Confabulation Based Sentence Completion for Machine Reading*

Qinru Qiu[1], Qing Wu[2], Daniel J. Burns[2], Michael J. Moore[2], Robinson E. Pino[2], Morgan Bishop[2], and Richard W. Linderman[2]

[1]Dept. of Electrical & Computer Engineering
State University of New York at Binghamton
Binghamton, NY 13850, USA

[2]Air Force Research Laboratory
Information Directorate, RITC
525 Brooks Road, Rome, NY 13441, USA

***Abstract* — Sentence completion and prediction refers to the capability of filling missing words in any incomplete sentences. It is one of the keys to reading comprehension, thus making sentence completion an indispensible component of machine reading. Cogent confabulation is a bio-inspired computational model that mimics the human information processing. The building of confabulation knowledge base uses an unsupervised machine learning algorithm that extracts the relations between objects at the symbolic level. In this work, we propose performance improved training and recall algorithms that apply the cogent confabulation model to solve the sentence completion problem. Our training algorithm adopts a two-level hash table, which significantly improves the training speed, so that a large knowledge base can be built at relatively low computation cost. The proposed recall function fills missing words based on the sentence context. Experimental results show that our software can complete trained sentences with 100% accuracy. It also gives semantically correct answers to more than two thirds of the testing sentences that have not been trained before.**

Keywords – cogent confabulation, machine reading, sentence completion, unsupervised learning, hash table

I. INTRODUCTION

Sentence completion and prediction, which refers to the capability of filling missing words in an incomplete sentence, is one of the keys to reading comprehension. In this paper, we focus on modeling, training and recall techniques for automatic sentence completion using unsupervised machine learning. Automatic sentence completion can have many important applications. It can be used to predict and complete a sentence to reduce the user keystrokes. It can be used to improve the accuracy of Optical Character Recognition (OCR) by providing semantic information. With careful design, a sentence completion test can provide quantitative measurement of the quality of the knowledge base accrued during unsupervised machine reading.

To complete a sentence, it requires not only the appropriate vocabulary, but also the ability to analyze the given sentence and identify the structural and semantic clues that determines the meaning and nature of the missing words. Therefore, a large vocabulary, the prior knowledge in semantic connections of words as well as a good language sense are important to accomplish this task. These must be obtained from extensive reading and training.

* Received and approved for public release by AFRL on 11/03/2010, case number 88ABW-2010-5869.

Many of the previous works in sentence completion aim at providing a "tab-complete" editing assistance. In [1], an "interactive keyboard" is proposed that predicts the most likely keystrokes based on the past sequence. References [2]~[4] propose techniques to predict the next user command in a Unix system. The problem of natural language sentence completion is considered in some typing assistance tools for *apraxic* [5] and *dyslexic* [6]. They provide a list of possible word completions for users to choose from. The authors of [7] propose an interactive word-completion algorithm based on integrated semantic knowledge and *n*-gram probabilities. The authors of [8] propose an information retrieval approach for sentence completion. It is further improved in [9] by using an *n*-gram language model [12].

In this work we focus on the general sentence completion problem. The input of our problem is a sentence fragment with missing words at random locations. We are interested in filling in the missing words so as to create a syntactically correct and semantically coherent sentence. The sequence analysis techniques in [1]~[4] are not applicable to our problem since there is no relative past information. The *n*-gram language model analyzes the sentence and predicts words in a sequential order from left to right. It may not be effective to the general sentence completion problem where the missing words can locate in the middle or at the beginning of a sentence.

In this paper, we adopt the cogent confabulation model [10] to solve the sentence completion problem. Cogent confabulation is a bio-inspired model that mimics the human information processing. It is an unsupervised machine learning algorithm that extracts *posterior probabilities* among objects at the symbol level. In this work, we apply cogent confabulation to extract the relations among words in a sentence. A *knowledge base* (KB) is obtained by reading an extensive body of literature. When given an incomplete sentence, the most appropriate words will be selected based on the knowledge base. The selection procedure is an analogy to the activation of the human neurological system. Each word (or phrase) is analogy to a set of neurons and the posterior probabilities among words/phrases are analogy to the weight of the synapses connecting the neurons. The neurons with the highest excitation will be activated and further excite other neurons. When this procedure converges, the neurons (i.e. words or phrases) with the highest excitation will be selected.

In the rest of the paper, we refer to the procedure of knowledge base construction as the *training* process and refer to the procedure of words selection and sentence completion as the *recall* process. The characteristics of the proposed sentence completion technique can be summarized as the follows:

1. It is based on the cogent confabulation model, which captures the posterior probability among words and phrases in a sentence. Therefore, our technique completes the sentence in a way such that the likelihood of the observed part is maximized. In contrast, the *n*-gram model maximizes the likelihood of the missing words. Although there is no definite advantage of one approach over another, they sometimes give different results.
2. Our knowledge base provides the relations between all words or word pairs in the sentence. This enables the software to fill in the missing words at the beginning of the sentence based on the information provided later.
3. The recall function mimics the information processing in the human neurology system, where neurons are exciting and being excited at the same time. Therefore, when multiple entries are missing, the selections of these entries evolve simultaneously.
4. Comparing to the original confabulation model proposed in [10], our model achieves better performance as we allow multiple symbols to be excited at the same time and hence has a larger search space. Our training function adopts a 2-level hash table, which significantly improves the training speed, so that we can extract knowledge from a larger training corpus in relatively short time. This enables the system to perform extensive reading while still maintaining a high quality knowledge base.

The rest of the paper is organized as the follows. The background of the cogent confabulation model is provided in Section II. The detailed discussion of our knowledge model and our training/recall algorithms are provided in Section III. Section IV presents the experimental results and Section V gives the conclusions and our vision on future works.

II. BACKGROUND

Cogent confabulation [10] is an emerging computation model that mimics the Hebbian learning, the information storage and inter-relation of symbolic concepts, and the recall operations of the brain. Based on the theory, the cognitive information process consists of two steps: learning and recall. During the learning step, the knowledge links are established and strengthened as symbols are co-activated. During recall, a neuron receives excitations from other activated neurons. A "winner-takes-all" strategy takes place within each lexicon. Only the neurons (in a lexicon) that represent the winning symbol will be activated and the winner neurons will activate other neurons through knowledge links. At the same time, those neurons that did not win in this procedure will be suppressed.

Figure 1 shows an example of lexicons, symbols and knowledge links. The three columns in Figure 1 represent three lexicons for the concept of shape, object and color with each box representing a neuron. Different combinations of neurons represent different symbols. For example, as shown in Figure 1, the pink neurons in lexicon I represent the cylinder shape, the orange and yellow neurons in lexicon II represent a fire extinguisher and a cup, while the red neurons in lexicon III represent the red color. When a cylinder shaped object is perceived, the neurons that represent the concepts "fire extinguisher" and "cup" will be excited. However, if a cylinder shape and a red color are both perceived, the neurons associated with "fire extinguisher" receives more excitation and become activated while the neurons associated with the concept "cup" will be suppressed. At the same time, the neurons associated with "fire extinguisher" will further excite the neurons associated with its corresponding shape and color and eventually make those symbols stand out from other symbols in lexicon I and III.

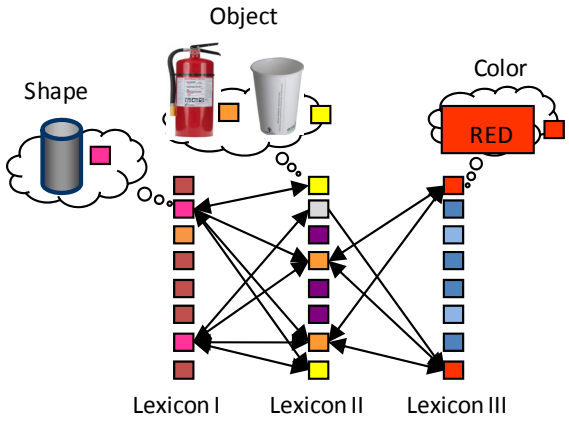

Figure 1. A simple example of lexicons, symbols and knowledge links.

A computational model for cogent confabulation is proposed in [10]. Based on this model, a *lexicon* is a collection of symbols. A *knowledge link* (*KL*) from lexicon A to B is a matrix with the row representing a source symbol in A and the column representing a target symbol in B. The (i, j)th entry of the matrix represents the strength of the synapse between the source symbol s_i and the target symbol t_j. It is quantified as the conditional probability $P(s_i \mid t_j)$. The collection of all knowledge links is called a *knowledge base* (*KB*). The knowledge bases are obtained during the learning procedure. During recall, the excitation level of all symbols in each lexicon is evaluated. Let l denote a lexicon, F_l denote the set of lexicons that have knowledge links going into lexicon l, and S_l denote the set of symbols that belong to lexicon l. The excitation level of a symbol t in lexicon l can be calculated as:

$$I(t) = \sum_{k \in F_l} \sum_{s \in S_k} I(s) \left[\ln \left(\frac{P(s|t)}{p_0} \right) + B \right] , t \in S_l.$$

The function $I(s)$ is the excitation level of the source symbol s. Due to the "winner-takes-all" policy, the value of $I(s)$ is either "1" or "0". The parameter p_0 is the smallest meaningful

value of $P(s_i \mid t_j)$. The parameter B is a positive global constant called the *bandgap*. The purpose of introducing B in the function is to ensure that a symbol receiving N active knowledge links will always have a higher excitation level than a symbol receiving (N-1) active knowledge links, regardless of the strength of the knowledge links.

III. CONFABULATION BASED SENTENCE COMPLETION

In this section, we will present the knowledge base (KB) model for the sentence completion software followed by the training and recall algorithms.

A. Knowledge Base Model

Similar to [10], in this work, we assume that the maximum length of a sentence is 20 words. Any sentence that is longer than 20 words will be truncated. We also assume that the empty space is a word. Any sentence that is shorter than 20 words will be padded with empty spaces.

A total of 39 lexicons are constructed for a sentence. They are divided into 2 groups. Lexicons 0 through 19 belong to the first group. Each group 1 lexicon associates to a single word in the sentence. The ith lexicon represents the ith word. Lexicons 20~38 belong to the second group. Each group 2 lexicon associates to a pair of adjacent words. The lexicon labeled (20+i) represents the pair of words in the (i+1)th and (i+2)th location. Associated to each lexicon is a collection of symbols. A symbol is a word or a pair of words that appears in the corresponding location. We use S_A to denote the set of symbols associated to lexicon A.

A knowledge link (KL) from lexicon A to B is an $M \times N$ matrix, where M and N are the cardinalities of symbol sets S_A and S_B. The (i, j)th entry of the knowledge link gives the conditional probability $P(i|j)$, where $i \in S_A$, and $j \in S_B$. Symbols i and j are referred as *source symbol* and *target symbol*.

For our sentence completion system, between any two lexicons there is a knowledge link. If we consider the lexicons as vertices and knowledge links as directed edges between the vertices, then they form a complete graph.

B. Training Algorithm Using Hash Technique

The training of the confabulation model is the procedure to construct the probability matrix between source symbols and target symbols. Figure 2 gives a simple algorithm for the construction of the knowledge base. First the program scans through the training corpus and count the number of co-occurrences of symbols in different lexicons. Then for each symbol pair it calculates their posterior probability.

Although computationally simple, the challenge of this training algorithm is its memory complexity. For example, the English version of the book "Round the Moon" has about 47×10^3 words. Our analysis shows that it has 23×10^3 distinguished symbols (i.e. words and word pairs). Figure 3 shows how the symbols are distributed over different lexicons. As we mentioned earlier, each knowledge link is an $M \times N$ matrix, where M and N are the symbol size of the source and target lexicons. Without any compression, the trained knowledge base will have 2.3×10^9 entries which are equivalent to be 9.2 GBytes.

Reset the co-occurrence matrix $\mathbf{co_{AB}}$ to 0, where $0 \le A, B \le 39$
//count the co-occurrence of symbols
For each sentence in training corpus {
  For each lexicon A, $0 \le A \le 39$ {
    For each lexicon B, $0 \le B \le 39$ {
      If $A \ne B$, $co_{AB}[s_A][s_B]$++;
    }
  }
}
//calculate the posterior probability (i.e. knowledge link)
$kl_{AB}[i][j] = \frac{co_{AB}[i][j]}{\sum_i co_{AB}[i][j]}, \forall i \in S_A, \forall j \in S_B, 0 \le A, B \le 39$

Figure 2. A simple algorithm of knowledge base construction.

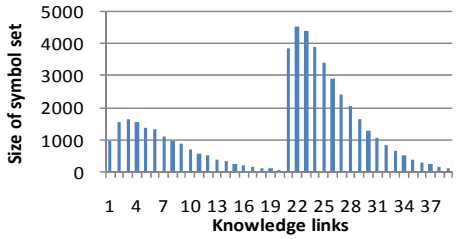

Figure 3. Symbol distribution in different knowledge links.

Fortunately, the knowledge links are sparse matrices. Only less than 0.1% of the matrix has non-zero values. Therefore, an option to reduce the memory cost is to store the knowledge using the *list of list* (LIL) or the *Yale* format, which have been widely used for sparse matrix storage. However, this leads to the second problem. As the size of the training corpus grows, the number of symbols of each lexicon can easily go up to hundreds of thousands. Even with the best search algorithm, the time to locate the entry in the compressed matrix grows logarithmically and soon the algorithm will become prohibitively slow. Furthermore, each symbol is a string of characters (i.e. word or word pair). Therefore, the search procedure consists of a sequence of string comparisons, which will further slow it down.

In this work, we propose to store the knowledge base using a hash table and speed up the search using a 2-level hash technique. Figure 4 gives the architecture of a hash table. The hash function maps each identifying value (i.e. key) to a bucket in an array. Hash collisions occur when multiple keys map to the same entry. Each entry's keys are maintained in a linked list. Each entry in the array and the collision list is associated with a number that gives the location of the key in the storage table. If the key is neither in the array list nor the collision list, then it is a new key and a new entry will be allocated for it.

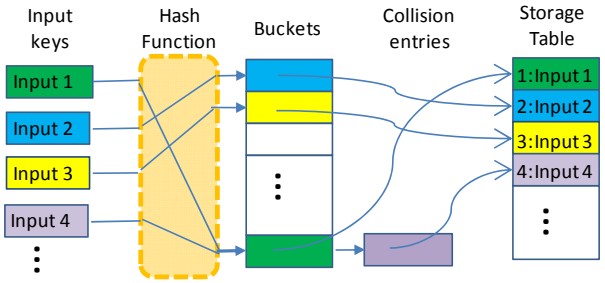

Figure 4. Architecture of a hash table.

Our training function maintains two levels of hash tables. At the upper level, there is one large hash table with 2^{21} buckets. Its function is to encode each unique word or word pair to an identifying number, which is referred as source/target symbol. At the lower level, there are $38 \times 39 = 1482$ smaller hash tables. Each one of them has 2^{15} buckets. The second level hash tables transform the pair of symbols in the source and target lexicon to a number that identifies the location of their corresponding knowledge link entry. Figure 5 gives an example of the 2-level hash function.

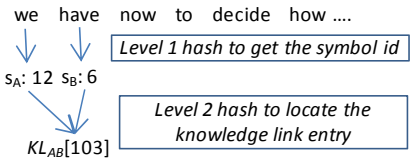

Figure 5. Two-level hash function locates the knowledge link entry of the given input strings.

The average size of our training file is 358 Kbyte. On average, each training file has 64×10^3 words and 35×10^3 distinguished words and word pairs. Our experimental results show that the average collision number of the first level hash table is 0.02. This means that, to search the symbol value of a string, we need 1.02 read accesses in average. Compared to a binary search technique which requires $log_2(35 \times 10^3) \approx 15$ read accesses, the hash table provides almost 15 times speed up. More experimental results on the performance of the hash based training function will be presented in Section IV.

At the end of training, the knowledge base will be written to a file. The first part of the file is the symbol encoding of all the unique words and word pairs that have been encountered in the training corpus. The second part of the file gives the content of the knowledge links. Each knowledge link is a sorted LIL. Each element in the LIL specifies the source and target symbol of the corresponding entry in the knowledge link matrix as well as the strength of this link (i.e. the posterior probability between the source and target).

C. Incremental learning

To enable extensive reading, the training algorithm should be able to read and learn continuously. Each time after a new book or article is read, the knowledge learned must be added into the existing knowledge base, which is referred as *main knowledge base*, and we refer this as incremental learning. It is the procedure of merging two knowledge bases together.

Figure 6 shows the work flow of incremental learning. It consists of 2 steps, reading and merging. The former is to generate a new KB from the given training file and the latter is to combine the new KB with the main KB.

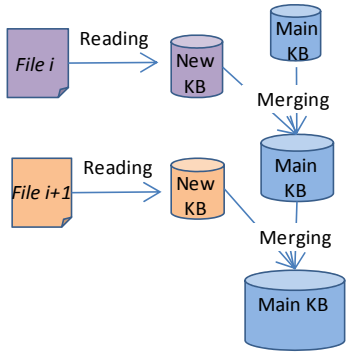

Figure 6. The incremental learning involves 2 steps: reading and merging.

```
For each symbol in the new KB, find their encoding in the main KB;
For each knowledge link i{
  Load the knowledge link from main KB and denote it as KL1;
  Load the knowledge link from new KB and denote it as KL2 ;
  Sort the rows and columns of KL2 based on the new symbols coding;
  for each row in KL2 {
    if the row is in KL1 then merge the two rows;
    else append the row to KL1;
  }
}
```

Figure 7. Flow of knowledge base merging.

Figure 7 gives the algorithm of knowledge base merging. Because the new knowledge base and the main knowledge base are trained at different times, they encode the word and word pairs in different ways. Therefore, the first step to merge two knowledge bases is to unify their symbolic representation. After that, for each knowledge link, we load the main KB and new KB and store them as a list of list (LIL). The knowledge links from main KB and new KB are denoted as KL_1 and KL_2 respectively. Note that both knowledge links are sorted however based on their own symbol encoding scheme. We keep the KL_1 and sort the KL_2 based on the new symbol encoding scheme. After that, we merge the two together.

Although the size of the knowledge base can easily go up to several Gigabytes, during the merging, only 2 knowledge links are maintained in the memory, therefore, the memory complexity of the merging algorithm is well controlled. The computation complexity of merging two sorted lists is linear with respect to the size of the list. Therefore, the computation complexity of the algorithm is bounded by the complexity to sort KL_2, which is $O(nlogn)$, where n is the size of KL_2. Note

that KL_2 is one of the knowledge links in the new KB, which is obtained after reading one training file, while KL_1 is one of the knowledge links in the main KB, which is the combination of multiple KBs. It is obviously more efficient to sort KL_2 instead of KL_1 because $|KL_1| \gg |KL_2|$.

D. Recall Algorithm

```
for each lexicon whose content is known{
    encode the word/word pair to its symbol representation;
    set the symbol to be active;
}
for (N = MAX_AMBIGUIOUS; N > 0; N--) {
    converged = FALSE;
    iter = 0;
    while(!converged) {
        for each lexicon whose content is unknown{
            for each symbol i associated to the lexicon {
                calculate the excitation level of i;
            }
            select the N highest excited symbols and set them to be active;
        }
        iter ++;
        if (the activation set does not changesince the last iteration)
            then converged = TRUE;
        if (iter >= MAX_ITERATION)
            then converged = TRUE;
    }
}
```

Figure 8. Sentence completion: recall.

The recall algorithm fills in the blank spaces in an incomplete sentence. For those lexicons whose content is given, we first encode the string to its symbolic representation. The symbol is labeled as "*active*". Then for each lexicon whose content is unknown, we calculate the *excitation level* of its symbols. The excitation level of a symbol i in lexicon B is defined as:

$$EL_B[i] = \sum_{A \neq B} \sum_{j \in \{active\ symbols\ in\ A\}} kl_{AB}[j][i],$$

where $kl_{AB}[j][i]$ is the knowledge value from symbol j in lexicon A to symbol i in lexicon B. The N highest excited symbols in this lexicon are set to be active. These symbols will further excite other symbols in other unknown lexicons. This procedure will continue until the activated symbols in all lexicons do not change anymore. If the convergence cannot be reached after a given number of iterations, then we will force the procedure to converge. After that, we will reduce the value of N and repeat the above procedure. The entire recall procedure will stop when N is reduced to zero.

Experimental results show that increasing the value of N helps to give more meaningful sentence completion results; however, it also increases the runtime exponentially.

IV. Experimental Results

In this section, we provide experimental results for the performance of the training algorithm and the quality of the recall algorithm.

A. Performance of the training function

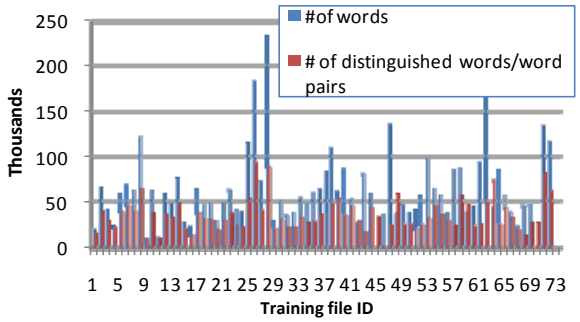

Figure 9. Size of the training corpus.

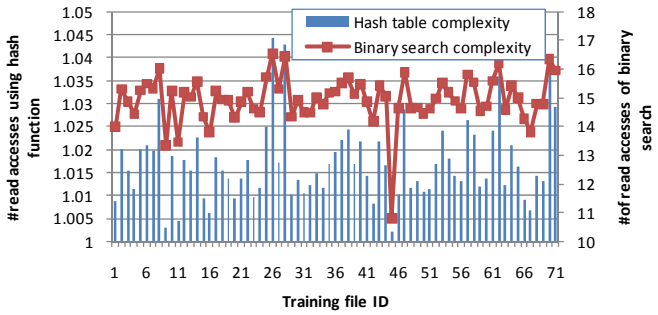

Figure 10. Performance of level 1 hash function.

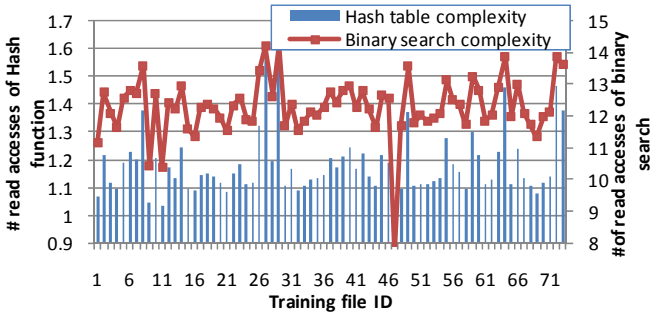

Figure 11. Performance of level 2 hash function for KL #400.

Our training corpus consists of 73 classic literatures, including the works from Aesop, Louisa May Alcott, James Matthew Barrie, and the Bronte sisters, et al. Each book is used as a separate training file. To avoid extremely long sentences, we assume that all punctuations except '`' and '-' indicate an end of a sentence. Figure 9 shows the number of words and the number of distinguished words/word pairs in the training corpus. As we can see, the variance in the number of distinguished words/word pairs is much smaller than the variance of the sizes of the training files.

In the first setup, incremental training is used in the experiment. Each time after a book is trained; the new knowledge base will be merged to the main knowledge base. Figure 10 shows the performance of the first level hash function by comparing its complexity to the complexity of

binary search. In average, to find the symbol encoding of a string, the average number of read access is 1.02 and 14.9 using the hash table and the binary search respectively. Therefore, using the hash technique provides almost 15X speed up in performance.

Figure 11 shows the similar results for the 2nd level hash function. There are $38 \times 39 = 1482$ hash tables in the second level. Here we only report the result for the one which corresponds to the knowledge link #400. As we can see from the figure, the average number of read access to locate a knowledge entry in the knowledge link matrix is 1.3 using hash table, and it is 12.8 using binary search.

Figure 12 shows the distribution of the number of collisions of the hash table for knowledge link #400. The training file for this data is L. M. Alcott's "Little Woman". The largest collision is 4, which happens to only 1 entry in the knowledge link matrix while the vast majority of the entries have no collision.

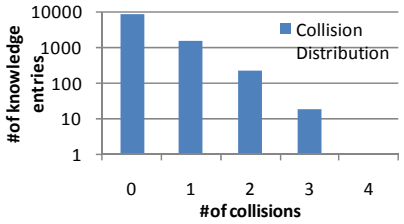

Figure 12. The distribution of the number of collisions in the second level hash table for KL #400. (training file: "Little Woman" by L. M. Alcott)

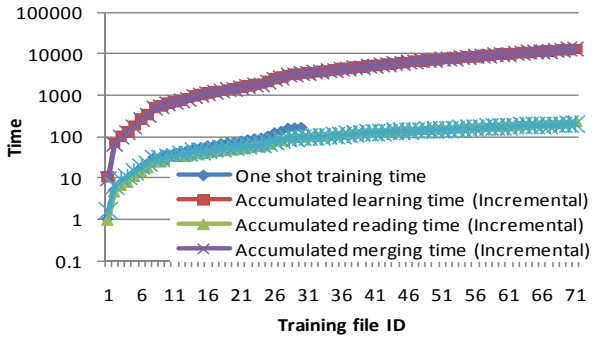

Figure 13. Performance of training.

Figure 13 shows the performance of our incremental learning algorithm. In the figure, we report the overall accumulated learning time, as well as the accumulated reading time and the accumulated merging time. As we can see, the times to merge the two knowledge bases dominate the computation cost of the incremental learning. We also concatenated all the training files and training them together. The result is reported as "One shot training time" in the figure. As we can see, the training time of the concatenated files is almost the same as the accumulated training time when these files are trained separately. This is because, due to the help of the hash function, the time to locate a knowledge entry remains almost constant even though the size of the knowledge base increases. However, because the "one shot" training has to maintain the entire KB in the memory, it soon exceeds the memory limit, which is set to be 500,000 entries per knowledge link. Therefore, it can train at most 31 books. In contrast, the incremental training only has to keep just one knowledge link in the new KB and the main KB. Therefore, its performance is sustainable.

B. Quality of sentence completion outputs

In the first setup, we use the recall function to complete some trained sentences. We arbitrarily selected 69 sentences from the training text and randomly remove words with 0.3 probabilities. Table 1 gives the quality of the recall when the MAX_AMBIGUITY varies from 1 to 20. We categorize the result of the recall into 3 groups. A *perfect recall* refers to the cases when the exact test sentences are recalled. A *correct recall* refers to the cases when the recall function does not give the exact test sentence; however gives another sentence in the test corpus. Although the result is not exactly what we asked for, it is still correct. A wrong recall refers to the cases when a sentence not in the training text has been confabulated. Our results show that, when MAX_AMBIGUITY is greater than or equal to 5, all sentences are recalled correctly. Otherwise, about 3% of the sentences cannot be correctly recalled.

Table 1. Recall of the trained sentences.

MAX_AMBIGUITY	Perfect recall	Correct recall	Wrong recall
20	66	3	0
10	66	3	0
5	66	3	0
2	64	3	2
1	64	3	2

In the second setup, we evaluate the algorithm with simple sentence completion tests extracted from a Kindergarten workbook. Table 2 gives the list of input sentences and sentences completed by our program. None of the sentences have been read during the training. The answers that are not reasonable are highlighted in bold. As we can see, 10 of the 15 sentences are completed correctly.

In the third setup, we use the book "Great Expectations" by Charles Dickens, and a children's story book "Why the Sea is Salt" as our test files. Neither of the books has been read during the training procedure. From each book we randomly picked 100 sentences and then randomly took out 30% of the words from each selected sentence.

Some of the decimated sentences and the results produced by our sentence completion software are listed. The underlined texts in italic are completed by the software. We can see that the majority of the confabulated sentences read reasonably well with correct grammar. While a few of them are with minor grammar errors or do not really make sense.

Table 2. Completion of simple sentences.

	Input sentence	Completed sentences
1	I __ a good idea	I *had* a good idea
2	But he has a lot __ work to do	But he has a lot ***to*** work to do
3	He gets help _ his friends	He gets help *to* his friends
4	__ are many jobs to do	***You*** are many jobs to do
5	They __ so much work	They *do* so much work
6	__ the boys work	*And* the boys work
7	They clean __ clean	They clean ***were*** clean
8	That car __ not clean	That car ***took*** not clean
9	There ____ many houses in town	There *were* many houses in town
10	Cars __ on the streets	Cars ***lamps*** on the streets
11	He works in __ store	He works in *her* store
12	He lives __ the house	He lives *in* the house
13	A tornado __ coming	A tornado *is* coming
14	The boy didn't want the tornado __ come	The boy didn't want the tornado *to* come
15	No __ was happy about this	No *I* was happy about this

Input: ____ the poor ____ wife.
Output: *Said* the poor *invalid* wife.

Input: ____ gone with ____ ____.
Output: *And* gone with *the clouds*.

Input: Ladies ____ ____.
Output: Ladies *and gentlemen*.

Input: At which crisis I partially ___ the use of my ____.
Output: At which crisis I partially *recovered* the use of my *senses*.

Input: As its on ____ ____ service.
Output: As its on *the morning* service.

Input: ____ these things ____ ____ without then knowing that ____ ____ ____.
Output: *And* these things *had been* without then knowing that *they became realities*.

Input: ____ Joe ____ to hammer and clink.
Output: *And* Joe *have* to hammer and clink.

Input: ___ drew a pitcher of beer ___ the ___.
Output: *And* drew a pitcher of beer *in* the *wall*.

Input: ___ the ___ man ___ determined to deliver the cows ___ ___ Hiysi.
Output: *At* the *last* man *he* determined to deliver the cows *back to* Hiysi.

Input: To make ____ ____.
Output: To make *matters straight*.

Input: Where are we ___ to get something to ___.
Output: Where are we *going* to get something to *me*.

Input: ___ will we celebrate.
Output: *When* will we celebrate.

Input: He did ___ know what ___ do.
Output: He did *not* know what *to* do.

Input: Why ___ ___ come ___?
Output: Why *are you* come *here*?

Input: Looking ___ ___ rich man.
Output: Looking *at the* rich man.

Input: ___ ___ take it to Hiysi.
Output: *Will you* take it to Hiysi.

Input: ___ ___ ___ lived deep in the ___
Output: *And I have* lived deep in the *way*.

Input: ___ caused ___ Joe reentering the ___ empty handed.
Output: *That* caused *Ned* Joe reentering the *brook* empty handed.

Input: And then ___ stood ___.
Output: And then *she* stood *still*.

Input: Opening ___ door ___spit stiffly ___ ___ high stocks.
Output: Opening *the* door *and* spit stiffly *enter before* high stocks.

Input: I ___ ___ little ___ of my scattered ___.
Output: I *went a* little *out* of my scattered *to*.

Input: ___ am ___ a ___ in ___ name ___ the ___.
Output: *I* am *going* a *tell* in *the* name *of* the *father*.

Input: And pray ___ ___ you ___ ___ ___.
Output: And pray *circulate them* you *three influential quarters*.

In the last setup, we test the system using the Rotter incomplete sentence test [11]. The Rotter test is usually used to find out the personality and the psychological state of the subject. About 24 incomplete sentences are fed into the system; all of them are completed with meaningful output. 11 out of the 24 output sentences are exactly the same as sentences in the training file. The following is a list of these 11 sentences.

- I feel *deeply interested in her*.
- I regret *that they did not visit us before sailing*.
- I am afraid of *what may happen if I*.
- My father is *willing to give you a last chance*.
- The future *wellbeing of their child*.
- My nerves are *torn to pieces*.
- Girls were *made to take care of boys*.
- School *would be a complete change*.
- I need *not narrate in detail the further struggles I had*.
- I hate *anybody to come upon me so unexpectedly*.
- I wish *these papers did not come in the house*.

The other 13 sentences are made up (or "confabulated") by the program. They are listed in the next.

- Other people are *like those stupid hoppers*.
- I am best when *to be able to do it*.
- What bothers me is *going to say next before he says it*.
- The happiest time is *coming when it will take the place*.
- I dislike to *dwell on the result*.
- I failed to *see what I did*.
- A home is *at the end of the gallery opened*.
- Boys can be *of the greatest assistance to me*.
- My mother always *on leaving the letter unanswered*.
- I suffer *matters to take their course*.
- Other kids *were couched in language which made Michel jump*.
- My greatest worry is *to love her for her tender sympathy*.
- What pains me is *to be in the house*.

V. Conclusions and Future Works

We have introduced the modeling, training and recall techniques of our confabulation based sentence completion software. The hash based training algorithm supports extensive reading in relatively short time and at the same time maintains a high quality knowledge base. The software can recall sentences in the training files with 100% accuracy. It can also fill in missing words in simple sentences or provide meaningful sentences based on the given initial words. Although it gives promising results, some of the sentences provided by the software are not logically correct. One promising method to improve the results is to incorporate semantic information (such as parts of speech) with the confabulation model and algorithms.

Acknowledgment of Support and Disclaimer

This work is funded by the Air Force Office of Scientific Research, under contract FA8750-10-C-0233.

Any Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of AFRL or its contractors.

References

[1] J. darragh and I. Witten, "The Reactive keyboard," *Cambridge University Press*, 1992.

[2] H. Motoda and K. Yoshida, "Machine learning techniques to make computers easier to use," *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[3] B. Davison and H. Hirsch, "Predicting sequences of user actions," *Proceedings of the AAAI/ICML Workshop on Predicting the Future: AI Approaches to Time Series Analysis*, 1998.

[4] B. Korvemaker and R. Greiner, "Predicting Unix command lines: adjusting to user patterns," *Proceedings of the National Conference on Artificial Intelligence*, 2000.

[5] N. Garay-Vitoria and J. Abascal, "A comparison of prediction techniques to enhance the communication of people with disabilities," *Lecture Notes in Computer Science,* Vol. 3196/2004, p400-417, 2004.

[6] W. Zagler and C. Beck, "FASTY - faster typing for disabled persons," *Proceedings of the European Conference on Medical and Biological Engineering*, 2002.

[7] J. Li and G. Hirst, "Semantic Knowledgee in Word Completion," *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, 2005.

[8] K. Grabski and T. Scheffer, "Sentence Completion," *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, 2004.

[9] S. Bickel, P. Haider, and T. Scheffer, "Predicting sentences Using N-Gram Language Models," *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005.

[10] R. Hecht-Nielsen, "Confabulation Theory: The Mechanism of Thought", *Springer*, Aug. 2007.

[11] J. B. Rotter, M. I. Lah, and J. E. Rafferty, "Rotter Incomplete Sentences Blank Second Edition manual," *New York: Psychological Corporation*, 1992.

[12] Christopher D. Manning, Hinrich Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press: 1999. ISBN 0-262-13360-1.