Using B^+ Tree To Represent Secret Messages For Steganography Purpose

Dr. Suhad M. Kadhem

Received on: 10/1/2010
Accepted on: 3/6/2010

Abstract

In this research we suggest a new approach by using B^+ tree for storing the secret massages (that want to be sent) in a manner that prevent redundancy of these massages or even sub massages in order to provide efficient memory usage. The proposed system includes two stages: Stores the secret message and get its unique code by using B^+ tree, and retrieves the unique secret message when we have its code by using B+ tree. The proposed system was implemented using Visual Prolog 5.1 and after testing, the proposed system gives an efficient time in storing and in retrieving the secret messages, also the proposed method solved the ambiguity problem that some steganography methods suffer from.

Keywords: Steganography, B^+ tree, Information Hiding.

استخدام الهيكل الشجري B^+ لتمثيل الرسائل السرية لأغراض الإخفاء

الخلاصة

في هذا البحث سنقترح طريقة جديدة وهي استخدام الهيكل الشجري B^+ لخزن الرسائل السرية (المراد ارسالها) بطريقة نراعي من خلالها عدم تكرار هذه الرسائل أو حتى أجزاء منها مما يوفر كفاءة في الخزن. يتكون النظام المقترح من مرحلتين وهما: خزن الرسالة السرية واسترجاع الترميز الوحيد لها باستخدام الهيكل الشجري B^+ واسترجاع الرسالة السرية الوحيدة المقابلة لترميزها باستخدام الهيكل الشجري B^+ . تم تنفيذ النظام المقترح باستخدام اللغة البرمجية المرئية Visual Prolog 5.1 ولقد اعطى النظام المقترح بعد اختباره وقت كفوء في خزن واسترجاع الرسائل السرية، كما ان مشكلة الغموض التي تعاني منها بعض طرق الإخفاء قد حلت باستخدام الطريقة المقترحة.

Introduction

Steganography is the technique of hiding information within some format in a way that makes it difficult to detect by one who doesn't know it's there. In the computer age, steganography has become quite advanced and allows for information to be hidden in all types of data files, such as images, audios, videos and documents (text) [1].

In the past, people used hidden tattoos or invisible ink to convey steganographic content. Today, computer and network technologies provide easy-to-use communication channels for steganography. Essentially, the information-hiding process in a steganographic system starts by identifying a cover medium's redundant bits (those that can be modified without destroying that medium's integrity). The embedding

***Computer Science Department, University of Technology/ Baghdad**

https://doi.org/10.30684/etj.28.15.18
2412-0758/University of Technology-Iraq, Baghdad, Iraq

process creates a *stego medium* by replacing these redundant bits with data from the hidden message[2].

One important method of information hiding is constructing the CFG for the secret message and gives it a unique code then this code is embedded in a cover media, but this method is not suitable when the grammar is ambiguous[3]. Thus in our proposed method we will use B^+ tree instead of CFG in order to avoid this problem and to provide efficient time for gating the code or retrieving the secret message. In other words we will use B^+ tree to represent a special dictionary for secret messages such that each message has a unique code, this code used to be imbedded in the cover text, also this code can be used to retrieve the corresponding secret message from this dictionary. So the aim of this research is finding a tool to represent secret messages that can be used for steganography purpose such that:

- Prevent the ambiguity problem such that there is only one secret message correspond to its code.
- Provide efficient search time for gating the code of the secret message.
- Provide efficient search time for retrieving the secret message from its code.
- Provide efficient memory usage in storing the secret messages.

Steganography In Documents

Some methods of steganography use a special tool to convert the secret message to a unique code, and then this code will be embedded into a digital object, also refereed to us the cover, in such a manner that the information (code) becomes part of the object, and the steganography categorized by the type of data that the cover belongs to, such as text or documents, images or sound[4], and in our proposed method we use a text as a cover.

The use of Steganography in documents works by simply adding white space and tabs to the ends of the lines of a document. This type of Steganography is extremely effective, because the use white space and tabs is not visible to the human eye at all, at least in most text/document editors. White space and tabs occur naturally in documents, so there isn't really any possible way using this method of Steganography would cause someone to be suspicious[5].

Using CFG for Steganography Purpose [3, 6]

Build context free grammar (CFG) as a tool of steganography system, such that each secret message will be convert to a set of production rules of CFG and give them a code(this code used to be imbedded in the cover text). For example if the secret message is: "meeting: nine o'clock at my home". Then the corresponding CFG will be:

{ S→ meeting: nine#, 000
A→ o'clock @, 0010
B →at my home, 0011 }

Where '#' represent 'A', '@' represent 'B' and {000, 0010, 0011} represent the code for this secret message.

If the grammar is unambiguous then the receiver can extract the information by applying standard parsing techniques, otherwise he can not extract the information that he want probably.

B^+ Tree[7, 8, 9]

B^+ Tree is a structure of nodes linked by pointers is anchored by a special node called the root, and

bounded by leaves has a unique path to each leaf, and all paths are equal length stores keys only at leaves, and stores reference values in other, internal, nodes guides key search, via the reference values, from the root to the leaves.

B^+ tree is called an index to database, such that each record will be stored in the database, the reference number (and the key) of that record will be stored in the B^+ tree. So when we want to reach a certain record, we need to know its key to get its reference number from the B^+ tree. When we get the reference number of that record we can retrieve the required record directly.

B^+ tree is an arranged and balanced tree (see figure 1), and this is why it is so fast in retrieving the required data.

Description of the Proposed Method

In this research we suggest a new approach by using B^+ tree as a special dictionary for storing the secret massages (with their codes) in a manner that prevent redundancy of these massages or even sub massages in this dictionary (in order to provide efficient memory usage). So the proposed method includes two stages:

- Store the secret message in this dictionary (if it is not found) and get its unique code (at send process).
- Retrieve the unique secret message when we have its code from this dictionary (at received process).

The proposed method uses one database(dbase) that represent the dictionary of the secret messages and its corresponding codes, and uses two index trees (Bt1, Bt2) that refer to the same database (dbase). Each new secret message (sentence) will be converted to a list of words (with discarding to any un useful letter) and these words will be stored in dbase in a manner that prevent the redundancy of these sentences or even sub sentences.

Bt1 is used for storing purpose to check if the sentence or even sub sentence is already found in dbase. So Bt1 use the first word of the sentence as a key, while Bt2 is used for retrieving purpose, so it uses the code of the sentence (secret message) as a key. In general if the secret message is composed of [$word_1$, $word_2$, …, $word_n$] then it will be represented as shown in figure 2. The arrows in the figure represent the database references, we follow the down arrow (↓) in send process of steganography to get the code for the secret message, and we follow the above arrow (↑) in received process of steganography to retrieve the secret message that correspond to a certain code.

If we want to get the code of a secret message (sentence) from the dictionary (dbase), first we must check if the first word (key) of the sentence is found in Bt1 then we have one of the following cases:

1. All the words of the sentence are found in dbase, and in this case we have:
   - The sentence already has a code, and in this case we need only to retrieve the stored code.
   - The sentence has no code, and in this case we give it a new code and store it in dbase and in Bt2 as a key used for received process of steganography.
2. All the words of the sentence are found in dbase except the last word, and in this case we store the last word in dbase with its reference of previous word and give it a code and store it in Bt2, also the reference of the last word will be stored with

the references of next words for previous word.

3. If the first words of sentence (or even sub sentence) are found in dbase, then the remaining words will be added to the dbase as a new sentence with storing the previous and next references of them.

If the secret message is not found in dbase (new sentence), then we will add it to the dbase and store the first word of the message as a key in Bt1, and give it a new code and store it in Bt2 as a key, and each time we need to add a new word of a sentence we check if it is found in Bt1 such that the middle words are found in dbase, so we need not to restore it to prevent redundancy.

Knowledge Representation

We represent knowledge as compound of the logical terms:

word (Termlist)

Where:

word: predicate name.

Termlist: list of logical term:

p(W,NRef,PRef,N,X)

Where:

p: predicate name.

W: string represents the word.

NRef: database reference number for next words.

PRef: database reference number for previous words.

N: integer value represents the length of secret message.

X: integer value represents the Code of secret message.

Algorithms of the Proposed Method

The proposed method includes two stages:

- *Send process.*
- *Received process.*

Since Our paper is focus on using B^+ tree with steganography, so we will produce the algorithms of the functions that deal with B^+ tree only.

Send process

In this stage we need to get an integer value that represents the code of the secret message from B^+ tree (see algorithm1), and then convert this integer value to the binary code, after that this binary code was embedded in a cover media (text) by using open space method, such that single space encodes as zero and two spaces encodes as one.

Algorithm1: "get_code"

Input: secret message.

Output: code for the secret message.

Process:

Begin

1. If the file "*dbase*" exist then open dbase, *Bt1* and *Bt2* otherwise create them.
2. Convert the secret message to a list of words (*List1*);
3. Search in (*Bt1*) for the first word(*W1*) of *List1*;
4. If(not found) then

4.1.Call *add_new_sentence* function to get Ref; /* see algorithm 2 */

4.2.Insert the key(W1) in *Bt1* with the reference of*dbase* (Ref);

Else

4.3.Get reference of dbase(Ref) for the key (W1);

4.4.Call *check_found_sentence* function to get the code X; /* see algorithm 4 */

5. Get the new code (X);
6. Return(X) ;

End.

Algorithm2:"add_new_sentence"

Input:list of words (List1).

Output:database reference number.

Process:

Begin

1. Remove the first word (*W*) from *List1*;

2. If (List1==[]) then
2.1. Compute the length of *List1* to be *N*;
2.2. Get the new code to be X1;
2.3. Insert the term *word([p(W,~0,~0,N,X1)])* to *dbase* at *NRef*;
2.4. Insert the key(*X1*) in *Bt2* with the reference of *dbase* (*NRef*);
2.5. Return(*NRe*f);

Else
2.6. Call *check_add* function to get *NRef*; /* algorithm3 */
2.7. Insert the term *word([p(W,NRef,~0,0,0)])* to *dbase* at *Ref*;
2.8. Let *Ref* to be the previous reference;
2.9. Return (Ref);

End.

Algorithm3: "check_add"

Input: list of words (List1).

Output: database reference number (NRef).

Process:

Begin
1. Search in *Bt1* for the first word(*W1*) of *List1*;
2. If(not found) then
2.1. Call *add_new_sentence* function to get *Nref*;

Else
2.2. Get reference of dbase(*NRef*) for the key (*W1*);
2.3. Call *check_found_sentence* function; /* take *List1and Ref* as input */
3. Return(NRef);

End.

Algorithm4:"check_found_sentence"

Input: list of words (List1), and the database reference number (Ref).

Process:

Begin
1. Retrieve the term of *Ref* from *dbase* to be *word(Termlist)*;
2. Remove the first word (*W*) from *List1*;
3. If (List1==[]) then
3.1. If all the words of List1 is found in dbase except the last word then
3.1.1. Get the length of List1 to be N;
3.1.2. Get the new code to be X1;
3.1.3. Get the previous reference to be Pref;
3.1.4. Add the term *p(W,~0,Pref,N,X1)* to the *Termlist* to get *Termlist1*;
3.1.5. Replace the term of *Ref* with the term *word(Termlist1)* in *dbase* ;
3.1.6. Insert the key(X1) in *Bt2* with *Ref*;

Else
3.1.7. If all the words of List1 is found in dbase and the sentence has a code X1 then
3.1.7.1. Let *X1* be the code;

Else
3.1.7.2. If all the words of List1 is found in dbase but the sentence has no code then
3.1.7.2.1. Get the length of *List1* to be *N*;
3.1.7.2.2. Get the new code to be *X1*;
3.1.7.2.3. Replace the term of *Ref* with a new one that have the code X1 and the length N;
3.1.7.2.4. Insert the key(X1) in *Bt2* with *Ref*;

Else /*if List1 is not empty*/
3.2. If the current word is found in dbase but the next word is not found then
3.2.1. Call *add_new_sen*tence function to get *Ref1*; /* add the remaining words of List1 to dbase*/

3.2.2. Replace *p(W,_,Pref,_,_)* with *p(W,Ref1,Pref,0,0)* in *Termlist* to get *Termlist1*;
3.2.3. Replace the term of *Ref* with the term *word(Termlist1)* in *dbase*;

Else
3.2.4. Let the current reference (Ref) be the new previous reference(Pref);
3.2.5. Let the reference of next word(Nref) be the new current reference(Ref);
3.2.6. Retrieve the term of *Ref* from *dbase* to be *word(Termlist)*;
3.2.7. Goto 2;

End.

Received process

In this stage, the code of the secret message need to be extracted from the cover text, and since we use the open space method, the single space decoded as 0 while two spaces decoded as 1. After that, this binary code must be converted to its integer value, then the Retrieve-message function is used to get or retrieve the unique secret message according to this code from the B^+ tree(see algorithm5).

Algorithm5: "retrieve-message"
Input: the code(C).
Output: secret message(S).
Process:
Begin
1. Search in *Bt2* for the key(*C*) and If *found* then
1.1. Get its reference number (*Ref*) of *dbase*;
1.2. Retrieve the term of *Ref* from *dbase* to be *word(Termlist)*;
1.3. Find the term *p(S,_,PRef,N,C)* of *Termlist*; /* that contains the code *C* */
1.4. N1=N-1;
1.5. While (N!=0) do
Begin
1.5.1. Retrieve the term of *PRef* from *dbase* to be *word(Termlist1)*;
1.5.2. Find the term *p(W1,Ref,PRef1,_,_)* of *Termlist*1; /* that contains the reference of next words*Ref* */
1.5.3. Concatenate the string *W1* with *S* to get the new S;
1.5.4. Ref:=PRef;
1.5.5. PRef=PRef1;
1.5.6. N=N-1;
End;
1.6. Return(S);

End.

Implementation For The Proposed Method

We will take some examples in order to describe our proposed method.

Example1:

If we want to get the code for a secret message (sentence) from the dictionary (dbase), and suppose it is anew sentence, such as: **"meeting: nine o'clock today"**, then we put the first word of the sentence (meeting) as a key in b^+ tree (Bt1), and we compute the length(4) of the sentence, also we give the sentence a new unique code (101) and use this code as a key for b^+ tree (Bt2), and the sentence will be stored in dbase as shown in figure (3).

Example2:

If we want to get the code for a sentence, such that all its words are already found in dbase but with no code, such as: **"meeting: nine o'clock"**, then we give it a new code (102), and store it in Bt2 as a key, it will be as shown in figure (4).

Example3:

If we want to get the code for a sentence, such that all its words are already found in dbase except the last word, such as: **"meeting: nine o'clock tomorrow"**, then we will

store the last word in dbase and the reference of the previous word, and store at the previous word the new reference, and give it a new code (103), it will be as shown in figure (5).

Example4:

If we want to get the code for a sentence, such that some of its words are already found in dbase, such as: **"meeting: nine o'clock tomorrow at my home"**, then we will store the remaining words in dbase and the references of the next and previous word and give it a new code (104), it will be as shown in figure (6).

Example5:

If we want to get the code for a sentence is already found in dbase but with no code, such as: **"meeting"**, then we give it a new code (105), and store it in Bt2 as a key, it will be as shown in figure (7).

Example6:

If we want to get the code for a sentence, such that, some of its middle words are found in dbase, such as: **"today meeting: nine o'clock"**, then only the not found words will be store in dbase, and we will store the first word in Bt1, and store the new code (106) in Bt2, it will be as shown in figure (8).

Example7:

If we want to get the code for a sentence, such that it is already found in dbase and have a code, such as: **"meeting: nine o'clock"**, then we retrieve its code(102) only.

At the received process, the receiver extract the code from the cover text, he uses the same dictionary in order to retrieve the secret message that correspond to that cod.

If we take the code (101) as an example, First we will check if the code is found in Bt2, then we retrieve the term of the last word of the message that the code refer to it: **word([p("today" ,~0,0011,4,101),p("tomorrow",0101, 0011,4,103]),** then we will search in the list of this term on the code (101) and get: **p("today" ,~0,0011,4,101)**, this mean that the length of the sentence is **4** and have "**today**" as the last word, and the reference of the previous word is **0011,** after that we follow the reference of the previous word, and take its word and concatenate it with next word, then we follow the reference of its previous word and so on, until we get the sentence **"meeting: nine o'clock today"** that composed of four words.

Since there is only one term that corresponds to a reference, we can retrieve one secret message and only one, so we solved the ambiguity problem that some steganography methods suffer from.

Discussion

Some methods used to build a dictionary for secret sentence (with their codes) as a production rules. Table (1) will compare between this method and our proposed method.

Conclusions

In this paper the following points can be concluded:

- Flexibility of the B^+ tree gives good result to build many meaningful messages and builds a special dictionary.
- There is no ambiguity in retrieving secret message from its code (there is one and only one secret message correspond to its code) when we use B^+ tree.
- Using B^+ tree for storing and retrieving the secret sentences will increase the steganography system efficiency.
- Prevent the redundancy of messages in the dictionary or even the sub

messages will provide efficient memory usage.

- Using B^+ tree for building a dictionary used for steganography purpose will provide an efficient search time for gating the code of secret message from this dictionary and provide an efficient search time for retrieving the secret message that corresponding to its code from this dictionary.

References

[1] Krenn, R., "**Steganography Implementation & Detection**", 2004, www.krenn.nl/univ

[2] Niels, P. Peter, H., "**Hide and Seek: An Introduction to Steganography**", IEEE Security & Privacy, 2003, www.**pdf**-search-engine.com

[3] Hala, B. Abdul Wahab, "**Information Hiding in Written Text Using Context-Free Grammar (CFG)**", MSc. Thesis, university of Technology, Computer Science department, Iraq, 2001.

[4] Christian, G. Krista, G. Ludmila A. Ryan, S. Mikhail A. "**Translation-Based Steganography**", Information Hiding: 7th international Workshop, Springer, 2005.

[5] Aelphaeis Mangarae, "**Steganography FAQ**", Zone-H.org, 2006.

[6] Katzenbeisser S. Petitcolas F. (eds), "**Information Hiding Techniques for Steganography and Digital Watermarking**", Artech House, USA, 2000.

[7] Jan Jannink ,"**Implementing Deletion in B+-Trees**", Sigmod Record, Vol. 24, No.1, 1995.

[8] Goetz, Graefe ,"**B-tree indexes, interpolation search, and skew**", Chicago Illinois, USA, 2006.

[9] Anderson, S. ,"**B+ Trees**", Freed, 1998. http://babbage.clarku.edu/~achou/cs160/B+Trees/B+Trees.htm

Table (1): comparison between using CFG and the proposed method for steganography purpose.

Using B^+ tree	Using CFG
Increase the number of secret messages has no effect because B+ tree dealing with huge data.	Increase the number of secret messages Means increase in program size.
Ambiguity problem solved.	Deal with unambiguous grammar only.
Provide an efficient search time for gating The code of the secret message.	Search time for gating the code of the secret message will be increased when secret messages increased.
Provide an efficient search time for retrieving the secret message from its code.	Search time for retrieving the secret message from its code will be increased when secret messages increased.

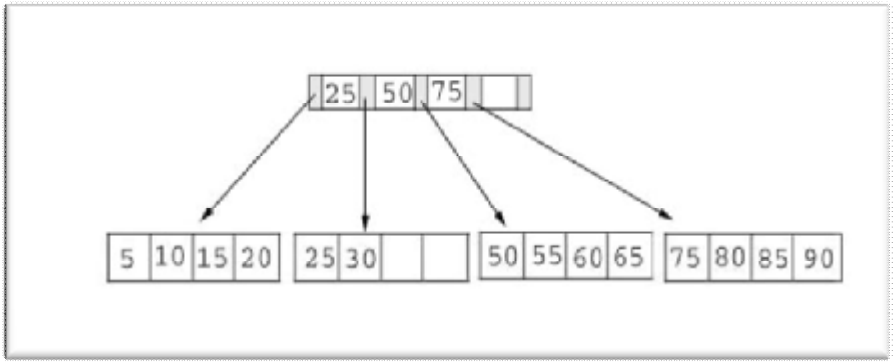

Figure (1) An example of B^+ tree

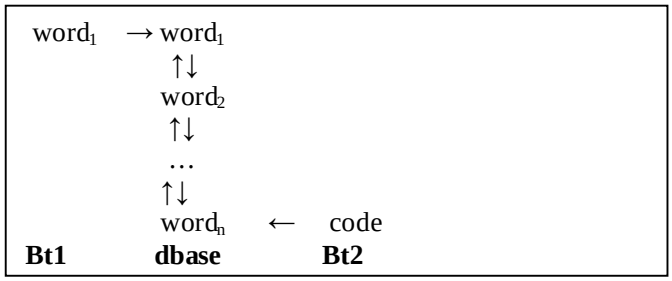

Figure (2) show how secret message is represented

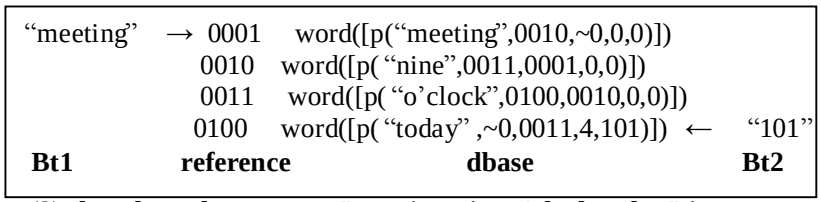

Figure (3) show how the message "meeting nine o'clock today" is represented

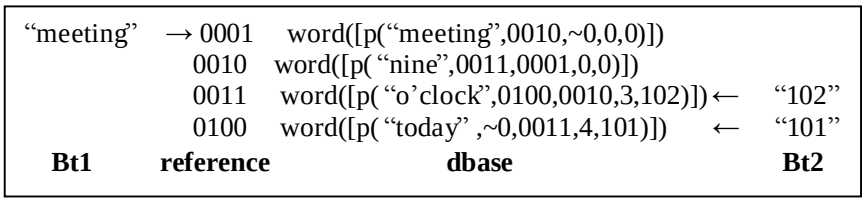

Figure (4) show how the message "meeting nine o'clock " is represented

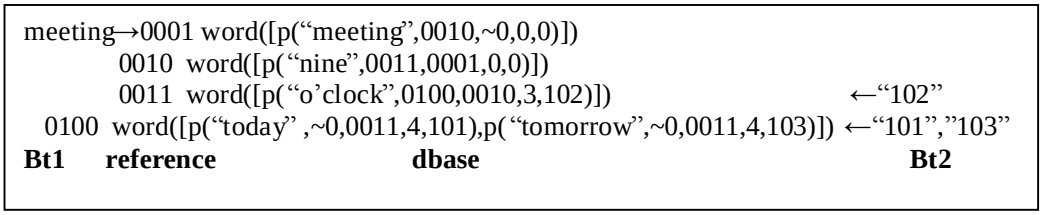

Figure (5) show how the message "meeting nine o'clock tomorrow" is represented

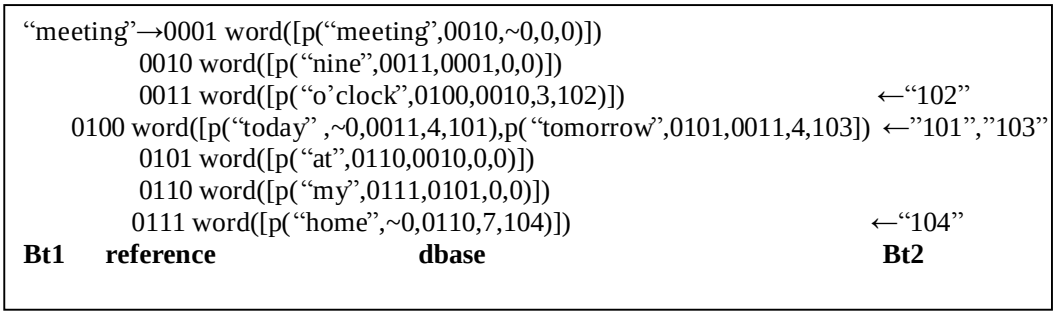

Figure (6) show how the message "meeting nine o'clock tomorrow at my home" is represented

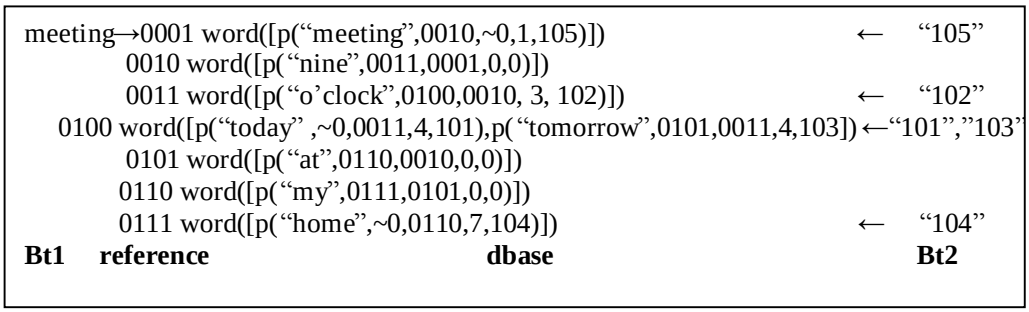

Figure (7) show how the message "meeting" is represented

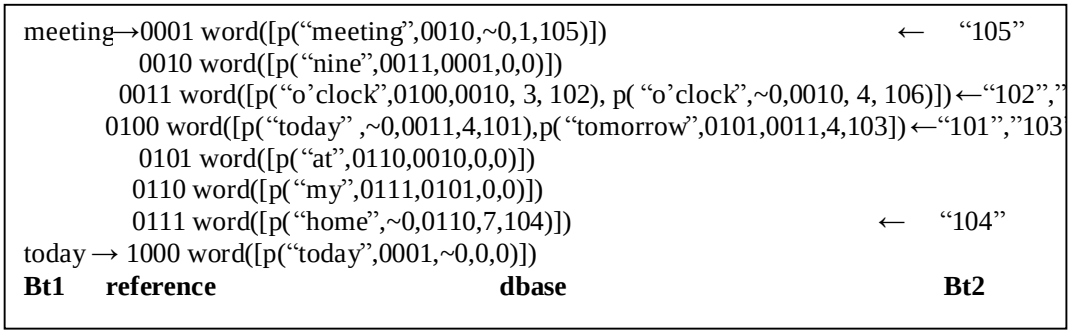

Figure (8) show how the message "today meeting nine o'clock " is represented