

# An Efficient Search for Context-Based Chatbots

Ayah Atiyah\*, Shaidah Jusoh†, Sufyan Almajali‡

King Hussein School of Computing Sciences, Princess Sumaya University for Technology  
Amman, Jordan

Email: \*aya20168009@std.psut.edu.jo, †s.ibrahim@psut.edu.jo, ‡s.almajali@psut.edu.jo

**Abstract**—A chatbot is one of the Artificial Intelligence applications which has a capability of making human users to feel real and understood in real time. Using chatbot in any business process may increase customers' satisfaction as it is available 24/7 to entertain customers' needs. A chatbot is expected to be more efficient than a human personnel in responding to customers' questions. Thus, having a chatbot that is able to retrieve answers for all questions in a quick and efficient way is very desirable. The aim of this paper is to introduce a new method called B-Point Tree to increase the efficiency of searching for accurate answers by adding an additional data structure to the traditional BST algorithm. The performance of the B-Point Tree has been compared to the traditional BST, results of experiments suggest that B-Point Tree outperforms the traditional BST. The findings suggest that the B-Point Tree is able to enhance the efficiency of a chatbot.

**Index Terms**—Conversational Agents, chatbots, Dialog Systems

## I. INTRODUCTION

Despite the complexity in handling natural language, applications which are based on the natural language have been seen as emerging technologies for current and future businesses [1]. Of natural language applications, a chatbot system which is based on natural language dialog interface is expected to give a great contribution to online businesses. The term chatbot and intelligent conversational agent are often used interchangeably in many works of literature. Whether it is a chatbot or a conversational agent, it is an application software that allows a conversation to occur between a human user and a computer device. The concept of conversation is central to the chatbot in spite of its limited capabilities. However, when the context of use is narrowed down, chatbots can have advantages over human agents. They can outperform humans in terms of speed and accuracy in a narrow domain.

The MIT technologies lists [2] stated that the chatbot is one of the ten revolution technologies in 2016. According to Satya Nadella, Microsoft CEO [3], the conversational interfaces are the next big thing along with the touch screens, web browsers, and graphical user interfaces [3]. William Meisel [4], an esteemed analyst in the technology world claimed that some limited-focused-goal conversational agents such as Apple's Siri will achieve profits near \$632 billion by 2020 [4]. In April 2016, Mark Zuckerberg, Facebook CEO, stated that chatbots are the key solution for application overhead problems [5]. The need for conversational agents has increased especially in the emerge of natural language processing [6].

The main benefit of developing and deploying a chatbot in any business domain is that it can behave as a personal

assistant which is available 24/7. This is very important in the business world due to the time and locations differences across countries [6]. By including the chatbot as a part of the business process, the business companies are able to entertain their customers without concerning borders of regions. Customers have benefits of having instant answers to their queries. There have been a number of business domains which had applied chatbots in their business processes. These include banking, marketing, finance, computer services, human resources, games, and entertainment [7].

Vendors need their customers to be loyal and to keep buying their products. One factor which can contribute to customers' satisfaction is the availability of a human personnel [8], [9], [10]. Unfortunately, it is difficult to keep a human personnel to be available all the time. The deployment of the chatbot in the business process may replace a human personnel and this can keep customer satisfaction [11], [12]. The chatbot allows customers to interact with vendors and express their opinions freely [7], [13]. Consequently, an active engagement between the customers and vendors can be achieved. It is no doubt that the chatbots may give a significant impact to the business world [7].

Technically, chatbots are intelligent machine-to-human conversation systems [14], [15]. The human-computer dialog is achieved by either text or speech. The use of text dialog may require natural language understanding techniques, while the latter requires speech recognition. In the former process, the main task of the chatbot is to understand input text and reply with the best answer to the human user. Generally, chatbots can be classified into two categories: independent chatbots and web-based chatbots. The independent chatbots are used in a stand-alone computer with no need to access web, while the Web-based chatbots [16] are accessed via web services and can be built on the cloud.

Parameters which can affect the chatbot dialog system in a positive or negative way depend on the use and behavior of the systems. These include the procedures that are used to develop and process the text using the grammar rules, the mechanisms that are used by the chatbot in order to match the patterns with every single word, and how the systems responses to the questions [17], [14]. Some studies [18], [19], [20] show that business entities which deployed chatbots in their business processes have increased their revenue over the business without chatbots. This is due to the flexibility and independence in the customer opinion and product choice. [21]. Despite the chatbots developed so far, developing a chatbot that is able to understand, interpret, and respond to

user's dialogs in an efficient and fast manner is still a matter of research. The purpose of this paper is to propose a new algorithm for having a fast and accurate response to users' queries in a chatbot system. The proposed algorithm has been implemented on a chatbot called FloristBot. The FloristBot behaves as human personnel that works at a flower's shop to entertain customers and take orders of a flower bouquets for various occasions. This paper is organized as follows. Section II presents the related topics, and our proposed algorithm is described in Section III. The Floristbot implementation is presented in Section IV. Experiments and results are presented in Section V. The paper is concluded in Section VI.

## II. RELATED TOPICS

There are two main topics are involved in this work. These include Artificial Intelligence Markup Language (AIML) and Binary Search Tree (BST). AIML has become a popular language to create chatbots. AIML describes a class of data objects called AIML objects and partially describes the behavior of computer programs that process them. AIML objects are made up of units called *topics* and *categories*, which contain either parsed or unparsed data. AIML has a deterministic ability in which it can determine the natural language models [22]. AIML objects are considered as language tags in which they are connected to language commands. The basic part of the AIML is category. Every single category is combined together to build the overall structure of chatbots. This category can be further composed into three parts: input, output, and selected context as shown in the following tags structure [23].

```
<category>
  <pattern> Hello Alice </pattern>
  <template>
    Hello User
  </template>
</category>
```

A conversation in any chatbot consists of users' inputs and responses from the system. To handle the conversation, the chatbot is normally consists of a *user interface* and three main modules; *responder*, *classifier*, and *master-graph* [24]. The process of a conversation in the chatbot is two-way communication. The interface is responsible of receiving input from the users, responder is responsible of transferring the input to the *classifier*. The responder also performs some control functions on the input by making filtering normalization to the input. The classifier is responsible of dividing the input data into a logical chunk before it is transferred to the *mastergraph*. The mastergraph component is considered as the brain of the whole chatbot in which it performs the intelligent components operations, such as ordering and organizing the data contents in the system. Comparing and matching the structure of this language is also done by the *graphmaster*. Finally, the data is visualized in a graph structure [17], [23], [25]. Inputs from users are normally short sentences. In processing the inputs, sentences are tokenized into words, and each word is labeled with Part of Speech (POS). Once the POS has been attached to

each word, meaningful words based on predefined keywords are filtered out. The output is the response to the input in which it is concerned with the keywords processed in the chatbot engine [26], [27]. BST is a balanced binary tree whose nodes hold one item or more [28]. The BST contains left and right data pointers, for pointing to its subtrees. Individually, a leaf BST has no child pointer. Also, the data pointers in a BST point to the identical user input data in the memory, hence the corresponding search keys and their data entries can be accessed using the data pointers. Another significant data term is the balance factor, which is the difference between the height of left subtree and the height of right subtree. From this point of view, the BST is a balanced binary tree, the balance factor can be 1, 0, or -1 [29]. There will be a special element *Count* in each BST node that hold the number of key values in the node. Reference [30] claimed that BST is the most significant data structure in database and file systems. However, deletion of data entries may also cause rotating the tree to keep it balanced. The repeated rotations of the tree will lower system performance [31], [32].

## III. THE PROPOSED ALGORITHM

In this work, we propose a method for speeding up search in data retrieval, by adding additional data structure to the BST. The aim is to reduce the depth of the search tree in order to reduce the number of memory accesses. In the proposed method, each tree node holds a search key value, and each data record added will be inserted into the tree. Moreover, counters are placed into the tree node in order to read the frequency of duplicate group of the key contexts. As mentioned earlier, BST is a rooted binary tree, where each node stores a value and each has two distinguished subtrees, denoted left and right. Also, tree traversal costs  $\mathcal{O}(\log n)$  time, (where  $N$  is the total number of nodes). Therefore, our aim is to reduce the time complexity to  $\mathcal{O}(1)$  due to the decreased number of rotations, which to outperform the normal BST algorithm. We have modified the BST into B-Point Tree, which allows an extra data structure to be linked to a BST in order to accelerate the tree search operation. This data structure consists of a shortcut pointer (see Figure 1). B-Point Tree uses a sorted array that contains most active  $M$  points that are adjusted instantly in each query execution. In other words, arrays elements hold pointers to nodes with the maximum counters. Also, the array plays a shortcut role in the search operation. Compared to other search mechanisms in BST, the added benefit of B-Point Tree is that it requires less number of search traverses for each data input. In this work, the proposed method will be used to handle multiple data inputs for a chatbot dialog system and increase the performance efficiency. The proposed method is expected to reduce the time complexity to constant time that shows a huge performance improvement. In the traditional BST, each node holds a search key value which represents the matched Contexts for user data input, and has a counter to track the frequency of each group of Contexts. Most active points are continuously traced, and a tree traversal is made for all the tree nodes. Then an array is used to sort them

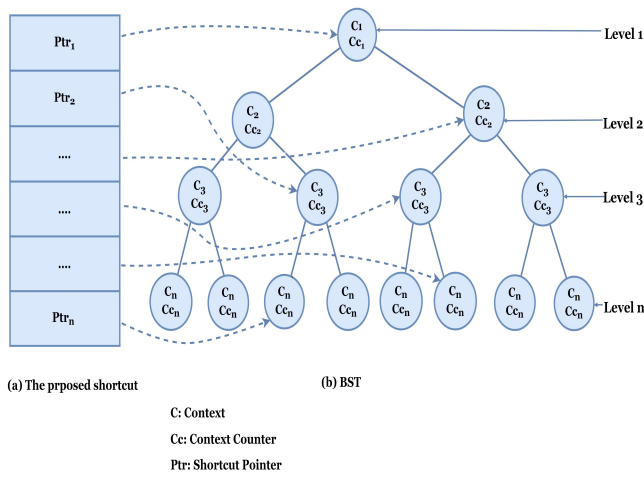


Fig. 1. The modified BST tree with Shortcut Pointer Structure

and read the counter constantly, the required data records are retrieved by the maximum values that are traced in the sorted array. This process increases the number of memory accesses, since specifics about how the tree is traversed are gathered, then, an array is created to insert and sort the tree nodes. *Insertion with Shortcut* module uses a sorted array that contains most active  $M$  points that are adjusted instantly on each data adjustment. By tracing each context, if the context is one of the array contexts, it increments its counter without any need to traverse all the nodes as in the case of the normal BST. If the context matches a nodes value, its counter will be immediately incremented, otherwise, it will be inserted into the tree as a new node. The proposed algorithm consists of 4 main modules, which are explained as follows:

#### A. Search

*Search* module is built on continuously adjusting the pointer's value. It searches for a value through the shortcut context which is the last updated context of the last inserted value into the shortcut array. If the value is found, it will increment its counter, then the pointer's value is checked if it is larger than the last one traced, to perform pointer swaps and keep the array sorted.

#### B. Update

*Update* module keeps tracing each duplicated context group added, if the new context group matches the last one traced by referring to its context, instead of traversing all the tree nodes, which is time consuming, *Update* module is used to modify the record counter. In this paper, Shortcut context is used to refer to non-replicated group.

#### C. Insertion with Shortcut

The main module of the algorithm is *Shortcut Insert*, where the shortcut context is introduced and checked if it is larger than 0, specifically, if the context group is replicated, *Update* procedure is called through the nodes context. Thus, this process escapes checking if each nodes counter is larger than

the minimum counter of the array and shifting nodes, which consecutively saves time. Otherwise, an additional check if the counter exceeds the minimum is made, to be inserted into the shortcut array. Otherwise, if it is a new node that has not previously inserted, the counter is set to 1 and the value is adjusted to match the last value. This operation needs rearranging between the BST nodes and the new node, and some movements around are needed in order to keep the tree balanced. The pseudo code of the insert with shortcut algorithm is shown below.  $C := Context$

#### Algorithm 1 Insert with a Shortcut Algorithm

```

1: procedure INSERTWITHSHORTCUT
2: Require:  $SC \leftarrow \{\}$  where  $SC$  is shortcut array
3: While New Context Group Inserted do
4:    $P \leftarrow NewInsertedcontextGroup$ 
5:   Read  $C$ 
6:   Compare  $C$  to previous context  $C_p$ 
7:   if  $C = C_p$  then
8:      $Node \leftarrow lastNode$ 
9:      $NodeIncrement \leftarrow NodeIncrement + 1$ 
10:    if  $Node \notin SC$  then
11:      if  $NodeIncrement \geq SCnodeMinIncrement$  then
12:         $SC \leftarrow Add\ Node$ 
13:         $NodeIncrement \leftarrow NodeIncrement + 1$ 
14:      end if
15:    end if
16:     $LastNode \leftarrow Node$ 
17:  else
18:    Do Normal BST Insert
19:  end if
20: end while

```

$C_p := PreviousContext$   
 $SC := Shortcut$

#### D. Environment Specification

1) *Search in Unique Contexts:* Unique contexts Identifier module takes an array of values (contexts) and the number of contexts that will be generated. Each new context generated is compared to the previously generated contexts stored in an array to identify unique context in the unique search identifier.

2) *Duplication:* Context Count is an element that is added to the BST node to count the number of search key values. To generate duplicate context groups, *Duplicate Groups* module is built with five parameters: the data array built in the other module (Unique contexts Identifier), the number of contexts, a new array to hold the number of context groups, the number of data groups and the ratio of duplicated contexts in all data records. In other words, if the ratio is 5% that means half of the data will be from the first 5% key contexts and 95% will be from the rest of contexts.

#### IV. SYSTEM IMPLEMENTATION

The FloristBot system has been implemented using API.AI tool [22], [33] and C++ programming language. The API.AI

tool was used to create the chat dialog interface while C++ programming language was used to implement the proposed algorithm. As mentioned earlier, the building blocks of our FloristBot are *Contexts* and *Intents*. The latter one represent a mapping between what a user says and what actions should be taken by FloristBot. Whereas the first one is built to differentiate the different intents that can result from a single word and choose the best one among them. The first step when the user input is to check for a matched pre-defined intent. If there is no matching intent, it can be handled by Dialogflow framework using a feature called Default Feedback for Intent. If it matches, a new list of contexts is introduced for that matched case. This combination of intents and contexts are built to form the complexity of the system. One disadvantage is considered for this Framework is that when there could be no matching resulted only from the lack of predefined contexts for a certain word. The FloristBot consists of the following *entities* and *intents*. An *entity* is a created object to represent a person, a place or a thing. For example, flower is an entity and it has many entries such as Tulip, Jasmine, and Roses. *Intents* are things that should be handled by the system. In our system there are many intents, such as *Compose*, which contains what the user says, such as “*I need a bouquet*”, “*I want to buy a bouquet*” or “*Yes*”. The matching is made between the key word and the suitable response. The intents are handled by Actions and Response. The action in this example is bouquet.compose and the response is “*Okay! Would you like to compose a bouquet yourself?*”. Default Fallback Intent is a default intent that handles any un-predefined context or unrecognized one. The response is “*I didn’t get that, can you say it again?*” or “*Sorry, can you say that again?*”. The user choice can be handled by *Define Prompt* feature which enables the system to respond by a default response when the user has no choice. If the user did not input the type that he is supposed to choose, a message appears. For instance, “*Sorry but you did not tell me the type of flowers you want to buy, what type of flowers do you want?*”. A sample of the list of user queries and system responses is shown below. **Default Welcome Intent:**

USER : Good Day!  
SYSTEM : Greetings!

#### **Default Fallback Intent:**

SYSTEM : I didn’t get that. Can you say it again?  
SYSTEM : I missed what you said. Say it again?  
SYSTEM : Sorry, can you say that again?  
SYSTEM : Can you say that again?  
SYSTEM : Sorry, I didn’t get that.  
SYSTEM : I missed that.  
SYSTEM : What was that?  
SYSTEM : Say that again, please?

#### **Greetings Intent:**

USER : Hello!  
USER : Hi!  
SYSTEM : Hello! How can I help you?  
SYSTEM : Hello! Would you like to buy a bouquet?

#### **Compose Intent:**

USER : I want to buy a bouquet.  
USER : I need a bouquet.  
SYSTEM : Okay! Would you like to compose a bouquet yourself?  
USER : Please let me compose them.  
USER : Yes, I want to compose them.

#### **No-Compose Intent:**

USER : No, usual bouquet is OK.  
USER : No, give me the usual bouquet.  
USER : Sure! I can offer you a bouquet for different occasions.  
For which occasion do you need a bouquet?  
SYSTEM : Okay, please let me know for which occasion do you need a bouquet?

#### **No-compose-birthday Intent:**

USER : For birthday, please.  
USER : I need a bouquet for birthday.  
SYSTEM : Here you can choose a wonderful birthday bouquet!

#### **No-compose-valentine Intent:**

USER : I need a bouquet for Saint Valentine’s Day.  
USER : For Saint Valentine’s Day.  
SYSTEM : Here you can choose a lovely bouquet for your sweetheart!

#### **No-compose-wedding Intent:**

USER : For wedding, please.  
USER : I need a bridal bouquet.  
SYSTEM : Here you can choose a wonderful bridal bouquet!

#### **No-add-compose-choose Intent:**

USER : No, that’s all.  
USER : No, thanks.  
SYSTEM : Super! You can purchase and get your wonderful bouquet at our office. Thank you!

#### **Yes-compose Intent:**

USER : I want to make a bouquet myself.  
SYSTEM : Alright! What kind of flowers would you like to add first?

@sys.number := the number of flowers selected by the user.  
@sys.color := the color of flowers selected by the user.  
@flower := the flower entity type. The System response for any intent is based on the previous parameters as follows:  
SYSTEM : I added \$ amount-2 \$ color-2 \$ flower-2.original to your bouquet. Anything else?

Figure 2 shows a snapshot of a conversation between a user and FloristBot. The chatbot can be complex and have tons of intents and contexts [27].

## V. EXPERIMENTS AND RESULTS

Experiments on the proposed algorithm which have been applied on the FloristBot have been conducted using different number of contexts, number of duplicated contexts, shortcut array sizes, and ratio of duplicated contexts in each duplicate context group. To assess the effectiveness of B-point Tree,

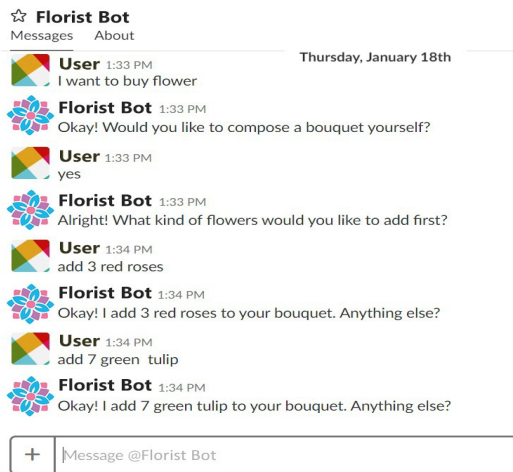


Fig. 2. An snapshot example of FloristBot Dialog System

the same ratio of duplicated contexts has been used over the normal BST. Figure 3-8 illustrate experiments results on both normal BST and B-point Tree. Search response time in milliseconds (MS) are presented in the graphs. As shown in figure 3, when the tree has a few number of nodes, the time decreases by decreasing the shortcut array size and the ratio. Increasing the number of groups to 1 contexts causes the time taken to rise by ten times. In spite of that, still the normal tree performance exceeds B-point Tree. As illustrated in Figures 4 and 5, when the number of contexts increases to 10,000, B-point Tree results started to outperform the normal BST. When the shortcut size is 5, B-point tree takes approximately half the time spent by the normal tree. But increasing the size to 7, B-point tree takes less than half of the time when the ratio of duplicated contexts is 15 and 20. It also takes almost the same time when the ratio equals to 5 and 10. Rising the shortcut array size to 10 as well gives a result that the same as in 5 and 7. To sum up, rising the ratio in this particular array size shows a better performance for the B-point Tree. As the number of groups increases to 1,000,000,

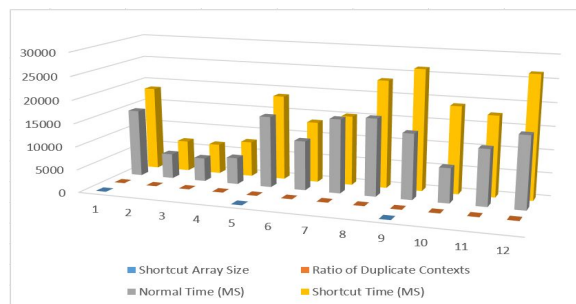


Fig. 3. Experiments with 1000 generated contexts and 1,000,000 context groups

B-point Tree performance curve fall down as well to nearly the same time spent by the normal tree, except for some cases such as the ratio of 20. In this case, B-point Tree achieves a better execution by 20%-30%. Figure 6 shows that when rising

the number of nodes (contexts) to 100,000 shows a massive improvement in the shortcut performance over the normal one when the number of groups is 100,000. The performance is increased by 75%-90% in general, regardless of the number of the shortcut array size and the number of duplicated contexts. When the number of contexts increases to 1,000,000, it gives

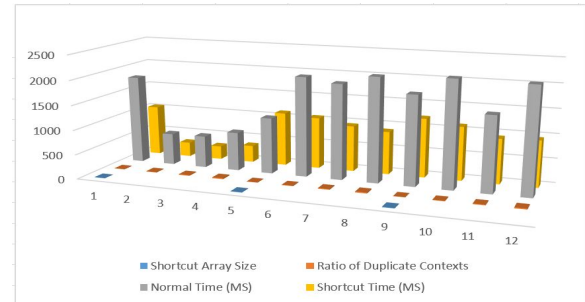


Fig. 4. Experiments with 10,000 generated contexts and 100,000 context groups

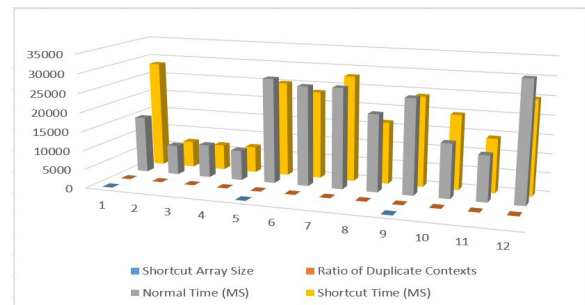


Fig. 5. Experiments with 10,000 generated contexts and 1,000,000 context groups

the shortcut an advantage over the normal tree by 50%-70% for most cases. Additionally, decreasing the shortcut array size to 5 takes even less time than the other cases (7, 10) as shown in Figures 7 and 8. This suggests that B-Point Tree algorithm is capable of increasing the efficiency of chatbots.

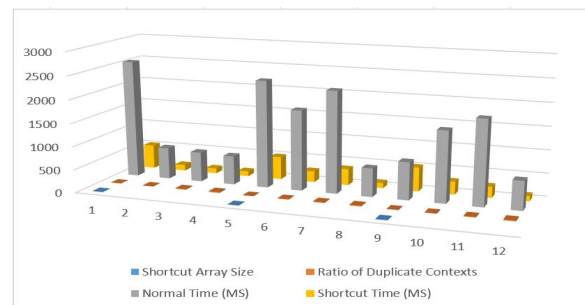


Fig. 6. Experiments with 100,000 generated contexts and 100,000 context groups

## VI. CONCLUSION

Retrieving a quick and accurate answer to any user's question seemed to be a challenging task in chatbot systems. This



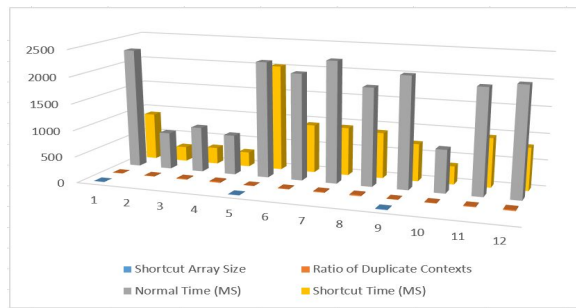


Fig. 7. Experiments with 1,000,000 generated contexts and 100,000 context groups

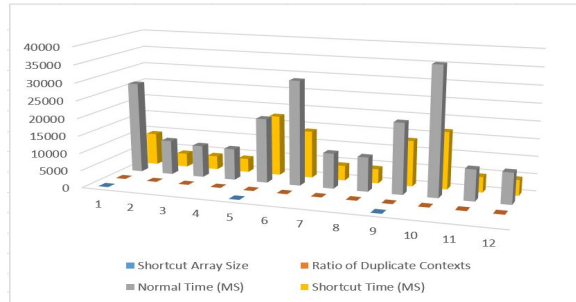


Fig. 8. Experiments with 1,000,000 generated contexts and 1,000,000 context groups

paper had introduced a method called B-Point Tree to speed up the search process by adding an additional data structure that contains shortcut pointers to the traditional search BST. The experiments had been conducted on a FloristBot, a chatbot that behaves as human personnel in a flower shop. The FloristBot is used to entertain customers and take orders. The results of the experiments indicate that the B-Point Tree manages to outperform the traditional BST algorithm. This suggests that the use of shortcut pointers may reduce the searching processing time, consequently may increase the efficiency of the chatbot.

## REFERENCES

- Jusoh, S., "A study on nlp applications and ambiguity problems." *Journal of Theoretical & Applied Information Technology*, vol. 96, no. 6, 2018.
- "Conversational interfaces powerful speech technology from chinas leading internet company makes it much easier to use a smartphone," <https://www.technologyreview.com/s/600766/10-breakthrough-technologies-2016-conversational-interfaces/>, accessed: 2018-01-05.
- "Will be the year of conversational commerce," <https://medium.com/chris-messina/2016-will-be-the-year-of-conversational-commerce-1586e85e3991>, accessed: 2018-01-10.
- "Tma associates specialized digital assistants and bots," <http://www.tmaa.com/specializeddigitalassistantsandbots.html>, accessed: 2017-12-25.
- "Facebook hopes chatbots can solve app overload," <http://www.wsj.com/articles/facebook-hopes-chatbots-can-solve-app-overload-1460930220>, accessed: 2018-03-20.
- Gregori, E., "Evaluation of modern tools for an omscs advisor chatbot," 2017.
- Kandampully, J. and Suhartanto, D., "Customer loyalty in the hotel industry: the role of customer satisfaction and image," *International journal of contemporary hospitality management*, vol. 12, no. 6, pp. 346–351, 2000.
- Hallowell, R., "The relationships of customer satisfaction, customer loyalty, and profitability: an empirical study," *International journal of service industry management*, vol. 7, no. 4, pp. 27–42, 1996.
- Lee, J., Lee, J., and Feick, L., "The impact of switching costs on the customer satisfaction-loyalty link: mobile phone service in france," *Journal of services marketing*, vol. 15, no. 1, pp. 35–48, 2001.
- Mägi, A. W., "Share of wallet in retailing: the effects of customer satisfaction, loyalty cards and shopper characteristics," *Journal of Retailing*, vol. 79, no. 2, pp. 97–106, 2003.
- Dimitriadis, Z. S., "Customer satisfaction, loyalty and commitment in service organizations: Some evidence from greece," *Management Research News*, vol. 29, no. 12, pp. 782–800, 2006.
- Tsvetkova, M., García-Gavilanes, R., Floridi, L., and Yasseri, T., "Even good bots fight: The case of wikipedia," *PloS one*, vol. 12, no. 2, p. e0171774, 2017.
- Ngan, H. T. et al., "Analysis the relationship among service quality, customer satisfaction, and customer loyalty with hotel industry in hanoi vietnam," Ph.D. dissertation, (Master), 2017.
- Wu, Y., Wu, W., Xing, C., Xu, C., Li, Z., and Zhou, M., "A sequential matching framework for multi-turn response selection in retrieval-based chatbots," *arXiv preprint arXiv:1710.11344*, 2017.
- Vandenbergh, B., "Bot personas as off-the-shelf users," in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2017, pp. 782–789.
- Kane, D. A., "The role of chatbots in teaching and learning," *E-Learning and the Academic Library: Essays on Innovative Initiatives*, p. 131, 2016.
- Abdul-Kader, S. A. and Woods, J., "Survey on chatbot design techniques in speech conversation systems," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 7, pp. 72–80, 2015.
- Thomas, N., "An e-business chatbot using aiml and lsa," in *Advances in Computing, Communications and Informatics (ICACCI)*, 2016 International Conference on. IEEE, 2016, pp. 2740–2742.
- Kuligowska, K., "Commercial chatbot: performance evaluation, usability metrics and quality standards of embodied conversational agents," 2015.
- Kowalski, S., Pavlovska, K., and Goldstein, M., "Two case studies in using chatbots for security training," in *IFIP World Conference on Information Security Education*. Springer, 2009, pp. 265–272.
- Lee, D., Han, S.-H., Oh, K.-J., and Choi, H.-J., "A temporal community contexts based funny joke generation," in *Mobile Data Management (MDM)*, 2017 18th IEEE International Conference on. IEEE, 2017, pp. 360–365.
- Mitrevski, M., "Getting started with wit.ai," in *Developing Conversational Interfaces for iOS*. Springer, 2018, pp. 143–164.
- Fox, E. A., Hix, D., Nowell, L. T., Brueni, D. J., Wake, W. C., Heath, L. S., and Rao, D., "Users, user interfaces, and objects: Envision, a digital library," *Journal of the American Society for Information Science*, vol. 44, no. 8, p. 480, 1993.
- Santangelo, A., Augello, A., Gentile, A., Pilato, G., and Gaglio, S., "A chat-bot based multimodal virtual guide for cultural heritage tours," in *PSC*, 2006, pp. 114–120.
- Grosz, B. N., Labrou, Y., and Chan, H. Y., "A declarative approach to business rules in contracts: courteous logic programs in xml," in *Proceedings of the 1st ACM conference on Electronic commerce*. ACM, 1999, pp. 68–77.
- Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., and Tsujii, J., "Brat: a web-based tool for nlp-assisted text annotation," in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2012, pp. 102–107.
- Dale, R., "The return of the chatbots," *Natural Language Engineering*, vol. 22, no. 5, pp. 811–817, 2016.
- Cormen, T. H., *Introduction to algorithms*. MIT press, 2009.
- Bayer, R. and Schkolnick, M., "Concurrency of operations on b-trees," *Acta informatica*, vol. 9, no. 1, pp. 1–21, 1977.
- Gray, J. and Reuter, A., *Transaction processing: concepts and techniques*. Elsevier, 1992.
- Knuth, D. E., "Optimum binary search trees," *Acta informatica*, vol. 1, no. 1, pp. 14–25, 1971.
- Sleator, D. D. and Tarjan, R. E., "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.
- "Dialogflow," <https://dialogflow.com/>, accessed: 2017-11-20.