Dynamic Programming: A Method for Taking Advantage of Technical Terminology in Japanese Documents

Eiko Yamamoto*, Mikio Yamamoto, Kyoji Umemura*, and Kenneth W. Church*****

* Toyohashi university of Technology, Department of Information and Computer Sciences, Japan,
Email: eiko@ss.ics.tut.ac.jp, umemura@tutics.tut.ac.jp,
** University of Tsukuba, Institution of Computer Sciences and Electronics, Japan,
Email: myama@is.tsukuba.ac.jp,
*** AT & T Labs —Research, U.S.A,
Email: kwc@research.att.com

Abstract

We introduce a new similarity measure based on dynamic programming, intended for technical terms such as *machine translation system*, which are quite common in technical writing. We compare our proposal with systems which use standard IDF cosine similarity, but on different vocabularies. The dynamic programming method is relatively strong when the query contains a single long technical term, and none of the words in the term are particularly good keywords.

Keywords: Terminology; Edit Distance; Information Retrieval; Technical term; Similarity Measure.

1 Introduction

What are the most effective units for Information Retrieval? The current consensus, at least for English, is leaning strongly in favor of words. Hardly anyone uses smaller units such as characters (or sequences of characters), though there are a few exceptions such as Damashek [4]. Character-based systems are more common for Asian languages [2,10,11,16]. Larger units such as phrases (or sequences of words) have received quite a bit of attention. Early results such as Fagan's [5] were quite promising. However, despite considerable effort over the past decade by many authors such as Strzalkowski and his colleagues [19], the community remains unconvinced. See Mitra *et al.* [14] for a discussion of phrases, suggesting that they don't help very much, especially at high ranks.

Mitra *et al.* found that phrases seem to be most helpful for the grey area (medium ranks), documents that are not obviously relevant (high ranks) and not obviously irrelevant (low ranks). In the experiments to be reported here we have also found that phrases are most helpful in the grey area, especially when both the query and the documents make heavy use of technical terminology, phrases like *Computer Science, Information Retrieval* and *machine translation.*

It is difficult to define the term: *technical terminology.* Unlike *general vocabulary*, the words that any speaker of the language would be expected to know, technical terminology is shared by a relatively small subgroup of domain experts. The distinction between general vocabulary and technical terminology is especially important to human translators, who find terminology quite challenging because they are rarely as familiar with the domain as either the authors of the source document or the readers of the target document. There is generally only a single correct translation of the term, and it can be dangerous to translate the parts of the term compositionally. We believe that the distinction between general vocabulary and technical terminology should also be important for Information Retrieval. Technical terms, when they are available, ought to be particularly strong clues for a relevancy. If you look at the index to almost any textbook in a technical area, such as page 975 of volume A of the *Handbook of Theoretical Computer Science* [20], one finds lots and lots of technical terms such as: *binary tree, bitonic sort, Boolean circuit, Boolean formula, breadth-first search, bubble sort, bucket sort, butterfly network, Chebychev's Inequality, Chinese Remainder Theorem, Chomsky hierarchy* and *Church's Thesis*. In fact, most of the terms, at least on page 975 of [20], are more than one word long. Multi-word terms are especially common in technical writing. The experiments to be reported here are based on a large corpus of technical abstracts in Japanese [7,8]; previous studies of phrases have worked with a variety of different (and often less technical) genres dynamo trap clamp bolt -> dynamo clamp bolt -> clamp bolt

gearbox end cover gasket -> end cover gasket -> gasket

exhaust valve lifter cable -> exhaust lifter cable -> cable

Figure 1: Examples of Variation from a Manual

Unlike general vocabulary, technical terminology is remarkably fixed. It is considered good style, in technical writing, to maintain, as much as possible, a one-to-one relationship between "terms" and "concepts," the things that terms refer to. It would be confusing to use two terms for the same thing (synonyms), or one term for two things (polysemy). Sager [17] makes this point by referring on page 125 to a British Standards Guide (BS 0, 1981, *A Standard for Standards*):

> TERMINOLOGY: Properly defined, unequivocal terminology is essential. Terminology within a standard should be consistent, so that the same object or concept is always described or expressed by the same term and not by the use of synonyms. ...

However, some very limited variation is possible. Sager [17] p. 215, mentions that it is sometimes possible to omit words in a long complex noun phrase. Figure 1 were selected from a manual, which by its instructive function, as Sager points out, would normally be expected to use stylistic variation very sparingly.

We find a similar pattern in Japanese. Technical terms are used very precisely. Just as with English, it is confusing for one term to refer to two things (polysemy) or two terms to refer to the same thing (synonymy). Stylistic variation is used very sparingly. Two instances of the same term are likely to match character for character.

We will introduce a dynamic programming method to take advantage of these kinds of variations, where it is possible to insert or delete a few characters in a term. Most of the characters of the term are kept intact. This kind of variation rarely changes the order of the remaining characters very much. That is why we believe that it would be better to model the variation with dynamic programming than to completely abandon the order and consider the string to be a bag of words or a bag of characters.

For example, it is possible to add or delete a word in a long Japanese phrase, at least in certain cases, so that *machine translation system* and *machine translation EXPERIMENTAL system* could both be used to refer to more or less the same thing. Especially when the phrase becomes quite long, as in the case of *GEOGRAPHICAL information-NO retrieval system*, the variation not only involves the Kanji characters corresponding to *GEOGRAPHICAL*, but can also involve a Hiragana character, *NO*.

2 Two Baseline Systems

Before introducing our dynamic programming proposal in detail, we will introduce two baseline systems. There are two types of systems in the literature, word-based systems and ngram-based systems. Baseline-Dict is intended to be representative of the first kind. We use an off-the-shelf program Chasen [13] to tokenize the query and the documents into words, and then we use a cosine measure with IDF weighting, as suggested by Salton [18]. Chasen uses a large dictionary of Japanese words to tokenize a sequence of characters into words. Chasen also assigns parts of speech. We use the nouns, verbs and unknown words as terms. Words with other parts of speech are considered stop words. More precisely,

Definition 1 *Let* $Score(t)$ *be* $-\log_2(df(t)/N)$.

$$SIM_{dict} = \sum_t tf(t) \cdot Score(t) \quad (1)$$

where term *t* appears in both the query and a document as a noun, verb or unknown word. *tf(t)* is term frequency in the document. *df(t)* is document frequency of term *t*. *N* is the total number of documents.

The similarity function for Baseline-Ngram is described precisely below. We also use a cosine measure with IDF weighting for all ngrams that appear in both a query and a document.

Definition 2 *Let* α, β, ξ *and* η *be strings. Let* α_{ik} *be the substring of* α *from position i to i+k-1. Let* β_{jk} *be the substring* β *from j to j+k-1. Let Score be a function from strings to reals.*

$$SIM_{ngram} = \sum_{i,j,k} Comp(\alpha_{ik}, \beta_{jk}) \quad (2)$$

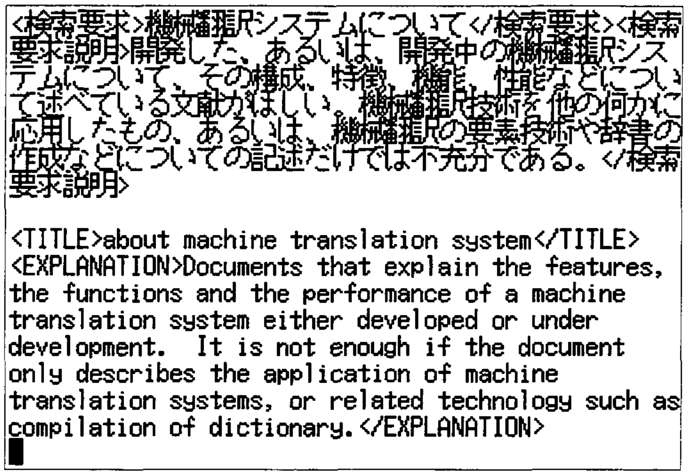
where $Comp(\xi, \eta)$ *is defined as follows:*

- *if* $\xi = \eta$ *then* $Score(\xi)$
- *if* $\xi \neq \eta$ *then 0.0*

where $Score(\xi) = -\log_2(df(\xi)/N)$.

In the definition, $df(\xi)$ is the document frequency of the substring ξ, just like the standard definition of *df* except that it applies to substrings rather than words.

Most character-based systems tokenize the query and the document into short, possibly overlapping, ngrams of characters such as bigrams or trigrams. Japanese text



```
<検索要求>機械翻訳システムについて</検索要求><検索
要求説明>開発した、あるいは、開発中の機械翻訳シス
テムについて、その構成、特徴、機能、性能などについ
て述べている文献がほしい。機械翻訳技術を他の何かに
応用したもの、あるいは、機械翻訳の要素技術や辞書の
作成などについての記述だけでは不充分である。</検索
要求説明>

<TITLE>about machine translation system</TITLE>
<EXPLANATION>Documents that explain the features,
the functions and the performance of a machine
translation system either developed or under
development.  It is not enough if the document
only describes the application of machine
translation systems, or related technology such as
compilation of dictionary.</EXPLANATION>
```

Figure 2: Query #24 and Its Translation

makes use of three alphabets, a very large alphabet of several thousand Kanji characters borrowed from Chinese a long time ago, plus two much smaller alphabets, about 50 Katakana characters for more recent loan words and a handful of Hiragana characters for Japanese function words. Because the Katakana alphabet is so much smaller than Kanji, Katakana words tend to be longer than Kanji words.

For example, the Kanji words for *machine* and *translation* are both two characters long, whereas the Katakana word for *system* is four characters long.

As pointed out by Fujii and Croft [6], short ngrams may work quite well for Kanji, but probably not for Katakana. Short ngrams may be more promising in Chinese, since it consists entirely of Chinese characters. See [2,10] for a recent discussion of short ngrams systems for Chinese, and [11] for Korean. Even so, it isn't necessary to restrict our attention to just short substrings. Using sophisticated data structures such as suffix arrays and PAT-trees [3,12,15,21], it is possible to work with very long ngrams.

We have found that long ngrams help. The bigram system has rather low performance (11 point average precision of 0.134), much lower than the Baseline-Ngram system (11 point average precision of 0.164). The Baseline-Ngram system uses all ngrams (substrings) in the query, not just the ngrams of length two. Since our proposed DP system uses all ngrams, we have chosen the character based system using all ngrams.

Figure 2 shows one of the 30 queries in the test collection, query #24. The query contains the Japanese term for *machine translation system*, and very little else that any of our retrieval systems are able to make use of. Both of the baseline systems have trouble with this query.

Baseline-Dict tokenizes *machine translation system* into the three Japanese words corresponding to *machine*, *translation* and *system*.

Unfortunately for Baseline-Dict, none of these three words are very useful keywords by themselves. In fact, *system* is particularly poor: it appears in about half of the 330,000 documents, so many that its IDF weighted contribution is negligible. Yet, the word *system* is crucial to the query, since, as the last sentence of the query explains, documents that mention *machine translation* but not *system* are considered irrelevant.

Baseline-Ngram is in a better position to deal with this term, because it will give a very high score to a document that contains the entire term, *machine translation system*, and lesser scores to documents that contain only fragments of the term. Although Baseline-Ngram does better than Baseline-Dict, at least for this query, Baseline-Ngram does not do a very good job of capturing the kind of variations of technical terms. In fact, for most of the queries in our test set, Baseline-Ngram performs remarkably poorly, much worse than Baseline-Dict.

Although the *Score* function could be more sophisticated, we use a function based on pure IDF. This is because we are more interested in relative improvements from one condition to another than absolute score. While the more sophisticated *Score* function will improve the absolute scores, it makes the comparison unclear. The terminology issues is real and independent of the choice of *Score* function.

3 Edit Distance and the Proposed System

Edit distance is a natural way to think about the variation between *machine translation system* and *machine translation EXPERIMENTAL system*, where a few insertions and deletions are possible. Edit distance is described in the appendix of the book [9]; it is the minimum number of edit operations, such as insertion and deletion, that are required to map one string into the other. It is widely used in spelling correction. We have tried to use a variation of this distance measure for information retrieval.

A few of modifications are required in order to achieve respectable precision. We need to introduce IDF-weights on substrings, so that not all edit operations count equally. This is accomplished by the function *Score*. In addition, because the weighting function can assign very good scores to long substrings (technical terms), we need to generalize the dynamic programming algorithm that is usually used to compute edit distances. Normally, the dynamic programming procedure considers just a single character at a time, but because long substrings can receive such good scores, we need to consider all prefixes of the longest common prefix, not just the next character.

We call the resulting similarity SWS (*String Weight dynamic programming Similarity*). It resembles Ukkonen's Enhanced Dynamic Programming ASM (Approximate String Matching) [1], but it is based on the weights of strings, not characters.

Definition 3 *Let α, β, ξ and η be strings. Let α_{km} be the substring of α from position k to k+m-1. Let α_{n*} be the substring of α from position n to the last. Let β_{km} be the substring β from k to k+m-1. Let β_{n*} be the substring of β from position n to the last. Let Score be a function from strings to reals.*

$$SIM_{DP} = \max_{i,j}(Comp(\alpha_{1i}, \beta_{1j}) + SIM_{DP}(\alpha_{i+1*}, \beta_{j+1*})) \quad (3)$$

where $Comp(\xi, \eta)$ is defined as follows:

- *if $\xi = \eta$ then* $\mathbf{Score}(\xi)$
- *if $\xi \neq \eta$ then 0.0*

where $Score(\xi) = -\log_2(df(\xi)/N)$.

The standard one-character-at-a-time dynamic programming method for computing edit distance assumes that long matches can't receive exceptionally good scores.

In other words, it regards as if $Score(\xi)$ is 0 if length of ξ is greater than one. If the scoring function obeys the inequality, $Score(\delta\gamma) \leq Score(\delta) + Score(\gamma)$, for all substrings δ and γ, then the best path would consist of a sequence of single characters, and we would not need to consider long phrases. However, the scoring function we have in mind is not like this. They will sometimes assign very good scores to long phrases (technical terms), and therefore we need to extend the dynamic programming procedure to consider more than just one character at a time.

Our experiments reported use IDF as the scoring function as the two baseline systems.

4 Experimental Results

As mentioned above, we used the Nacsis [7,8] test set which consists of 330,000 technical abstracts in Japanese, plus relevance judgements. We have used 30 queries in the first distribution. We report results for three systems: our proposal (DP), Baseline-Dict (BD) and Baseline-Ngram (BN). Many of the queries contain a single technical term, and very little else that any of the three systems can make use of.

Table 1: Dynamic programming (DP) shows best 11 point average precision with IDF term weight.

System	11pt
DP	0.281
Baseline-Ngram	0.164
Baseline-Dict	0.154

Table 2: Dynamic programming (DP) is better than both Baseline-Dict and Baseline-Ngram

	worse	better
DP vs BD	7	23
DP vs BN	1	29
BD vs BN	15	15

Table 1 shows 11 points average precision of all queries. Table 1 suggests that DP is better than both Baseline-Dict and Baseline-Ngram. Table 2 show the comparisons in each single query. Each pair of systems was tested on all 30 queries. Judgements (better, worse) were made by the 11 points average precision of each query. We can conclude DP shows better performance than both Baseline-Dict and Baseline-Ngram. The score 23 vs 7 is statistically significant with the level of 2.6×10^{-3}. The score 29 vs 1 is also statistically significant with the level of 2.9×10^{-8}.

Table 3 shows more detail for 13 of the 30 queries. A plus sign indicates good performance, and a minus sign indicates poor performance. The first three lines are easy for all the systems since the term and many of the words that make up the term are good keywords, as indicated by their IDF weights. The next six lines play into DP's strength. The term is a much stronger clue than its constituent words. The numbers in parentheses indicate how many times the term was found verbatim in the corpus, not counting the variant forms, which were also attested. The last four lines play into BD's strength. In these cases, the query is described in general vocabulary, and does not mention a technical term.

Four of the queries in Table 3 are marked with an asterisk. Recall plots for these four queries are shown in Figure 3 - Figure 6. When there are differences, the differences are most salient in the grey area. Mitra *et al.* [13] also found that phrases made little difference for the top few documents.

Table 3: Dynamic programming (DP) works well when the query mentions a technical term, and each of the words in the term have little IDF weight.

#	BD	BN	DP	Query	Comment
1*	+	+	+	autonomous mobile robot	easy task
4	+	+	+	document image understanding	easy task
6	+	+	+	Agent	easy task
12*	-	+	+	data mining	no variation
16	-	-	+	parse tree analysis (56)	variation
23	-	-	+	news paper article data (22)	variation
24*	-	-	+	machine translation system (244)	variation
30	-	-	+	data flow processor (9)	variation
19	-	-	+	natural language knowledge acquisition	only variation
5	+	-	-	reduction of space dimension	general term
10	+	-	-	automatic extraction of keyword	general term
13	+	-	-	analysis of loop region	general term
29*	+	-	-	measurement of position	general term

5 Conclusion

We have proposed a new dynamic programming method and shown that it works well especially when the query consists of a single technical term such as *machine translation system*, and the words in the term are poor keywords with low IDF weights. Since we have a characterization of when the proposed method works well, it ought to be possible to combine the new dynamic programming method with more traditional methods to form a hybrid system that would get the best of both worlds.

References

1 H. Berghel and D. Roach. An Extension of Ukkonen's Enhanced Dynamic Programming ASM Algorithm. *ACM Transactions on Information Systems*, 1996, 14(1), pp. 94 - 106.

2 Aitao Chen, Jianzhang He, Liangjie Xu, Fredric C. Gey, and Jason Meggs. Chinese text retrieval without using a dictionary. *In SIGIR97*. 1997, pp. 42 - 49.

3 Lee-Feng Chien. Pat-tree-based keyword extraction for chinese information retrieval. *In SIGIR97*. 1997.

4 Marc. Damashek. Gauging similarity with n-grams: Language independent categorization of text. *Science* Feb 1995, 267(10), pp. 843 - 848.

5 J. L. Fagan. Experiments in Automatic Phrase Indexing For Document Retrieval: A Comparison of Syntactic and Non-Syntactic Methods. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1987.

6 Hideo Fujii and W. Bruce Croft. A comparison of indexing techniques for japanese text retrieval. *In SIGIR93*. 1993, pp. 237 - 246.

7 K Kageura and et al. Nacsis corpus project for ir and terminological research. *In Natural Language Proceeding Pacific Rim Symposium 97*. Dec 1997, pp. 2 - 5.

8 Noriko Kando and et al. Ntcir: nacsis test collection project. *In 20*th Annual Colloquium of BCSIRSG. Mar 1997, pp. 25 - 27.

9 Robert R. Korfhage. Information Storage and Retrieval, chapter Appendix A, pp. 300 - 304. WILEY COMPUTER PUBLISHING, Johon Wiley & Sons, Inc., USA, 1997.

10 K. L. Kwok. Comparing representations in chinese information retrieval. *In SIGIR97*. 1997, pp.34 - 41.

11 Joon Ho Lee and Jeong Soo Ahn. Using n-grams for korean text retrieval. *In SIGIR96*. 1996, pp. 216 - 224.

12 Udi Manber and E. Myers, Suffix array: A new method for on-linestring searchcs. *SIAM Journal on Computing*. 1993, 22(5), pp. 935 - 948.

13 Yuji Matsumoto, Akira Kitauchi, Tatsuo Yamashita, Yoshitaka Hirano, Osamu Imaichi, and Tomoaki Imamura. Japanese morphological analysis system chasen manual. Technical Report NAIST-IS-TR97007, NAIST, Nara, Japan, Feb 1997.

14 M. Mitra, C. Buckley, A. Singhal, and C. Cardie. An analysis of statistical and syntactic phrases. *In RIAO97*, 1997, pp. 200 - 214.

15 M. Nagao and S. Mori. A new method of n-gram statistics for large number of n and automatic extraction of words and phrases from large text data of japanese. *In Coling94*. 1994, pp. 611 - 615.

16 Yasushi Ogawa and Toru Matsuda. Overlapping statistical word indexing: a new indexing method for japanese text. *In SIGIR97*. 1997, pp. 226 - 234.

17 Juan C. Sager. A Practical Course in Terminology Processing. John Benjamins Publishing company, Amsterdam/Philadelphia, 1990.

18 Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval *Information Proceeding and Management,* 1988, 24, pp. 513 - 523.

19 T. Strzalkowski, L. Guthrie, J. Karlgren, J. Leistensnider, F. Lin. Perez-Carballo, T. Straszheim, J. Wang, and J. Wilding. Natural language information retrieval: Trec-5 report. *In The Fifth Text REtrival Conference (TREC-5),* E. M. Voorhees and D. K. Harman editors, 1996, pp. 291 - 314.

20 Jan van Leeuwen. Handbook of Theoretical Computer Science. The MIT Press/Elsevier, 1990.

21 Mikio Yamamoto and Kenneth W. Church. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *In 6 th Workshop on Very Large Corpora*. 1998, pp. 28 - 37.

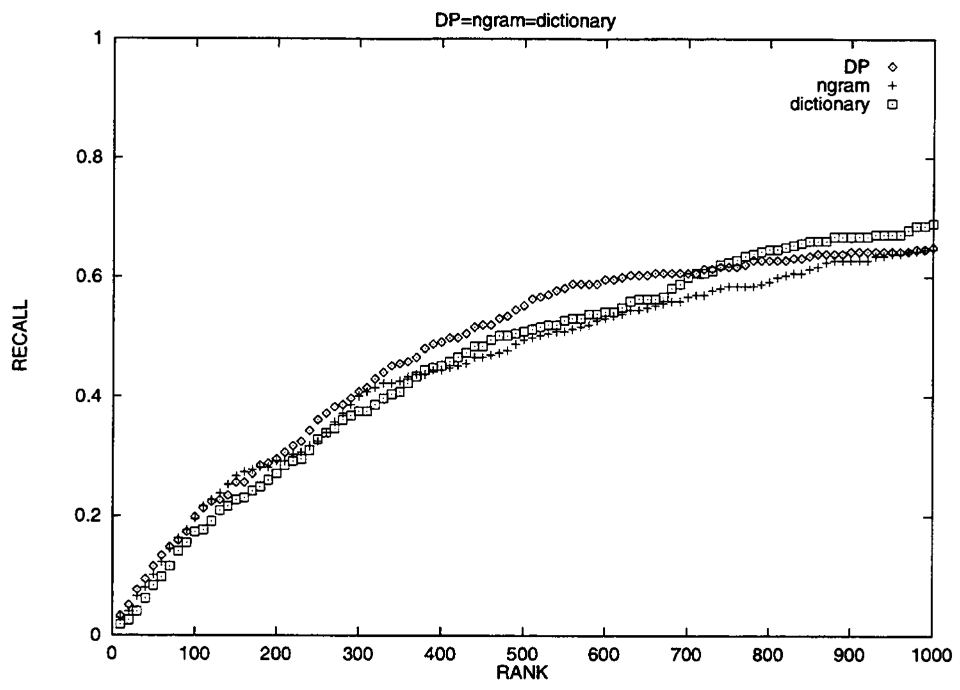


Figure 3: Query #1,* autonomous mobile robot*: easy for all three systems.

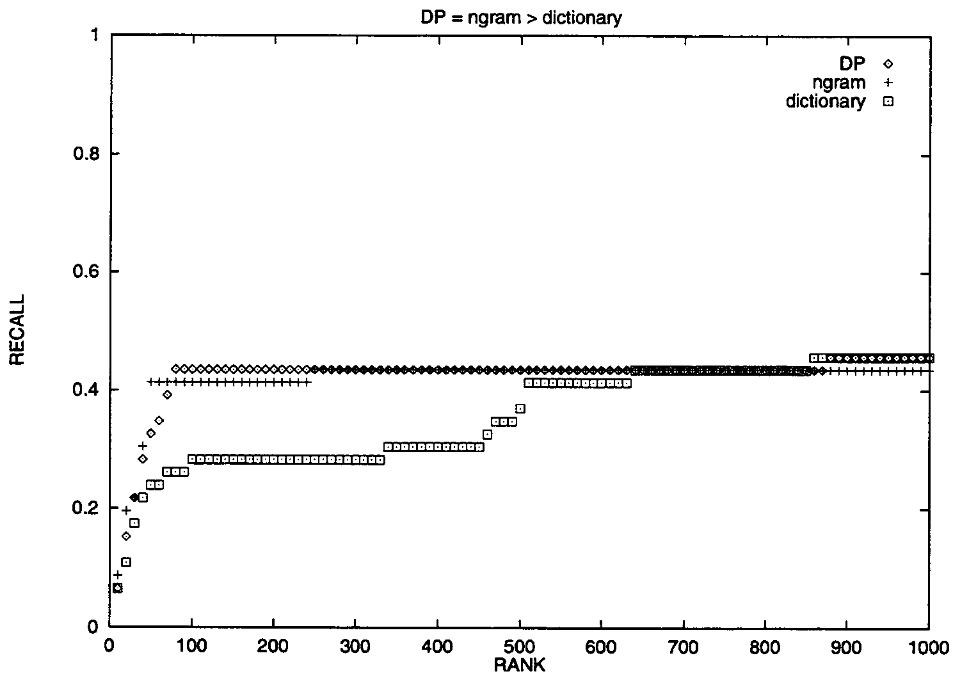


Figure 4: Query #12,* data mining*: hard for Baseline-Dict, because the word segmentator split* data *inappropriately.

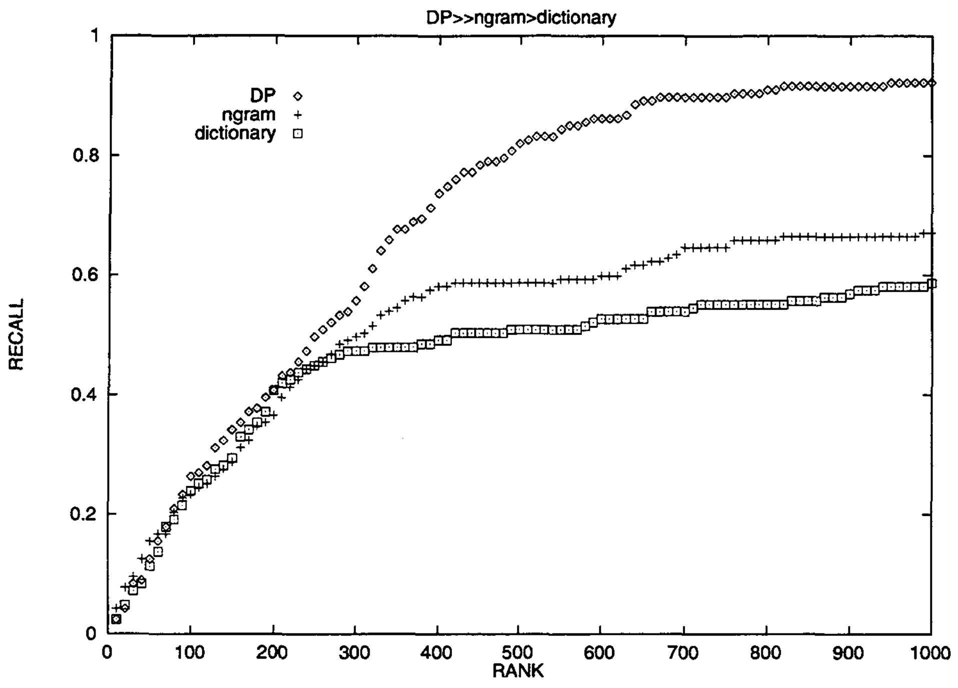


Figure 5: Query #24, machine translation system: ideal for dynamic programming (DP)

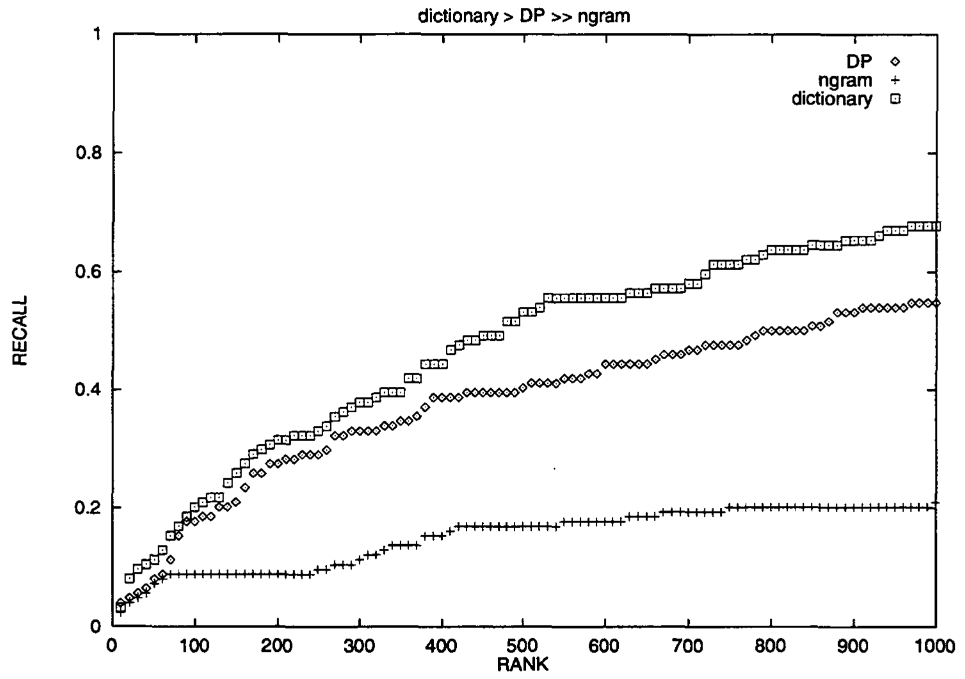


Figure 6: Query #29, **position + measurement*:*** ***hard for dynamic programming (DP) because there is no technical term.***