisines (like Italian, Chinese, Spanish etc.) and other classifiers (like specialty dish for a festival, seasonal dish etc). The node in yellow, denoted by Ni, represents the common ingredients used to prepare the dishes like (pork, sugar, salt etc.)

Each of the Node Nr is linked with Nc using the edge type E1, representing the classifiers associated with a particular recipe. Furthermore, the Node Nr is linked with Ni using the edge type E2, representing the ingredients used in the preparation of the recipe.

Suppose that a user the following natural language question: “Suggest me a baking recipe that uses eggs and cream”. This recipe can be classify as a “baking dish” and includes “eggs” and “cream” as main ingredients. The requirements are expressed as a bounded simulation query Q[1] (Figure 1) as follows: (1) the recipe should contain eggs and cream as the main ingredients which mean there should be an Edge of type E2(RI) joining Node Nr(R) to two different nodes Ni denoting cream Ni(C) and eggs Ni(E); (2) the Node Nr (R) should be connected to a Nc node denoting Baking Nc (B) using an edge type E1(RC) as represented in Figure 1.

However, in the Graph G, there might be many Nodes Nr(R) that satisfies this query Q. Which mean that there can be multiple nodes satisfying 1, 2 and 3 :

$$E1\ (RI) \in E1\ (Nr\ (R), Ni\ (E)) \quad (1)$$

$$E1\ (RI) \in E1\ (Nr\ (R), Ni\ (C)) \quad (2)$$

$$E2\ (RC) \in E2\ (Nr\ (R), NC\ (B)) \quad (3)$$

These multiple nodes are the result of ambiguous question that may result in multiple answer based on the interpretation of the system. In our daily communication, in case one person is not sure of the meaning of a question, she will try to make it clear by asking some questions back to the asker [10] [2]. In light of this natural interactive process [6], we present a Recipefinder, a novel system to efficiently recommend the recipe for the user using the natural language question and then after remove the ambiguity of the request by asking more question back to user without using any Natural Language processing tools [10] [5]. To keep the interaction engaging, we limit the number of the questions to 3. The basic principle is to understand the question and generate the structured query through the talking between the data (i.e., the knowledge graph) and the user and then using these queries over the available knowledge base to retrieve the answer [8]. It benefits are three folds: Since the user knows exactly what she wants, letting the user verify ambiguities can achieve better accuracy; (2) The knowledge graph specifies the structural relations among the mapping objects (entities/concepts/predicates) of the phrases in the question, which can help the query formulation; (3) Since the final query is more precise, the overall system is less resource hungry and more cost efficient. To the best of our knowledge, Recipefinder is among the first efforts to identify the recipe based on the natural language question provided by the user and remove the ambiguity of the question based on asking questions back to the user. It should also be remarked that recipe searching is just one application of this technique; one may apply the technique to find, for example, movies, places, persons etc.

2 System Architecture

The architecture of the Recipefinder system is shown in Fig. 2 It consists four modules. (1) A Graphical User Interface (GUI) provides a graphical interface to help users formulate queries, interact with the user and show the query result to the user. (2) An Entity extractor module, which extract the entities from the natural language query provided by the user. (3) A Query refinement module which fine tune the user query by asking questions back to them. (4) The result module which uses the final refined query to interact with the knowledge base and provide the final result to the user.

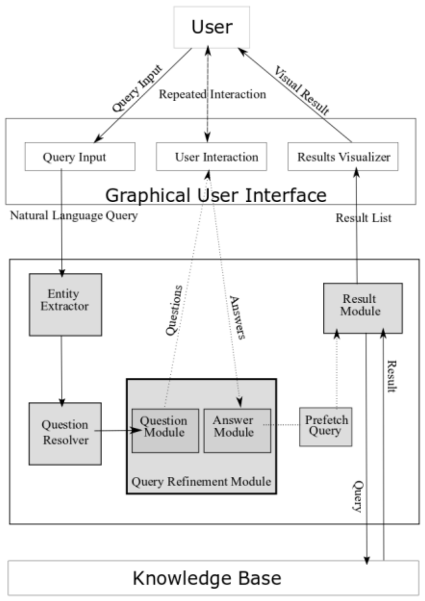


Figure 2: System Architecture

Entity Extractor: We built a corpus of known named entities collated from various sources. Later we designed a Chinese text segmentation module based on it. This Chinese text segmentation module attempts to cut the sentence into the most accurate segmentation, which is suitable for text analysis. From these segments, we try to extract the noun and verbs from the sentences for identifying the intent of the user question and the keywords. This entity extractor is based on a prefix dictionary structure to achieve efficient word graph scanning which built a directed acyclic graph (DAG) for all possible word combinations. It also uses dynamic programming to find the most probable combination based on the word frequency. In the run-time we might come across some unknown words, for them, a HMM-based model is used with the Viterbi algorithm.

Example 2.1. Continuing from the earlier example, when the user asks the question: Suggest me a baking recipe, which uses eggs and creams, the natural language query, goes to the entity extractor and the entity extractor returns baking, eggs and cream

Knowledge Base: The knowledge base of the system contains the information of the different entities, which we want to search along with their important attributes. As for example, the knowledge base of the demo system, RecipeFinder, consists the records of 338192 recipe in total. The record includes the recipe name with attributes like name of the recipe, its alias or other common name, related information about the recipe regarding its introduction and usefulness, main ingredient list, supplementary ingredient list, directions or steps involved to cook the dish, etc.

Question Resolver: The question resolver determines which sets of questions to be asked from the users. We have defined a set of entities and relationship beforehand and stored in a look up table. These set of entities-relationship will vary will different search system. For instance, in RecipeFinderdemo, we have defined 3 kinds of relationship named 1) The relationship classifier consisting the food preparation techniques (like fried, baking steamed etc.), type of cuisine (Chinese, Italian, Spanish), specialty food (like special food for festivals, special food a region, etc.), variety of food (hot food, cold food, drinks etc.), timing of the meal (main meal, breakfast, dessert etc.), etc. 2) The relationship recipe which consists of names of culinary dishes from the knowledge base. 3) The relationship ingredients, which consists of the names of common ingredients used for preparations of the culinary dishes from the knowledge base. The records of this entity-relation table are used by question definition algorithm for formulating the questions to be asked from the users. Based on the keywords extracted from the entity extractor, appropriate relationship is selected and the types of question set is defined.

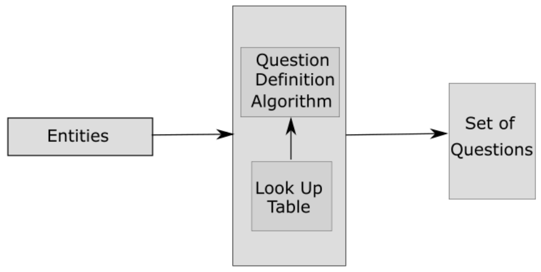


Figure 3: Query Refinement Module

Example 2.2. From the continuing example, the Question resolver to take baking and eggs and cream as input and will search in the look up table for the relationship match. In the above example, there will be a match M (baking) as which represent the cooking method (denoted by classifier) and two other relation matches M (eggs) and M (cream) as ingredients. So the relationship set R obtained, consists of

$$R = \{\{r1 \in C\}, \{r2 \in I\}, \{r3 \in I\}\} \quad (4)$$

where C represent the set of classifier and I represent the set of ingredients and ri the relation matches obtained from the look up table as shown in 4

Based on these information, the question resolver module will define the sets of the questions to be asked by the user which in this case are 1) what is the taste of the dish 2) the level of the difficulty in making this dish 3) how much is the preparation time, as shown in fig3 In some other cases if the relationship set R consists of

$$R = \{\{r1 \in I\}, \{r2 \in I\}\} \quad (5)$$

where C represent the set of classifier and I represent the set of ingredients and ri the relation matches obtained from the look up table as shown in 5.

Based on 5 the question resolver will define the sets of the questions as 1) what is the taste of the dish 2) what are the classifier for the recipe 3) list of possible recipes satisfying condition 1 and 2. Similarly for other sets of relations, related sets of questioned is defined by the Question Definition Algorithm.

Query refinement Module: The Query refinement module (QRF), as shown in Figure 4 use to interact with the user by asking the questions to the user. This interaction is used for removing the ambiguity of the question The Query refinement module consists of two part. 1) The Question module, which is used to ask the question to the user, and 2) The Answer module, which is used to store the reply of the user. The replies from the user is used to formalize the query to reduce the interaction cost, and design an efficient strategy to solve the problem. In this module, we also use a query prefetching technique by exploiting the latency in the interactions with users using the replies at each interaction stage captured by the Answer module. The interaction is useful for refining the question of the user but clearly, it will degrade the user experience if there are too many rounds of interactions, so to keep the Interaction with the users engaging, we try to limit the interaction to a maximum of three.

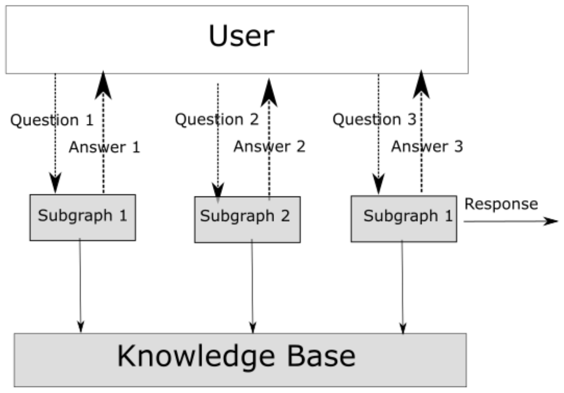


Figure 4: Result Module

Example 2.3. From the continuing example, the QRF uses the entity relation lookup table and decides on following three questions to ask for the user: 1) What is the taste of the recipe? 2) What is the level of the difficulty in this recipe preparing? 3) How much time this recipe preparation should take?

These questions are asked one by one using the question module. These are multiple-choice question where the user is shown some possible answers in a drop down list. The answer module will record the User answer (item from this drop down list). With each round of the interaction, the user intent will become clearer. The final query is more precise, the overall system is less resource hungry, and more cost efficient.

Result Module: The result module uses the query obtained from Query refinement module after rounds of interactions with the user. This resultant query is used for the answer generation from the knowledge base. Since at this stage, we already refined the user question by interacting with her, the total cost of the evaluation comes down drastically. Once the answer is generated, it is shown to user using the Graphical user Interface.

As shown in Figure 4, once the Query refinement module receives the answer from the user, it uses the question, answer and the knowledge base to create resultant data sets. Naturally, with each interaction, the resultant sub graph will become smaller and close to the intended answer of the user. In the first iteration, we will receive a sub graph D1 where D1 is the set of all the results of graph G, a pattern query Q and attribute A, M (Q, A, G) is the relation such All the nodes v ∈ KB there exist an edge E (v, u) for each node ∈ Q having an attribute A So it can define as

$$D1 \in M\ (Q1, A1, KB) \tag{6}$$

Now for the next iteration, the result of 8 will serve as the graph and new pattern query Q2 with attribute is used. Resulting

$$D2 \in M\ (Q2, A2, D1) \tag{7}$$

Similarly, for the final iteration,

$$D3 \in M\ (Q3, A3, D2) \tag{8}$$

These sub graphs are created to utilize the latency in user's reply. Since the matches involves the answer from the user, we can refine the query match at each stage. The final match obtained from 8 will be sent to the graphical user Interface.

Example 2.4. In the continuing example, following will be the interaction with the user:

Q1: What is the taste of the recipe?
A1: Sweet
Q2: What is the difficulty level of the recipe preparation?
A2: Ordinary
Q3: How much time the recipe will take?
A3: 1 hour

These sets of questions and answers will be used for generating the response from the knowledge base. Based on the records of knowledge base, we can say that these represent the attributes of the recipe Cup cake. Therefore, after a few interactions from the user, we have narrowed down the matches of Query Q.

Graphical User Interface: The Graphical User Interface(GUI) helps the users to input the natural language query, interact with the system and browse the final query results though 3 different web page constructed on PHP, Node.js and HTML5. This graphical User interface consists of (1) Query Input Page; which accepts the natural language query from the user and sends to the entity extraction module. (2) User Interaction Page; which performs the interaction between user and the system by asking questions to user and storing the answers provided by the user using the query refinement module. (3) Result Visualizer Page; which uses the results obtained from the result module and show the recipe name, its alias or other common name, type of the recipe, related information about the recipe regarding its introduction and usefulness, main ingredient list, supplementary ingredient list, directions or steps involved to prepare the recipe, time taken to prepare the recipe, difficulty level of the recipe etc.

3 Demonstation Overview

The demonstration is to show the following: (1) The use Query Input Page of GUI to formulate the query where the user inputs the natural language query to find a recipe. (2) The use of User Interaction Page for GUI for the interaction with the user. (3) The use of Result Visualizer Page of GUI, which is used for showing the result to the user.

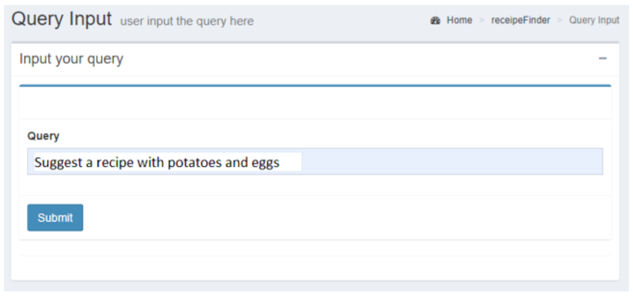

Query Input user input the query here
Home > receipeFinder > Query Input
Input your query
Query
Submit
Suggest a recipe with potatoes and eggs

Figure 5: Query Input Page

Setup: To demonstrate the effectiveness and performance of this RecipeFinder, we used an in-house made recipe knowledge base which consists the records of 338192 recipes in total gathered from the open web. The record includes the recipe name with attributes like name of the recipe, its alias or other common name, related information about the recipe regarding its introduction and useful-ness, main ingredient list, supplementary ingredient list, directions or steps involved to cook the dish, etc. The system is implemented on Python and PHP, deployed on a virtual machine with 3.30 GHz CPU and 8GB Memory.

Interacting with the GUI: We invite users to use the GUI, from natural language question construction to intuitive illustration of query results. (1) Query Input Page to formulate the query where the user inputs the natural language query to find a recipe, as shown in Fig 5, where user inputs the natural language query in Chinese language like please suggest me a recipe that contains potatoes and eggs. (2) The Interaction page where the knowledge base and user interacts, using question and answers, resulting in a better understanding of the user intent. Like as shown in Figure 6, the system asks the user to refine the query by asking her questions like how she wants the dish to be classified, what is the taste of the recipe and in the end selecting the dish name.

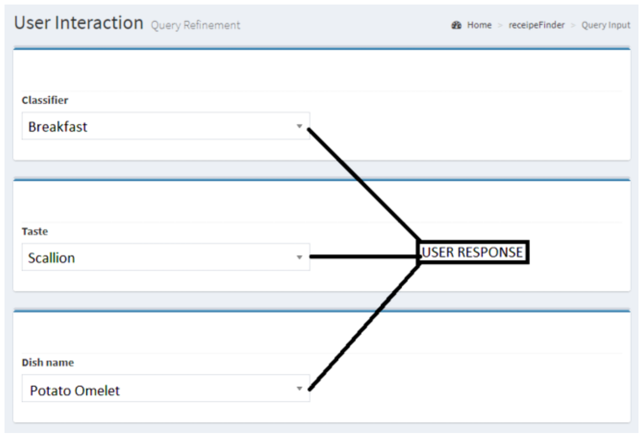
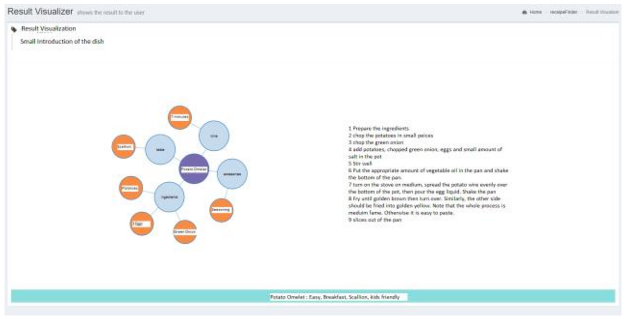

User Interaction Query Refinement
Home › receipeFinder › Query Input
Classifier
Breakfast
Taste
Scallion
Dish name
Potato Omelet
USER RESPONSE

Figure 6: User Interaction Page

The GUI provides intuitive ways to help users interpret query results. In particular, the GUI allows users to browse (a) all the final match w.r.t. Q, as an example, the query results of Q, given in Fig. 4, refined by the additional question as shown in fig 5 and showing the details of query response as shown in fig 6. The GUI shows the information about the recipe (like the taste, main ingredients used, supplementary ingredients user, time taken for preparing the dish), along with the steps to prepare the recipe. It also shows an introduction of the recipe and some of its category, as extracted from different web sources.


Result Visualizer shows the result to the user
Home › receipeFinder › Result Visualizer
Result Visualization
Small introduction of the dish
Potato Omelet
taste
time
accessories
ingredients
Scallion
Potatoes
Eggs
Green Onion
Seasoning

1 Prepare the ingredients
2 chop the potatoes in small peices
3 chop the green onion
4 add potatoes, chopped green onion, eggs and small amount of salt in the pot
5 Stir well
6 Put the appropriate amount of vegetable oil in the pan and shake the bottom of the pan.
7 turn on the stove on medium, spread the potato wire evenly over the bottom of the pot, then pour the egg liquid. Shake the pan
8 Fry until golden brown then turn over. Similarly, the other side should be fried into golden yellow. Note that the whole process is meduim fame. Otherwise it is easy to paste.
9 slices out of the pan

Potato Omelet : Easy, Breakfast, Scallion, kids friendly

Figure 7: Result Visualizer Page

Performance: The main performance overheads of the system is in the entity extraction, query formation and query response. For a better user's experience, we load the entity extraction dictionary beforehand, therefore, the once the system is initiated the entity extraction works in real time for entity extraction. Owing to the large number of record in the knowledge base, the initial query formation takes some time (about a few seconds) but thereafter as the query is refined with every step of user interaction, the system works in near real time. It is expected that the performance of the system will be better once it is hosted on a dedicated server with better resources.

4 Summary

This demonstration aims to show the key ideas and performance a natural language Question-Answering system that uses user Interaction to refine the query. The recipe search system, RecipeFinder, as demonstrated here is one of the applications of this technique. The system is able to (1) Understand the natural language query of a user (2) Efficiently formulate and interact with user and knowledge base to remove ambiguities in the question (3) Compute matches with query prefetch and temporary table-based technique to utilize the interaction latency; (4) Facilitate users request with an intuitive graphical interface. These together convince us that the RecipeFinder can serve as a promising tool for recipe search in real-life scenario and the mentioned technique of natural interaction to understand users query better can be extended to other system (for example, for searching movies, places, persons etc.) as well.

REFERENCES

[1] Shenghui Cheng, Wei Xu, and Klaus Mueller. 2019. ColorMapND: A Data-Driven Approach and Tool for Mapping Multivariate Data to Color. IEEE Trans. Vis. Comput. Graph. 25, 2 (2019), 1361–1377.
[2] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open question answering over curated and extracted knowledge bases. In The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014. 1156–1165.
[3] Huahai He and Ambuj K. Singh. 2008. Graphs-at-a-time: query language and access methods for graph databases. In Proceedings of the ACM SIGMOD Interna-tional Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008. 405–418.
[4] Frederik Hogenboom, Viorel Milea, Flavius Frasincar, and Uzay Kaymak. 2010. RDF-GL: A SPARQL-Based Graphical Query Language for RDF. In Emergent Web Intelligence: Advanced Information Retrieval. 87–116. https://doi.org/10.1007/978-1-84996-074-8_4
[5] Nandish Jayaram, Mahesh Gupta, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri. 2014. GQBE: Querying knowledge graphs by example entity tuples. In IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014. 1250–1253.
[6] Esther Kaufmann and Abraham Bernstein. 2010. Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. J. Web Semant. 8, 4 (2010), 377–393.
[7] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. PVLDB 8, 1 (2014), 73–84.
[8] Robert Pienta, Acar Tamersoy, Hanghang Tong, Alex Endert, and Duen Horng (Polo) Chau. 2015. Interactive Querying over Large Network Data: Scalabil-ity, Visualization, and Interaction Design. In Proceedings of the 20th International Conference on Intelligent User Interfaces Companion, IUI 2015, Atlanta, GA, USA, March 29 - April 01, 2015. 61–64.
[9] Tony Russell-Rose, Jon Chamberlain, and Farhad Shokraneh. 2019. A Visual Approach to Query Formulation for Systematic Search. In Proceedings of the 2019 Conference on Human Information Interaction and Retrieval, CHIIR 2019, Glasgow, Scotland, UK, March 10-14, 2019. 379–383.
[10] Weiguo Zheng, Hong Cheng, Lei Zou, Jeffrey Xu Yu, and Kangfei Zhao. 2017. Nat-ural Language Question/Answering: Let Users Talk with The Knowledge Graph. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017.