A Sequential Truncation Parsing Algorithm based on the Score Function

Keh-Yih Su[*], Jong-Nae Wang[**], Mei-Hui Su[**] and Jing-Shin Chang[***]

[*]Department of Electrical Engineering
National Tsing Hua University, Hsinchu, Taiwan, R.O.C.

[**]BTC R&D Center
28 R&D Road II, 2F
Science-Based Industrial Park, Hsinchu, Taiwan, R.O.C.

[***]Institute of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

Key Words: Machine Translation, Parsing Strategy, Score Function, Sequential Truncation Parsing Algorithm

ABSTRACT

In a natural language processing system, a large amount of ambiguity and a large branching factor are hindering factors in obtaining the desired analysis for a given sentence in a short time. In this paper, we are proposing a sequential truncation parsing algorithm to reduce the searching space and thus lowering the parsing time. The algorithm is based on a score function which takes the advantages of probabilistic characteristics of syntactic information in the sentences. A preliminary test on this algorithm was conducted with a special version of our machine translation system, the ARCHTRAN, and an encouraging result was observed.

Motivation

In a natural language processing system, the number of possible analyses associated with a given sentence is usually large due to the ambiguous nature of natural languages. But, it is desirable that only the best one or two analyses are translated and passed to the post-editor in order to reduce the load of the post-editor. Therefore, in a practical machine translation system, it is important to obtain the best (in probabilistic sense) syntax tree having the best semantic interpretation within a reasonably short time. This is only possible with an intelligent parsing algorithm that can truncate undesirable analyses as early as possible.

There are several methods to accelerate the parsing process [Su 88b], one of which is to decrease the size of the searching space. This can be accomplished with a scored parsing algorithm that truncates unlikely paths as early as possible [Su 87a, 87b] and hence decreases the parsing time.

As for the searching strategy for the scored parsing algorithm, it may be either parallel or sequential. But in our system, a time limit is used to stop the parsing process when a sentence is taking too long to parse because its length or because it has a very complicated structure. Therefore, the sequential searching strategy is better for us than the parallel approach because we are likely to have some complete syntax trees to work with even if the parsing was suspended abnormally when its time expires. On the other hand, the parallel approach will not have this advantage because none of the on-going paths have traversed to the end.

In this paper, we are proposing a sequential truncation algorithm for parsing sentences efficiently. This algorithm employs the score function we proposed in [Su 88a]. However, this algorithm is different from the one proposed in [Su 87a, 87b], which described a parallel truncation algorithm for scored parsing. Here, we are adopting a sequential truncation method. While we are using this sequential approach, a large speed-up in the parsing time has been observed.

Definition of the Score Function

In a scored parsing system, the best analysis is selected base on its score. Several scoring mechanisms have been proposed in the literatures [Robi 83, Benn 85, Gars 87, Su 88a]. The one we adopt is the score function based on the conditional probability we proposed in [Su 88a]. How to select the best analysis of a sentence is now converted into the problem of finding the semantic interpretation (*Sem*$_i$), the syntactic structure (*Syn*$_j$) and the lexical categories (*Lex*$_k$) that maximize the conditional probability of the following equation,

$$\begin{aligned} &SCORE(Sem_i, Syn_j, Lex_k) \\ &\quad = P(Sem_i, Syn_j, Lex_k | w_1 \ldots w_n) \\ &\quad = P(Sem_i | Syn_j, Lex_k, w_1 \ldots w_n) * P(Syn_j | Lex_k, w_1 \ldots w_n) * P(Lex_k | w_1 \ldots w_n) \\ &\quad = SCORE_{sem}(Sem_i) * SCORE_{syn}(Syn_j) * SCORE_{lex}(Lex_k) \quad , \end{aligned} \tag{1}$$

where w_1 to w_n stands for the words in the given sentence and the last three product terms are semantic score, syntactic score and lexical score respectively. Since we are using just the syntactic information in our current implementation, we will focus only on the syntactic aspect of this score function (i.e. *SCORE*$_{syn}$*(Syn*$_j$*)*, which can be approximated by $SCORE_{syn}(Syn_j) \approx P(Syn_j | Lex_k) = P(Syn_j | v_1 \ldots v_n)$, where v_1 to v_n are the lexical categories corresponding to w_1 to w_n).

To show the mechanism informally, first refer to the syntax tree in Fig. 1. shown here with its reduction sequences (produced with a bottom-up parsing algorithm), where L_i is i-th phrase level consists of terminals and nonterminals. The transition from a phrase level L_i to the next phrase level L_{i+1} corresponds to a reduction or derivation of a nonterminal at time t_i.

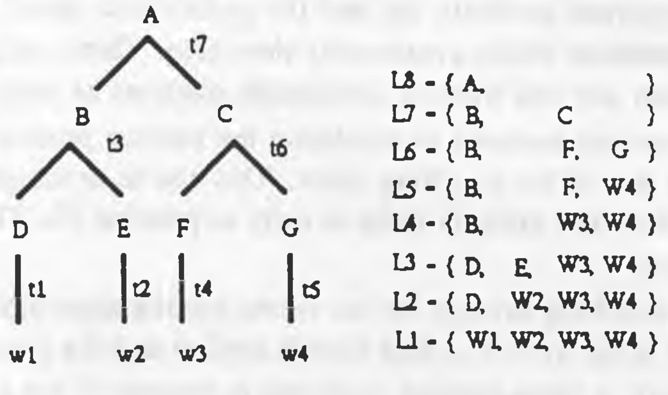

Fig. 1 Different Phrase Levels for a bottom-up Parsing

The syntax score of the tree in Fig. 1 can be formulated as the following conditional probability equation, where l_i and r_i are the left and right contexts of the reducing symbols:

$$\begin{aligned} &SCORE_{syn}(Syn_A) \\ &\quad = P(L_8, L_7 ... L_2 | L_1) \\ &\quad = P(L_8 | L_7 ... L_2, L_1) * P(L_7 | L_6 ... L_1) * ... * P(L_2 | L_1) \\ &\quad \approx P(\{A\} | \{l_7, B, C, r_7\}) * P(\{C\} | \{l_6, F, G, r_6\}) * ... * P(\{D\} | \{l_1, w_1, r_1\}) \end{aligned} \tag{2}$$

Eq. 2 can be further reduced to the following equation if only one left and one right context symbol are considered where "$\emptyset$" is the null symbol.

$$\begin{aligned} &SCORE_{syn}(Syn_A) \\ &\quad \approx P(\{A\} | \{\emptyset, B, C, \emptyset\}) * P(\{C\} | \{B, F, G, \emptyset\}) * ... * P(\{D\} | \{\emptyset, w_1, w_2\}) \end{aligned} \tag{3}$$

If we want to calculate the score at the point where a word is just being fetched (compact multiple reductions and one shift into one step), the $SCORE_{syn}(Syn_A)$ can also be approximated into the following equation.

$$\begin{aligned} &SCORE_{syn}(Syn_A) \\ &\quad = P(L_8, L_7 ... L_2 | L_1) \\ &\quad = P(L_8, L_7, L_6 | L_5, L_4 ... L_1) * P(L_5 | L_4, L_3 ... L_1) * P(L_4, L_3 | L_2, L_1) * P(L_2 | L_1) \\ &\quad \approx P(L_8, L_7, L_6 | L_5) * P(L_5 | L_4) * P(L_4, L_3 | L_2) * P(L_2 | L_1) \\ &\quad \approx P(L_8 | L_5) * P(L_5 | L_4) * P(L_4 | L_2) * P(L_2 | L_1) \end{aligned} \tag{4}$$

Two assumptions were made in formulating Eq. 2–4. First, it is assumed that the forming of phrase level i is only dependent on its immediate lower phrase level, since most information percolated from other lower levels is contained in that level. And second, a reduction is only locally context sensitive to its left or right context at each phrase level. This assumption is also supported in other systems as well [Marc 80, Gars 87].

A simulation based solely on this syntactic score was conducted and reported in [Su 88a] with a full-path searching algorithm. The result shows that the correct syntactic structures of over 85% of the test sentences were successfully picked when a total of three local left and right context symbols were consulted.

The Sequential Truncation Algorithm

Using the score function defined in the previous section, we will present the idea of sequential truncation algorithm with Fig. 2.

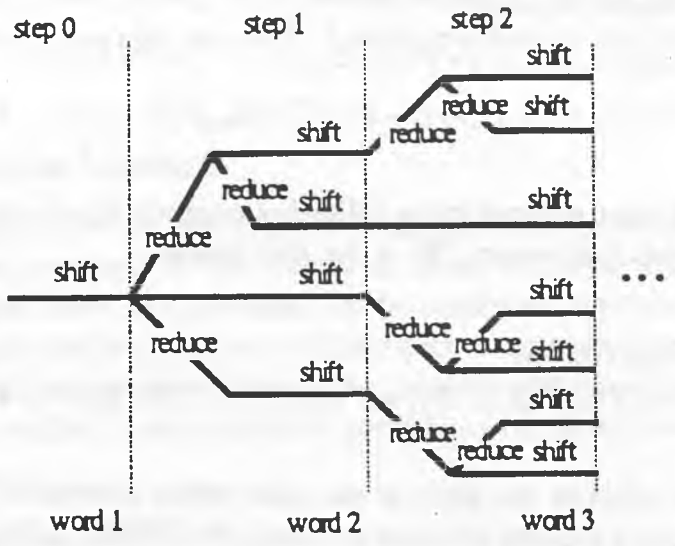

Fig. 2 The searching tree

Each path in Fig. 2 corresponds to a possible derivation of a given sentence. The parser will use the depth-first strategy to traverse the searching tree. But during the searching process, the parser compares the score of each path accumulated so far with a running threshold $C(\alpha_i)$ (a detailed definition will be given in the following section) at each step i when the next word is fetched. If the score of the path is less than the running threshold $C(\alpha_i)$, it will be truncated, i.e. blocked, and the next path will be tried. This process continues until we get the first complete parse tree (i.e. when the whole sentence is reduced to a S node). After we obtain the first complete parse tree, a lower bound for the scores is acquired. The parser will continue to traverse other pathes, but from now on, the score of each path will also be compared with the final accumulated score of the first complete parse tree in addition to be compared with the running threshold. This additional comparison is similar to the branch and bound strategy employed in many AI applications [Wins 84] and it will accelerate the parsing process further. The whole process is shown in the flow chart in Fig. 3. If the test fails in either case, this path will be truncated. Continuing in this manner, we may get a second complete parse tree which has a final score higher than the first one. In this case, we will replace the lower bound with the final score of the second parse tree and repeat the whole process until the end of the entire searching process.

If all the paths are blocked without arriving at any complete parse tree, we can adopt one of two possible strategies. First, we could loosen the running thresholds, i.e. lowering the $C(\alpha_i)$, and try the deepest path gone so far again. Second, we can process this sentence in fail-soft mode. The fail-soft mechanism will skip and discard the current state and attempts to continue the parsing at some later point.

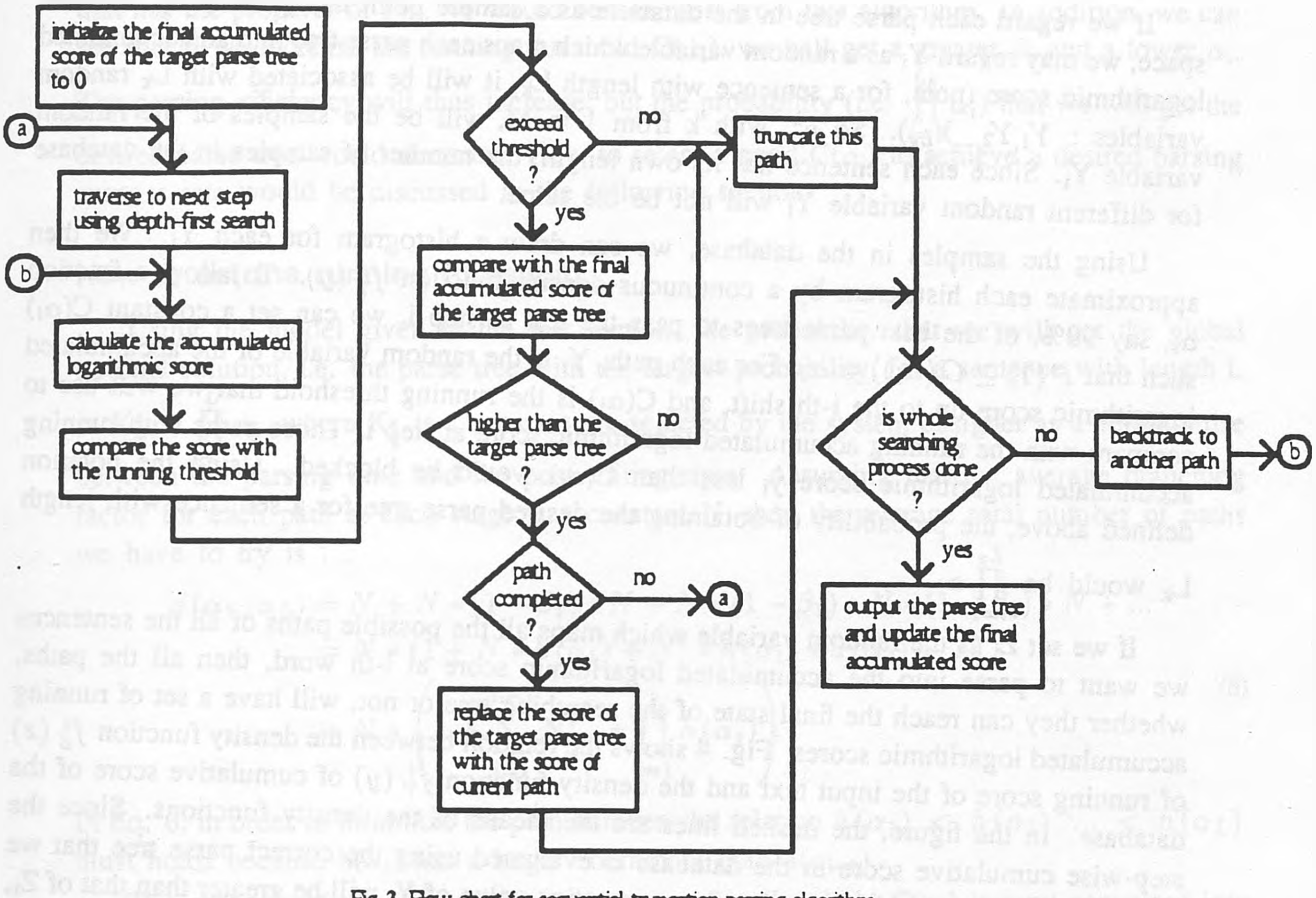
initialize the final accumulated score of the target parse tree to 0
a
traverse to next step using depth-first search
b
calculate the accumulated logarithmic score
compare the score with the running threshold
exceed threshold ?
no
yes
truncate this path
compare with the final accumulated score of the target parse tree
higher than the target parse tree
no
yes
path completed ?
no
a
yes
replace the score of the target parse tree with the score of current path
is whole searching process done ?
no
backtrack to another path
b
yes
output the parse tree and update the final accumulated score

Fig. 3 Flow chart for sequential truncation parsing algorithm

The effectiveness of the sequential truncation algorithm depends on the distribution of scores of the database and the input sentences. As we can see, for each syntax tree can be expressed as the product of a sequence of conditional probability as shown in Eq. 4. Each term in the product corresponds to a transition between two "shift" actions and is evaluated immediately after a "shift". Taking the logarithm on both sides of Eq. 4, we get the following equation where X_i denotes a sequence of phrase levels at i-th step and L is the length of the sentence.

$$log\,(SCORE_{syn}\,(Syn)) = \sum_{i=1}^{L} log\ P(X_i|X_{i-1}) \qquad (5)$$

If we define $y_j = \sum_{i=1}^{j} log\ \ P(X_i|X_{i-1})$, then y_j denotes the accumulated logarithmic score up to the j-th word which is also the j-th shift of the sentence.

Suppose we have M sentences with their correct parse trees in the database. For each parse tree, we can evaluate y_j by using the logarithmic score function defined before. So for the k-th sentence in the database, we obtain a sequence $y_1^k,\ y_2^k,\ \cdots y_{L_k}^k$, where y_j^k denotes the accumulated logarithmic score of the k-th sentence and L_k denotes the length of the k-th sentence.

If we regard each parse tree in the database as a sample point in a probability outcome space, we may regard Y_i as a random variable which maps each parse tree into an accumulated logarithmic score (note, for a sentence with length L_k, it will be associated with L_k random variables : $Y_1, Y_2, \ldots Y_{L_k}$). So y_i^k, with k from 1 to M, will be the samples of the random variable Y_i. Since each sentence has its own length, the number of samples in the database for different random variable Y_i will not be the same.

Using the samples in the database, we can draw a histogram for each Y_i. We then approximate each histogram by a continuous density function $f_Y^i(y)$. To allow a fraction α_i, say 99%, of the best parse trees to pass the test at step i, we can set a constant $C(\alpha_i)$ such that $P(Y_i \geq C(\alpha_i)) = \alpha_i$. For each path, Y_i is the random variable of the accumulated logarithmic score up to the i-th shift, and $C(\alpha_i)$ is the running threshold that we will use to compare with the running accumulated logarithmic score at step i. Those paths with running accumulated logarithmic score y_i less than $C(\alpha_i)$ would be blocked. Using the notation defined above, the probability of obtaining the desired parse tree for a sentence with length L_k would be $\prod_{i=1}^{L_k} \alpha_i$.

If we set Z_i as the random variable which maps all the possible paths of all the sentences we want to parse into the accumulated logarithmic score at i-th word, then all the paths, whether they can reach the final state of the searching tree or not, will have a set of running accumulated logarithmic scores. Fig. 4 shows the relation between the density function $f_Z^i(z)$ of running score of the input text and the density function $f_Y^i(y)$ of cumulative score of the database. In the figure, the dashed lines are the means of the density functions. Since the step-wise cumulative score in the database is evaluated using the correct parse tree that we have selected, we would expect that the expectation value of Y_i will be greater than that of Z_i, that is, $E[Y_i] > E[Z_i]$; and the variance of Y_i is less than that of Z_i, that is $Var[Y_i]<Var[Z_i]$.

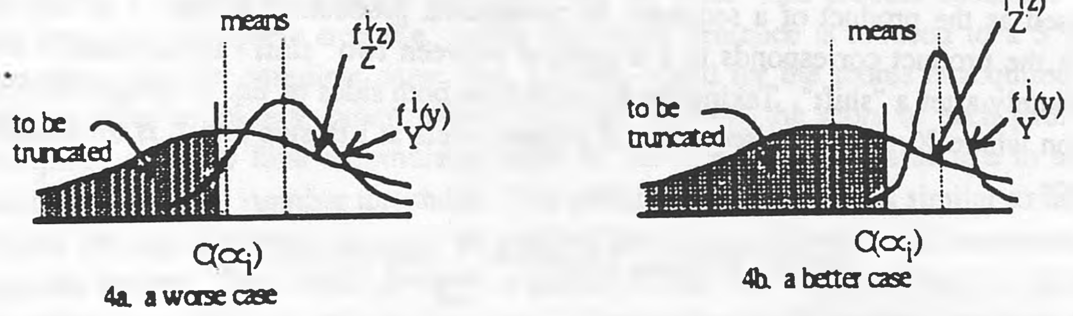

Fig. 4 Relationship between the running score of the input text and the cummulative score of the database

Let β_i denotes $F_Z^i(C(\alpha_i))$, where $F_Z^i(z)$ is the cumulated distribution function of Z_i, then β_i is the probability that a path will be truncated at the i-th step of the searching tree. By using this sequential truncation method, the searching space would then be approximately reduced to $\prod_{i=1}^{L_k}(1-\beta_i)$, which is a small portion of the original searching space generated by a full path searching algorithm. Therefore the efficiency of parsing is increased. Since β_i in Fig. 4a is less than that in Fig. 4b, which correspond to the situation that has a large expectation difference (E[Y$_i$]-E[Z$_i$]) and a small variance ratio (Var[Y$_i$]/Var[Z$_i$]), the underlying grammar that has the property of Fig. 4b would benefit most from this algorithm. In addition, we can see that if we increase the running threshold C(α_i), we will get a greater β_i and a lower α_i. The parsing efficiency will thus increase, but the probability (i.e. $\prod_{i=1}^{L_k} \alpha_i$) that we will get the desired parse tree would decrease. How to select a good C(α_i) to achieve a desired parsing success rate would be discussed in the following section.

How to set the running threshold

Using the model given in the last section, the probability that we will get the global optimal solution, i.e. the parse tree with the largest probability, for a sentence with length L is $K_L = \prod_{i=1}^{L} \alpha_i$, where K$_L$ is a constant pre-selected by the system designer as a compromise between the parsing time and the post-editing time. Assuming that the average branching factor for each path at each stage is a constant N, then the average total number of paths we have to try is :

$$\begin{aligned} g(\alpha_1 ... \alpha_L) &= N + N * (1-\beta_1) * N + N * (1-\beta_1) * N * (1-\beta_2) * N + ... \\ &= N * (1 + N * h(\alpha_1) + N^2 * h(\alpha_1) * h(\alpha_2) + ...) \\ &= N * \left(1 + \sum_{i=1}^{L-1} N^i \quad * \prod_{j=1}^{i} h(\alpha_j) \right) \end{aligned} \tag{6}$$

In Eq. 6, in order to minimize the path number, the relation $h(\alpha_1) < h(\alpha_2) \; ... < h(\alpha_L)$ must holds because h(α_i) has a larger coefficient than h(α_{i+1}).

The problem of selecting an appropriate running threshold C(α_i) is now converted into one of minimizing g(α_1...α_L) under the constraint of $\prod_{i=1}^{L} \alpha_i = K_L$. Taking the logarithm on both sides, we get $\sum_{i=1}^{L} log\ \alpha_i = log\ K_L$. Then the Lagrange multiplier λ is used to get $g^*(\alpha_1 ... \alpha_L) = g(\alpha_1 ... \alpha_L) + \lambda * \sum_{i=1}^{L} log\ \alpha_i$. Taking the partial derivative of g* with respects to α_1...α_L, we will get the following equations :

$$\frac{\partial g^*}{\partial \alpha_1} = 0, \quad \frac{\partial g^*}{\partial \alpha_2} = 0, \quad ... \quad \frac{\partial g^*}{\partial \alpha_L} = 0, \quad and \quad \sum_{i=1}^{L} log\ \alpha_i = log\ K_L \tag{7}$$

There are (L+1) variables, which are α_1...α_L, and λ, and (L+1) equations. So, α_1...α_L can be solved by the numerical method. Since α_i is usually very close to 1, we can linearize the function h(α_i) in the region around α_i=1 and approximate by $h(\alpha_i) \approx a * \alpha_i + b$. In this way, we can substitute h(α_i) in the above equation by $a * \alpha_i + b$ to simplify the calculation.

During our derivation, we have assumed that the average branching factor at each stage is a constant N. This constraint can be relaxed by assuming the average branching factor at i-th stage to be N$_i$. In this way, we will get a more complicated expression for g(α_1...α_L), but it can still be solved in the same way.

The running threshold C(α_i) can now be computed off-line by selecting different K$_L$ for different sentence length L. We will call this set of C(α_i) the "static running threshold", because once they are computed, they will not be changed during the sentence parsing. However, if we arrive at a complete parse tree with much higher final accumulated running score than the final accumulated running threshold, then even if a path can pass all the accumulated running thresholds it might still be discarded when it is being compared with the final accumulated running score. So, the running threshold should be adjusted to reflect a high final accumulated running score. Therefore, it would be better if the running threshold is changed to $C'(\alpha_i)=C(\alpha_i)+\Delta C(\alpha_i)$, where $\Delta C(\alpha_i)$ is set to $\gamma * (y_i^* - C(\alpha_i))$, where $0<\gamma<1$ and y_i^* is the accumulated logarithmic score of the current best parse tree at the i-th step, and γ is a tunning constant pre-selected by the system designer. $C'(\alpha_i)$ is then the "dynamic running threshold". Using the dynamic running threshold, the efficiency of parsing would be further improved.

If it so happen that all the pathes are blocked before any complete parse tree is formed, we can find the deepest path (let us assuming it to be at the j-th step) among the blocked ones and continue it with a lowered running threshold of $C'(\alpha_j)=y'_j$, where y'_j is the score of this path at the j-th step. Since the procedure to lower the running threshold is quite complicated and uses up memory space in run time, it might be better just invoke the fail-soft mechanism for sentences whose paths are all blocked.

Testing

We completed two preliminary testings of truncation algorithm with special versions of our English-Chinese MT system and a database of 1430 sentences.

In the first experiment, the sentence parsing time needed by a charted parser that uses bottom-up parsing with top-down filtering is compared with the time needed by the same charted parser with truncation mechanism. From the test, we found that the average sentence parsing time by the charted parser with truncation is improved by a factor of four. For some sentences, the improvement can go as high as a factor of twenty. This result is encouraging because minimizing parsing time is critical to a practical MT system.

Nevertheless, we noted that our output quality has degraded slightly. By this, we mean that the best selected tree produced by the charted parser with no truncation is not among the trees produced by the charted parser with truncation. Exploring this problem further, we discovered that the chart [Wino 83] used during parsing is in conflict with the truncation mechanism. The reason for having chart is to be able to store all subtrees that were parsed in previous path traversal. So, when we backtrack to the next path and arrive at the same range of inputs, the same subtrees can be used again without reparsing. However, the idea behind the truncation mechanism is to discard subtree in the context in which it has low probability. Therefore, if we adopt the truncation mechanism during parsing, not every subtree between a string of inputs is successfully constructed and stored into the chart. For example, in Fig. 5, there are two possible subtrees between b and c when the pathes in the block A are expanded.

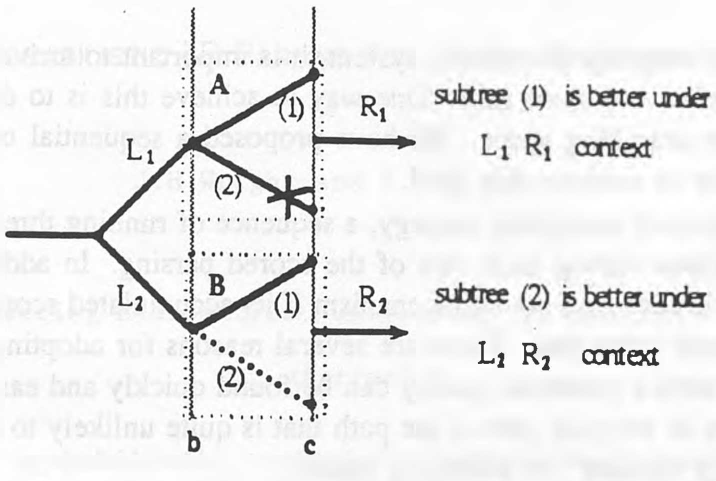

Fig. 5. Chart with truncation mechanism

In Fig. 5, one of the subtrees is discarded and the other is stored into the chart. There are two reasons why a subtree may be discarded. First, it might be caused by a natural language's constraints on the context dependency. Second, a subtree might be discarded because of its small running accumulated score (and thus truncated by the truncation mechanism.) Either will leave us a chart with incomplete subchart. So, this will result in the best possible tree being missing as a side-effect of using this chart. For instance, in Fig. 5, the best tree might be the second subtree with the left context of L_2 and with the right context of R_2 (i.e., its probability is the highest.) But, since the path expansion starting from the left context of L_1 has the second subtree discarded because its probability under the context of L_1 and R_1 is small, the best tree will never be formed. Therefore, with a chart having incomplete subcharts, the possibility of obtaining the best tree is determined by the pathes traversed before.

One solution to this incompatibility problem is to mark the sections of the chart that are complete. Hence, if an incomplete subchart is encountered again, it will be reparsed. On the other hand, if a complete set of chart is encountered, the subtrees can be copied directly from the chart. Another solution is to suspend the truncation mechanism when a set is being tried the first time. And if subtrees are copied directly from the chart, the truncation mechanism resumes its normal function. In this way, it is guaranteed that every subchart in the chart is complete. Both of these solutions increase our sentence parsing time as the overhead. This compromise, however, is unavoidable if the advantages of using chart are to be maintained.

In the second experiment, we converted the charted parser for the first experiment into one with sequential searching strategy and without the use of the chart. Similar sentence parsing test is conducted for this chartless parser but with a smaller analyses grammar. The result shows that the total parsing time for this parser with truncation mechanism added is better than the same parser without truncation by the factor of three.

From the positive results of the above two experiments, we have shown the inclusion of the sequential truncation algorithm is advantageous for a MT system. In addition, we have also shown the feasibility of harmonize the use of chart and the truncation algorithm. Currently, we are in the process of resolving the incompatibility problem between the chart and the truncation mechanism and constructing a working system with this solution.

Conclusion

In a natural language processing system, it is important to arrive at a good analysis for a sentence in a relatively short time. One way to achieve this is to decrease the parsing time by reducing the searching space. We have proposed a sequential truncation algorithm with a score function to achieve this goal.

In this sequential truncation strategy, a sequence of running thresholds are used to bound the searching space during each step of the scored parsing. In addition, a path can also be blocked by the branch-and-bound mechanism if its accumulated score is lower than that of an already completed parse tree. There are several reasons for adopting this strategy. First, the first parse tree with a moderate quality can be found quickly and easily. Second, the running threshold serves to truncate part of the path that is quite unlikely to lead to the best analysis, and thus greatly reduces the searching space.

We have made a pilot test on the truncation mechanism with a charted parser that adopts bottom-up parsing with top-down filtering. With a database of 1430 sentences, the result indicates an average improvement in the sentence parsing time by the factor of four (for some sentences the improvement goes as high as a factor of twenty). However, we also discovered an incompatibility problem between the use of chart and the truncation mechanism. In another pilot test we conducted on the truncation mechanism, the sentence parsing time is tested for a chartless parser that adopts sequential parsing strategy. The result shows an improvement in parsing time by a factor of three for the inclusion of the truncation mechanism. These encouraging results demonstrate a great promise for the sequential truncation strategy.

As our current research topic, we shall resolve the incompatibility problem between the chart and the truncation algorithm and include the solution into our working MT system, the ARCHTRAN.

References

[Benn 85] Bennett, W.S. and J. Slocum, "The LRC Machine Translation System," *Computational Linguistics*, vol. 11, No. 2-3, pp. 111-119, ACL, Apr.-Sep. 1985.

[Gars 87] Garside, Roger, Geoffrey Leech and Geoffrey Sampson (eds.), *The Computational Analysis of English : A Corpus-Based Approach*, Longman , New York, 1987.

[Marc 80] Marcus, M.P., *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, MA, 1980.

[Robi 82] Robinson, J.J., "DIAGRAM : A Grammar for Dialogues," *CACM*, vol. 25, No. 1, pp. 27-47, ACM, Jan. 1982.

[Su 87a] Su, K.-Y., J.-S. Chang, and H.-H. Hsu, "A Powerful Language Processing System for English-Chinese Machine Translation," *Proc. of 1987 Int. Conf. on Chinese and Oriental Language Computing*, pp.260-264, Chicago, Ill, USA, 1987.

[Su 87b] Su, K.-Y., J.-N. Wang, W.-H. Li, and J.-S. Chang, "A New Parsing Strategy in Natural Language Processing Based on the Truncation Algorithm", *Proc. of Natl. Computer Symposium (NCS)*, pp. 580-586, Taipei, Taiwan. 1987.

[Su 88a] Su, K.-Y. and J.-S.Chang, "Semantic and Syntactic Aspects of Score Function," *Proc. COLING-88*, vol. 2, pp. 642-644, 12th Int. Conf. on Comput. Linguistics, Budapest, Hungary, 22-27 Aug. 1988.

[Su 88b] Su, K.-Y., "Principles and Techniques of Natural Language Parsing : A Tutorial," *Proc. of ROCLING-I*, pp.57–61, Nantou, Taiwan. Oct. 1988.

[Wino 83] Winograd, Terry, *Language as a Cognitive Process*, Addison-Wesley, Reading, MA., USA, 1983.

[Wins 84] Winston, P.H., *Artificial Intelligence*, Addison-Wesley, Reading, MA., USA, 1984.