**A pruning hierarchical graph method using in A\* algorithm for Vietnamese parsing**

Le Quang Thang

MICA Institute

HUST - INPG - CNRS 2954

E-mail: [quang-thang.le@mica.edu.vn](mailto:quang-thang.le@mica.edu.vn);

Tran Do Dat

MICA Institute

HUST - INPG - CNRS 2954

E-mail: [Do-Dat.Tran@mica.edu.vn](mailto:Do-Dat.Tran@mica.edu.vn)

*Abstract* — this paper presents our research on pruning hierarchical tree method (PHGM) in A\* (A-star) algorithm in Vietnamese parsing technique in order to improve the speed of Vietnamese parsing system. Based on the virtual node method proposed in [ref], we will describe our replace method: pruning hierarchical tree method. Unlike the virtual node method, PHGM processes only significant candidates and does not generate new redundancy candidates for each step of A\* algorithm. With this method, the speed of parsing system could be improved so much.

Keywords – A\*, parsing technique, PHGM, algorithm, Vietnamese

# Introduction

### [Probabilistic context free grammar](http://www.cs.utexas.edu/%7Emooney/cs388/slides/stats-parsing.ppt) (PCFG) and lexical probabilistic context free grammar (LPCFG) are very well-known models in parsing techniques. In the parsing system using these models, the final result is determined based on the score of candidates. The candidate with highest score will be the predicted outcome. With these models, especially LPCFG, the accuracy of parsing system is relatively high [TLTK]. However, when dealing with wide-coverage grammars and very long sentence, the parsing process is very complicated and the cost for processing time is too expensive [TLTK]. To solve this problem, many speedy algorithms have been researched to reduce the work, such as Beam Search, Greedy algorithm, and Dijkstra algorithm. 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 won’t be guaranteed to find the best result. The Dijkstra algorithm will find the best result, but its speed, in many cases, is too slow [TLTK].

### A\* parsing algorithm [TLTL] which was proposed by Dan Klein and Christopher D.Manning can overcome the both problems: best result and speed. A\* parsing which is belong 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 each step of parsing process:

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 the most important figure is h(x), it determines how fast the parsing process leads to the target.

There are many algorithms which have been researched and developed in Vietnamese parsing like Beam Search, Greedy algorithms and Machine learning... In our knowledge, there is no research about A\* algorithm. So, A\* algorithm for parsing is a good choice to research.

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

Some abbreviations:

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

POS tags (part of speech), is a tag lexical which appears in G

# A\* ALGORITHM FOR PARSING

### A\* parsing algorithm [TLTL] which was proposed by Dan Klein and Christopher D.Manning can overcome the both problems: best result and speed. A\* parsing which is belong 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 each step of parsing process:

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 the most important figure is h(x), it determines how fast the parsing process leads to the target.

## 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). [TLTK]

The initial CHART is empty.

The input string is tokenized into *n* words *a1…an*. And then, these words are POS tagged to create an initial AGENDA: *{(Xi [i, i+1], wi ),}*, wi is an initial weight of each tagged word Xi.

A context for *node X[i,j]* (with input string *a1…an*) is a parse tree whose leaf nodes are labeled as *a1…ai-1X**aj+1*...an. 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]*.

## A\* parsing process

While AGENDA is not empty and CHART does not contain S [1, n+1] (an assignment to S):

*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 (1)*

* For each *node* (Z[j,k],w’) in CHART where G contains a production , 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 , 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).

# pruning hierarchical GRAPH method (PHGM)

## The context for proposition

In the A\* parsing process (presented in section II.B), there are two cases that would be happened when combining the candidate *node Y [i,j]* with CHART:

*The relevant production is a Chomsky-form,* it means that it has *less-than or equal to* two elements (POS tags) on the extension part. In this case, the combination follows step (1) in A\* algorithm.

*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 a *wait* parameter which denotes the *POS tags* which are remained to complete the rule. For example, when A[1,2] *node* and B[2,5] *node* are combined together using a production like “E → A B C D”, they will form a virtual *node* E[1,5,wait = “C D”]. Later, if the virtual node E[1,5,wait=”C D”] meets C[5,7] *node*, this combination will form the *node* E[1,7,wait=“D”].

When 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 of the grammar rules the combination using VNM will generate a large quantity of redundancy *nodes.* For instance, in the table 1, the combination of two instance elements N[2,7] and V[7,8] has formed so many new elements, and only few of them are significant.

1. all the *nodes* was formed when combined N(2,7) and V(7,8)

|  |  |
| --- | --- |
| *NP[2,8, wait=""]* | *NP[2,8, wait=", A"]* |
| *NP[2,8, wait="AP"]* | *NP[2,8, wait="AP NP"]* |
| *NP[2,8, wait="AP PP"]* | *NP[2,8, wait="MP"]* |
| *NP[2,8, wait="N"]* | *NP[2,8, wait="NP"]* |
| *NP[2,8, wait="NP PP"]* | *NP[2,8, wait="NP VP"]* |
| *NP[2,8, wait="P"]* | *NP[2,8, wait="PP"]* |
| *NP[2,8, wait="PP PP"]* | *NP[2,8, wait="VP"]* |

## The proposed model

### The basic idea

Instead of using VNM, our pruning hierarchical graph method (PHGM) uses *combinable chain* to overcome the *Chomsky production*. A *combinable chain* is a *node* sequencewhich has the continuous position*.* For example, a simple combinable chain: (NP[1,3] PP[3,5] VP[5,8]). In PHGM model, all the combinable chains of a candidate *node* and CHART will be used in step (1) of A\* algorithm. Unlike virtual node method, PHGM model does not form the redundancy virtual *nodes* and it decreases the number of *node* in parsing process.

For instance, the candidate *node* has the start-end position as X[7,10] and the content of CHART has been shown in Table 2.

1. the *nodes in* CHART

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X1[1,8] | X2[6,16] | X3[13,35] | X4[5,20] | **X5[2,7]** | **X6[10,11]** |
| X7[8,27] | X8[2,21] | X9[9,11] | X10[2,13] | X11[6,14] | X12[15,26] |
| X13[14,23] | X14[5,18] | **X15[1,7]** | X16[9,16] | X17[12,17] | X18[7,18] |
| X19[6,25] | **X20[13,26]** | **X21[11,26]** | X22[9,24] | **X23[11,20]** | X24[8,18] |
| X25[7,16] | X26[14,16] | X27[4,6] | **X28[13,21]** | X29[4,8] | **X30[11,13]** |

From this input data (candidate and CHART *nodes* position), the PHGM will process and use the combinable chains which are presented in Table 3:

1. the combinable chains of the candidate with CHART *nodes.*

|  |  |  |
| --- | --- | --- |
| 1 | Position | *node* |
| 2 | [2-7] **[7-10]** | X5**X** |
| 3 | [1-7] **[7-10]** | X15**X** |
| 4 | [1-7] **[7-10]** [10-11] [11-13] | X15 **X** X6 X30 |
| 5 | [2-7]**[7-10]**[10-11] [11-13] [13-26] | X5 **X** X6 X30 X20 |
| 6 | **[7-10]** [10-11] [11-20] | **X** X6 X23 |
| 7 | **[7-10]** [10-11] [11-13] | **X** X6 X30 |
| 8 | **[7-10]** [10-11] [11-13] [13-26] | **X** X6 X30 X20 |
| 9 | **[7-10]** [10-11] [11-13] [13-21] | **X** X6 X30 X28 |
| 10 | [2-7] **[7-10]** [10-11] | X5 **X** X6 |
| 11 | [2-7] **[7-10]** [10-11] [11-16] | X5 **X** X6 X21 |
| 12 | [2-7] **[7-10]** [10-11] [11-20] | X5 **X** X6 X23 |
| 13 | [2-7] **[7-10]** [10-11] [11-13] | X5 **X** X6 X30 |
| 14 | **[7-10]** [10-11] [11-16] | **X** X6 X21 |
| 15 | [1-7] **[7-10]** [10-11] | X15 **X** X6 |
| 16 | [2-7] **[7-10]** [10-11] [11-13] [13-21] | X5 **X** X6 X30 X28 |
| 17 | [1-7] **[7-10]** [10-11] [11-16] | X15 **X** X6 X21 |
| 18 | [1-7] **[7-10]** [10-11] [11-20] | X15 **X** X6 X23 |
| 19 | **[7-10]** [10-11] | **X** X6 |
| 20 | [1-7] **[7-10]** [10-11] [11-13] [13-26] | X15 **X** X6 X30 X20 |
| 21 | [1-7] **[7-10]** [10-11] [11-13] [13-21] | X15 **X** X6 X30 X28 |

Thus, assuming that there is a production (A → X5 **X** X6 X30 X28)relevant to the 16th chain in the table 3, the *node* A(2,21) will be formed and will be added to AGENDA.

### PHGM combinable-chains generator model

PHGM combinable-chains generator model includes two phases: *classification phase* and *combinable chains generation phase*.

*Classification phase* (CP)(1): the parser classifies the *nodes* in CHART into the difference blocks.

*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.

#### Classification phase

The PHGM classification phase is based on *pigeon hole sort* algorithm idea. There are holes which are created for adding pigeon. But the holes in PHGM are used for CGP(2) instead of sorting.

The holes in the PHGM are divided into two types: the *left hole*s and the *right hole*s (Figure 1). Let assuming that X is a candidate *node*.

We have two kind of *node* in CHART:

* *Left node of X*: This is a set of *nodes* that have their *end* position <= *start* position of X. All the *nodes* which have the same *end* position will be grouped in a block labeled as *end* position of them*.* And a set of all these blocks is called as *left holes*.
* *Right node of X*: This is a set of *nodes* that have their *start* position >= *end* position of X. All the *nodes* that have the same *start* position will be grouped in a block labeled as *start* position of them*.* And a set of these blocks is called as *right holes*.

#### Combinable chains generation phase

With the input as the classified CHART, the parsing system begins generating the combinable chains. This phase includes three main parts: “generating *left chain*s”, “generating *right chain*s” and “*generating combinable chains*”.

***1. Generating left chains****:* this module generates all the combinable chains which end with candidate X, it is called as the *left chain*s. We imply that S(E) is the block in a ***left hole*** which is labeled as a *start* position of node E. This part can be described as below:

* Parsing system processes the X *node*, save the left combinable chain corresponding to X and get the S(X) from *left holes*.
* This progress is done recursively for all the *nodes* in the S(X).

Figure 1 – The instance example for generating *left chain*s.

***2. Generating right chains****:* the same progress as the “generating *left chain*s” is realized.

Figure 2 – The instance example for generating *right chain*s.

We imply that E(S) is the block in a ***right hole*** which is labeled as a start position of node S. This part can be described as:

* Parsing system processes the X *node*, save the right combinable chain corresponding to X and get the E(X) from *right holes*.
* This progress is done recursively for all the *nodes* in the E(X).

***3. Generating combinable chain****:* from two first phases we got the *left chain* and *right chain* of the candidate. The connection of three factors “*left chain*”, “*right chain*” and X will form the real combinable chains of X.

## Pruning graph in PHGM model

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

From our experiment on testing performance of PHGM model, we found that there are approximately 8% of the combinable chains that could be used. Because of this, PHGM 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, PHGM 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.

### The idea

A *node* has two information: position and POS. PHGM basic 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 PHGM model, they are just the same, even their POS is difference. So, a pruning graph in PHGM model will show you how to use the POS to reduce the processing time of PHGM.

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

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

### Statistic training phase

As described, the training data of the PHGM pruning graph is G.

Specifically, with each “T” POS in G, the system creates two graphs:

#### 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 creating algorithm of left-POS map:

* Process all the productions in the graph.
* For each production whose extension part likes [Pn … P2 P1 T…]:
  + - If P1 is not child of T, then add P1 into T children.
    - For i from 2 to n: if Pi is not child of Pi-1, then add Pi into Pi-1 children.

#### 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 P1 P2 … Pm]:
  + - If P1 is not child of T, then add P1 into T children.
    - For i from 1 to m: if Pi is not child of Pi-1, then add Pi into Pi-1 children.

### Pruning phase

As described, the PHGM 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 part: “Generating left chains”, “Generating right chains” and the collection part of those two “Generating combinable chains”.

The input is still the classified CHART; The PHGM process will perform normally with the support of pruning graph. The system get pruning graph for POS of candidate *node* and use it to prune the bad combinable chains (cannot lead to any production) at two phase “generating left chains” and “generating right chains”. 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 3 below is an illustration of pruning graph in the PGHM model.

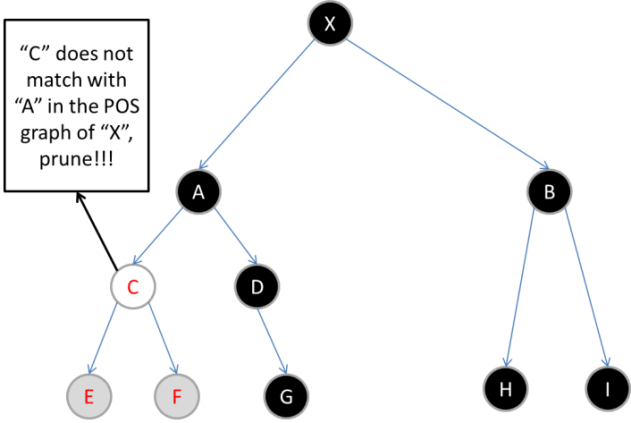


Figure 3 – the illustration of the PHGM model using pruning graph.

# experiment and result

This section presents the preparation and the result of experiment to illustrate the performance of A\* parsing algorithm using PHGM model.

We have made two tests. In the first one, we test and evaluate the speed of PHGM model by comparing it to the virtual node method (VNM). And the second one, we compare A\* PHGM algorithm with one other knowledge algorithm which have been developed for Vietnamese Parsing Technique: Beam search algorithm.

## Preparation for experiment

The training corpus of our parsing system is Viet Treebank database from VLSP project. It includes about 10.000 Vietnamese sentences parsed by hand. From this corpus, we extract the grammar rules G which have approximately 9900 productions. Each production has the weight parameter which is calculated by appearing probabilistic of production in G.

Corresponding to this experiment, we have used 200 sentences picked from Viet Treebank to test the performance of our proposed method.

## Conduct the experiment

The purpose of this experiment is to test the speed performance of PHGM model. We make a testing race between three candidates: PHGM model, virtual node method (VNM), PHGM model using pruning graph (PHGM-PG). The testing corpus which is used for this experiment is 500 sentences from VLSP corpus as described above. The testing set is grouped by number of its word. We have 6 groups: 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.

## Results and evaluate

The result of the first experiment is shown in figure 4. X-axis is number of tokens in the input string; y-axis is an average processing time measured in second. With the sentences has less than 20 words (tokens), PHGM, VNM and PHGM-PG got the same speed. When the number of tokens up to 50, the PHGM 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.

Figure 4 – result of the speed experiment between A\* PHGM, A\* VNM and A\* PHGM using pruning graph

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

* A\* “admissible heuristic”: The most important thing in A\* parsing algorithm is a heuristic of distance to the goal. Our heuristic is a classical one which was indicated in [3]. But this heuristic has not been researched and evaluated carefully in our Vietnamese parsing technique. In addition, there are so many better heuristics which have been proposed recently. So, we will continue our research on this domain and describe it on our future science paper.
* The grammar rules G: our grammar rules have been extracted from VLSP corpus.

# conclusion and future works

This paper have described our PHGM method based on A\* algorithm to improve the speed of Vietnamese parsing system. Through the experiment, the speed performance of PHGM is relatively good and acceptable. The Vietnamese parsing system has been implemented in JAVA to evaluate our method. This method could be used for other languages, too.

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 could also research the way to integrate lexical information into our algorithm. This will significantly increase the speed and the accuracy of parsing system.

# references

1. **Fei Xia**, “*Inside-Outside algorithm*”, LING 572.
2. **Website:** *http://VLSP.vietlp.org:8080/demo/?page=resources*
3. **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).
4. **Dan Klein and Christopher D. Manning**. 2002. *“A\* parsing: Fast exact Viterbi parse selection”*. Technical Report dbpubs/2002-16, Stanford University, Stanford, CA.
5. **Hoang Anh Viet**, “*Vietnamese parsing technique using PCFG*”, graduation thesis, HUST, 2006.
6. **Pham Thi Nhung**, “*Vietnamese parsing technique using beam-search algorithm*”, graduation thesis, HUST, 2009.
7. **Lam Do B., Huong Le T**. 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.
8. **Website,** [*http://en.wikipedia.org/wiki/A\*\_search\_algorithm*](http://en.wikipedia.org/wiki/A*_search_algorithm)*.*
9. **TRAN D.D.**, **CASTELLI E., TRINH V. L. & LE V.B.**. (2004) *Building a large Vietnamese Speech Database.* Revue vietnamienne "Science et Technologie" (ISBN 0868-3980). Vol. 46/47, February 2004, pp 13-17