A hierarchical graph method using in A* algorithm for Vietnamese parsing technique

Le Quang Thang
MICA Institute
HUST - INPG - CNRS 2954
E-mail: quang-thang.le@mica.edu.vn

Tran Do Dat
MICA Institute
HUST - INPG - CNRS 2954
E-mail: Do-Dat.Tran@mica.edu.vn

Abstract **– this paper presents our research about hierarchical graph method (HGM) in the classical A* searching algorithm for parsing technique. HGM is used in new-node generation step of A* parsing algorithm. Unlike classical virtual node method [4], HGM does not create any redundancy *node* in the parsing process and reduce the timing-cost for finding the best parse. This method can easily control the complex grammar productions (rules).**

Keywords – A*, parsing technique, HGM, algorithm, Vietnamese.

I. Introduction

Probabilistic context free grammar (PCFG) and lexical probabilistic context free grammar (LPCFG) are very well-known models in parsing technique. In both of them, the candidate with highest probability is the best result. With these models, especially LPCFG, the accuracy of parsing system is relatively high[3]. However, when dealing with wide-coverage grammars and very long sentence, the parsing process is very complicated and the cost for processing time is expensive. To solve this problem, many speedy searching algorithms have been researched to reduce the work, such as Beam Search algorithm [8], Greedy algorithm [6], and Dijkstra algorithm [7]. However, these algorithms still have limitations. Beam Search uses a beam to remove the underrated candidates, so it is not guaranteed to find the best result. The Greedy algorithm only follows the best path in each step, so it got a very fast parsing time but it will not be guaranteed to find the best result. The Dijkstra algorithm will find the best result, but its speed, in many cases, may be slow.

A* parsing algorithm which was proposed by Dan Klein and Christopher D.Manning[1] could solve both problems: best result and speed.

There are some searching algorithms which have been researched and developed in Vietnamese parsing like Beam Search, Greedy algorithms... However, in our knowledge, there is no research about A* algorithm.

In this paper, we present two main parts. The first part presents briefly about A* parsing algorithm. The second major part, which is a mainly focus of our research, presents about the hierarchical graph method, denoted as HGM. This method is a replacement for the classical virtual node method in order to reducing the complex for parsing process. Therefore, the speed of A* parsing algorithm could be improved.

II. A* Algorithm for Parsing

In order to making this paper easier to follow, two abbreviations are used:

G – The grammar productions. Each production in G has a corresponding weight *w*.

POS – A unique tag that indicates its *syntactic role*, for example, plural noun, adverb.

The training corpus of our parsing system is Viet Treebank database from VLSP project [8]. It includes about 10.000 Vietnamese sentences which were manually parsed. From this corpus, we have extracted the grammar productions G (or the grammar rules). Each production has the weight parameter which is calculated by appearing probabilistic of production in G.

A. General A* algorithm

A* algorithm which belongs to the Best-First-Search algorithm group is considered as one of the best searching algorithm in the world. It uses a heuristic f(x) to determine the best candidate for its each step [5]:

$$f(x) = g(x) + h(x)$$

In which:

- g(x) - the path-cost function, which is the cost from the starting *node* to the current *node*.
- h(x) - an admissible "heuristic estimate" of the distance to the goal.

And h(x) figure is very important; it determines how fast the parsing process leads to the target.

B. Basic concept

A* algorithm operates on items called as "*node*". A *node* includes three attributes: *name, start,* and *end. Name* attribute indicates the POS of *node*. And the attribute couple (*start, end*) is the start and end position of the text which is covered by *node* in the input string. Its format is *name*[*start, end*].

The parsing system maintains two data structures: a chart (note as CHART) which records *nodes* for which (best) parses have already been found, and an agenda of newly-formed *nodes* needs to be processed (note as AGENDA)[1].

The initial CHART is empty, the input string is tokenized into *n* words $a_1 \ldots a_n$. And then, these words are POS tagged to create an initial AGENDA: $\{(X_i\ [i,\ i+1],\ w_i), i = \overline{1,n}\}$, w_i is an initial weight of each tagged word X_i.

A context for *node* $X[i,j]$ (with input string $a_1 \ldots a_n$) is a parse tree whose leaf nodes are labeled as $a_1 \ldots a_{i-1}X$

a_{j+1}...a_n. The weight of a context is the sum of the weights of the productions appearing in the parse tree. A function *h(X[i,j])* - a real number will be called an admissible heuristic for parsing if *h(X[i, j])* is a lower bound on the weight of any context for *X[i, j]*.

C. A* parsing process

While AGENDA is not empty and CHART does not contain S [1, n+1] (the goal of parsing process):

Remove a candidate node (Y[i,j],w) with highest w + h(Y[i,j]) from AGENDA

If CHART does not contain Y[i,j] then:

** Combine Y with CHART*

(New-node generation step)

- For each *node* (Z[j,k],w') in CHART where G contains a production $X \xrightarrow{w''} Y\boldsymbol{Z}$, add the *node* (X[i,k], w+w'+w'') to AGENDA.
- For each *node* (Z[k,j], w') in CHART where the G contains a production $X \xrightarrow{w''} \boldsymbol{Z}Y$, add the *node* (X[k,j],w+w'+w'') to AGENDA.

** Add (Y[i,j],w) to CHART*

Finally, if AGENDA contains an assignment to S[1,n+1] then the parsing process is successful (a parse has been found) else terminate with failure (there is no parse).

III. HIERARCHICAL GRAPH METHOD (HGM)

A. The context for proposition

In new-node generation step of A*, here are two situations that happened when combining candidate *node* with CHART:

The relevant production is a Chomsky-form, means that it has *less-than or equal to* two elements on the extension part. In this case, the combination will perform normally.

The relevant production is not a Chomsky-form; it has more than two elements on the extension part. In this case, the parser uses a virtual node method (VNM) with the *wait* parameter which denotes the *POS tags* which are remained to complete the production.

For example, A *node* combine with B *node* using a production like "E→A|B|C|D" will form a virtual *node* E[wait = "CD"]. Later, if the virtual node E[wait="CD"] meets C *node*, the *node* E[wait="D"] will be formed.

After the parsing process ends, the successful of parsing process will be determined if the *node* S[1,n+1,wait=""] is founded in CHART.

The problem is that the cost of VNM is too expensive to deal with. Because of the huge grammar productions, the combination using VNM will generate a large quantity of virtual *nodes.*

B. The proposed model

1) The basic idea

HGM uses *combinable chain* to overcome the *Chomsky production*. A *combinable chain* is a *node* array which has the increasing continuous position. In HGM model, all the combinable chains of a candidate *node* and CHART will be processed in new-node generation step of A* algorithm.

For instance, the candidate *node* is X[7,10] and the content of CHART has been shown in Table 1.

TABLE I. THE *NODES IN* CHART

X_1[1,8]	X_2[6,16]	X_3[13,35]	X_4[5,20]	**X_5[2,7]**	**X_6[10,11]**
X_7[8,27]	X_8[2,21]	X_9[9,11]	X_{10}[2,13]	X_{11}[6,14]	X_{12}[15,26]
X_{13}[14,23]	X_{14}[5,18]	**X_{15}[1,7]**	X_{16}[9,16]	X_{17}[12,17]	X_{18}[7,18]
X_{19}[6,25]	**X_{20}[13,26]**	**X_{21}[11,26]**	X_{22}[9,24]	**X_{23}[11,20]**	X_{24}[8,18]
X_{25}[7,16]	X_{26}[14,16]	X_{27}[4,6]	**X_{28}[13,21]**	X_{29}[4,8]	**X_{30}[11,13]**

From this input data (candidate and CHART *nodes* position), the HGM will process and use the combinable chains which are presented in Table 2:

TABLE II. THE COMBINABLE CHAINS OF THE CANDIDATE WITH CHART *NODES.*

1	Position	*node*
2	[2-7] **[7-10]**	X_5**X**
3	[1-7] **[7-10]**	X_{15}**X**
4	[1-7] **[7-10]** [10-11] [11-13]	X_{15} **X** X_6 X_{30}
5	[2-7]**[7-10]**[10-11] [11-13] [13-26]	X_5 **X** X_6 X_{30} X_{20}
6	**[7-10]** [10-11] [11-20]	**X** X_6 X_{23}
7	**[7-10]** [10-11] [11-13]	**X** X_6 X_{30}
8	**[7-10]** [10-11] [11-13] [13-26]	**X** X_6 X_{30} X_{20}
9	**[7-10]** [10-11] [11-13] [13-21]	**X** X_6 X_{30} X_{28}
10	[2-7] **[7-10]** [10-11]	X_5 **X** X_6
11	[2-7] **[7-10]** [10-11] [11-16]	X_5 **X** X_6 X_{21}
12	[2-7] **[7-10]** [10-11] [11-20]	X_5 **X** X_6 X_{23}
13	[2-7] **[7-10]** [10-11] [11-13]	X_5 **X** X_6 X_{30}
14	**[7-10]** [10-11] [11-16]	**X** X_6 X_{21}
15	[1-7] **[7-10]** [10-11]	X_{15} **X** X_6
16	[2-7] **[7-10]** [10-11] [11-13] [13-21]	X_5 **X** X_6 X_{30} X_{28}
17	[1-7] **[7-10]** [10-11] [11-16]	X_{15} **X** X_6 X_{21}
18	[1-7] **[7-10]** [10-11] [11-20]	X_{15} **X** X_6 X_{23}
19	**[7-10]** [10-11]	**X** X_6
20	[1-7] **[7-10]** [10-11] [11-13] [13-26]	X_{15} **X** X_6 X_{30} X_{20}
21	[1-7] **[7-10]** [10-11] [11-13] [13-21]	X_{15} **X** X_6 X_{30} X_{28}

Thus, assuming that there is a "A→X_5|**X**|X_6|X_{30}|X_{28}" production relevant to the 16^{th} chain in the table 3, the *node* A[2,21] will be formed and will be added to AGENDA.

2) HGM combinable-chains generating model

HGM combinable-chains generating model shows how to process through all the combinable chains between candidate X and CHART.

This model includes two phases: *classification phase* and *combinable chains generation phase*.

Classification phase (CP)(1): the parser classifies the *nodes* in CHART based on the candidate *node* X.

Combinable chains generation phase (CGP)(2): the parser generates all the combinable chains and uses them to create a new *node* which is added into AGENDA.

a) Classification phase

This phase will use the "*hole"* conception. *Hole* is an array of *node* which is grouped together.

The *nodes* in the CHART are divided into two types: the *left holes* and the *right holes*. Let assuming that X is a candidate *node*.

- *Left holes of X*: This is a set of *hole* which has *node* positions are on the left of X position in the real number line. All the *nodes* which are in a *hole* have the same *end (or right)* position.
- *Right holes of X*: Resemble to *left holes* but on the right side of X.

b) Combinable chains generation phase

With the input as the classified CHART, the parsing system begins generating the combinable chains. This phase includes three steps: "generating *left chains*", "generating *right chains*" and "*generating combinable chains*".

1. Generating left chains: this step generates an array of *n* combinable chains which ends with candidate X; it is called as LC[n] - the *left chains*.

We imply that S(E) is the block in a ***left hole*** which have the end position equals to E start position.

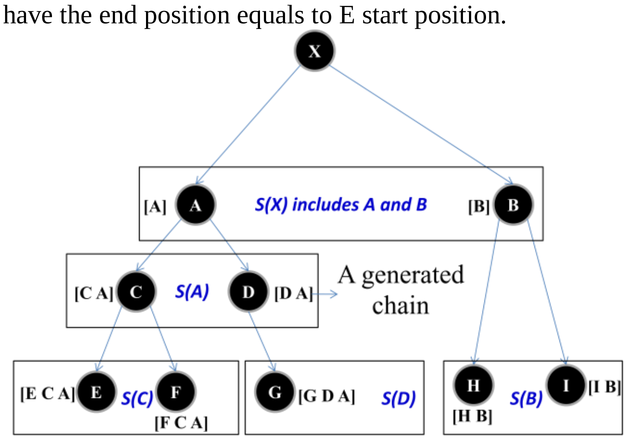

Figure 1 – The instance example for generating left chains.

This task can be described as below:

- Parsing system processes the X *node*, saves a generated chain to *left chains* and gets the S(X) from *left holes*.
- This progress is done recursively for all the *nodes* in the S(X).

2. Generating right chains: the same progress as the "generating *left chains*" but on the opposite position (right position) with the input is the *right holes*. We got *right chains* – RC[m] which is an array of *m* combinable chains which starts with candidate X.

3. Generating combinable chain: with the array of left chains – LC[n] and the array of right chains – RC[m], we have the array of the combinable chains of X:

(LC[i], $\overline{i=1,n}$)(X)(RC[j], $\overline{j=1,m}$)

C. Pruning graph in HGM model

As mentioned above, HGM model is proposed in order to improving the speed of parsing system, to reduce the number of *node* in parsing process. However, the HGM model is still not optimal because of the combinable chain redundancy.

From our experiment on testing performance of HGM model, we found that there are approximately 8% of the combinable chains that would be used. Thus, HGM model is not only slower than virtual node algorithm in some case, but also got stuck when the number of CHART is high.

To solve this problem, HGM model uses a pruning graph. Instead of processing all the combinable chains, the parser will use this graph to prune the redundancy combinable chains; it means that they are not relevant to any production in G. This algorithm is not only increasing the speed of parser but also reduce the complication of parsing process.

1) The idea

A *node* has its position and POS. The basic HGM model only uses the *node* position to generate chain, but the POS of *node* is not used.

For instance, if we have two nodes: NP[1,7] and PP[1,7]. In the basic HGM model, they are just the same, even their POS is different. So, a pruning graph in HGM model is proposed to use the POS in order to reducing the processing time of HGM.

The algorithm using pruning graph in HGM model includes two phases:

- *Statistic training phase*: create a pruning graph with the training corpus is G.
- *Pruning phase*: integrate pruning graph into HGM model.

2) Statistic training phase

The training data of the HGM pruning graph is G. With each "T" POS in G, the system creates two graphs:

a) The left-POS graph

The left-POS graph of "T" POS is a graph which stores the information about the POS being left-adjacent to "T" POS in G.

The algorithm for creating left-POS map:

- Process all the productions in the graph.
- For each production whose extension part likes [P_n … P_2 P_1 T…]:
  - If P_1 is not child of T, then add P_1 into T children.
  - For i from 2 to n: if P_i is not child of P_{i-1}, then add P_i into P_{i-1} children.

b) The right-POS graph

The right-POS graph of "T" POS is a graph which stores the information about the POS being right-adjacent to "T" POS in G.

The creating algorithm of right-POS map:

- Process all the productions in the graph.
- For each production whose extension part likes […T P_1 P_2 … P_m]:
  - If P_1 is not child of T, then add P_1 into T children.
  - For i from 1 to m: if P_i is not child of P_{i-1}, then add P_i into P_{i-1} children.

3) Pruning phase

The HGM model uses the combinable chain to overcome the Chomsky problem. For each loop step, the candidate *node* from AGENDA combines with the *nodes* in CHART through three major steps: "Generating left chains", "Generating right chains" and the collection part of those two "Generating combinable chains".

The HGM process will perform normally with the support of pruning graph. In each new-node generation step, the system gets pruning graph for POS of candidate *node* and uses it to prune the bad combinable chains (cannot lead to any production). With an X *node*, if S(X) contains any *node* whose POS is not contained in "X" POS children in pruning graph, they will be pruned. The figure 2 is an illustration of pruning graph in the PGHM model.

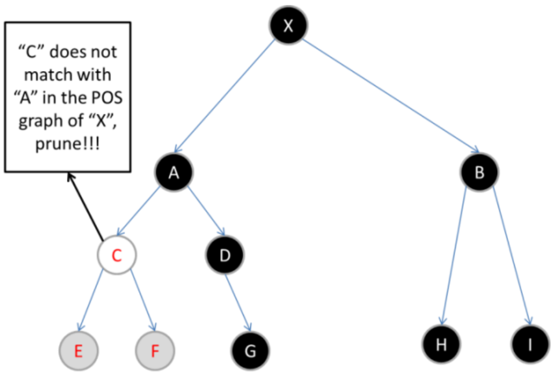

Figure 2 – the illustration of the HGM model using pruning graph.

IV. EXPERIMENT AND RESULT

This section presents the obtained experiment result of the performance of A* parsing algorithm using HGM model.

A. Preparation for experiment

Two hundred sentences randomly selected from Viet Treebank are used to test the performance of our proposed method.

TABLE III. TESTING CORPUS FOR EXPERIMENT

group	5 tokens	10 tokens	20 tokens	30 tokens	40 tokens	50 tokens
quantity	20	60	50	25	25	20

The purpose of this experiment is to test the speed performance of HGM model. We make a testing race between three candidates: HGM model, virtual node method (VNM), HGM model using pruning graph (HGM-PG). The testing corpus which is used for this experiment is 200 sentences from VLSP corpus as described above. The testing set is grouped by approximate number of its tokens. We have 6 groups: sentences with 5 tokens, 10 tokens, 20 tokens, 30 tokens, 40 tokens and 50 tokens. From this test, we will compare the speed of three candidates and evaluate the result.

B. Results

The result of the experiment is shown in table 4. With the sentences has less than 20 tokens, HGM, VNM and HGM-PG got the same speed. When the number of tokens up to 50, the HGM and VNM got an explosion of processing time, but the PGHM-PG speed is still stable. The reason is that PGHM-PG does not make any redundancy like PGHM and VNM. When the number of tokens increases, the quantity of the redundancy things is much more than the quantity of the useful things. Because of that, the more complex the sentence is, the better performance of PGHM-PG is when comparing to VNM.

TABLE IV. EXPERIMENT RESULT

Group	HGM	VNM	HGM-PG
5 tokens	2.741s	2.678s	2.352s
10 tokens	5.027s	4.699s	4.312s
20 tokens	18.142s	17.436s	8.326s
30 tokens	206.341s	169.786s	36.313s
40 tokens	808.936s	433.582s	64.488s
50 tokens	2477.436s	923.535s	120.724s

Figure 3 shows the visual illustration of the experiment. X-axis presents number of tokens in the input string; y-axis is an average processing time measured in second.

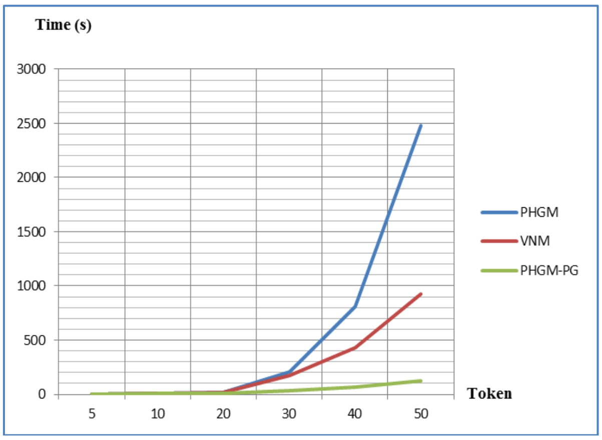

Figure 3 – result of the speed experiment between A HGM, A* VNM and A* HGM using pruning graph*

However, the speed of our parsing system is still relative slow, because of these following reasons:

- A* "*admissible heuristic*": the A* heuristic in our parsing system is a *grammar projection estimates* which has been proposed for a long time [1]. There are many more-advanced heuristics which were published. May be we will study and present about it in our future research.
- The grammar productions G: our grammar productions have been extracted from VLSP corpus. It is still in rude form and need to be refined. With the well-refined grammar productions, the performance of parsing system will be much more improved, too.

V. CONCLUSION AND FUTURE WORKS

This paper have described our HGM method based on A* algorithm to improve the speed of Vietnamese parsing system. Through the experiment, the speed performance of HGM is relatively good and acceptable. The

Vietnamese parsing system has been implemented in JAVA to evaluate our method.

In the future, we are going to research about A* algorithm and find the better "admissible heuristic" than the current one to improve the speed performance of parsing system. We also do the research on the way to integrate lexical information into our algorithm. This will significantly increase the speed and the accuracy of parsing system.

VI. REFERENCES

[1]. **Dan Klein and Christopher D. Manning**. 2003. "*A* parsing: Fast exact Viterbi parse selection*". In Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL), pages 119-126.

[2]. **LAM DO BA, HUONG LE THANH**. 2008. "*Implementing a Vietnamese Syntactic Parser Using HPSG"*. In Proceedings of the International Conference on Asian Language Processing (IALP), Nov. 12-14, 2008, Chiang Mai, Thailand.

[3]. **MICHAEL COLLINS**. 1997. *"Three generative, lexicalized models for statistical parsing"*. In Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics, pages 16-23.

[4]. **DONG DONG ZHANG, MU LI, CHI H. LI, MING ZHOU.** 2007. "Phrase Reordering Model Integrating Syntactic Knowledge for SMT". In Proceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), pp. 533-540.

[5]. **LI-YEN SHUE, REZA ZAMANI.** 1993. "*An Admissible Heuristic Search Algorithm*". ISMIS '93 Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems, pages 69-75.

[6]. **CORMEN, LEISERSON AND RIVEST.** 1990. *"Introduction to Algorithms"*. Chapter 17 "Greedy Algorithms" p. 329.

[7]. **MIT PRESS, MCGRAW-HILL**. 2001. "*Introduction to Algorithms"* (Second edition). pp. 595–601. ISBN 0-262-03293-7.

[8]. **HUONG LE THANH, LAM BA DO, NHUNG THI PHAM.** 2010. *"Efficient Syntactic Parsing with Beam Search".* Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010 IEEE RIVF International Conference on. ISBN: 978-1-4244-8074-6.

[9]. **Website:**
http://vlsp.vietlp.org:8080/demo/?&lang=en