A Dynamic Programming Approach to Improving Translation Memory Matching and Retrieval Using Paraphrases

Rohit Gupta[1(✉)], Constantin Orăsan[1], Qun Liu[2], and Ruslan Mitkov[1]

[1] University of Wolverhampton, Wolverhampton, UK
R.Gupta@wlv.ac.uk
[2] Dublin City University, Dublin, Ireland

Abstract. Translation memory tools lack semantic knowledge like paraphrasing when they perform matching and retrieval. As a result, paraphrased segments are often not retrieved. One of the primary reasons for this is the lack of a simple and efficient algorithm to incorporate paraphrasing in the TM matching process. Gupta and Orăsan [1] proposed an algorithm which incorporates paraphrasing based on greedy approximation and dynamic programming. However, because of greedy approximation, their approach does not make full use of the paraphrases available. In this paper we propose an efficient method for incorporating paraphrasing in matching and retrieval based on dynamic programming only. We tested our approach on English-German, English-Spanish and English-French language pairs and retrieved better results for all three language pairs compared to the earlier approach [1].

Keywords: Edit distance with paraphrasing · Translation memory · TM matching and retrieval · Computer aided translation · Paraphrasing

1 Introduction

Apart from retrieving exact matches, one of the core features of a TM system is the retrieval of previously translated similar segments for post-editing in order to avoid translation from scratch when an exact match is not available. However, this retrieval process is generally limited to edit-distance based measures operating on surface form (or sometimes stem) matching. Most commercial systems use edit distance [2] or some variation of it. Although these measures provide a strong baseline, they are not sufficient to capture semantic similarity between segments as judged by humans. For example, even though segments like *I would like to congratulate the rapporteur* and *I wish to congratulate the rapporteur* have the same meaning, current TM systems will consider them not similar enough to use one instead of the other. The two segments have only 71 % similarity based on word based Levenshtein edit-distance, even though one segment is a paraphrase of the other segment. To mitigate this limitation of TM, we propose an approach to incorporating paraphrasing in TM matching.

P. Sojka et al. (Eds.): TSD 2016, LNAI 9924, pp. 259–269, 2016.
DOI: 10.1007/978-3-319-45510-5_30

A trivial approach to implementing paraphrasing along with edit-distance is to generate all the paraphrased segments based on the paraphrases available and store these additional segments in the TM. This approach leads to a combinationtorial explosion and is highly inefficient both in terms of time necessary to process and space to store. For a TM segment which has n different phrases where each phrase can be paraphrased in m more possible ways, we get $(m+1)^n - 1$ additional segments (still not considering that these phrases may contain paraphrases as well). To measure how many segments will be generated in reality, we randomly selected a sample of 10 segments from the TM and used the same paraphrase database as used in our experiments to generate additional segments. By limiting only to not more than three phrases to be paraphrased per segment we generated 1,622,115 different segments.

This paper presents a simple, novel and efficient approach to improve matching and retrieval in TM using paraphrasing based on dynamic programming. Our tool is available on Github.[1]

2 Related Work

Several researchers have pointed out the need for more advanced processing in TMs that go beyond surface form comparisons and proposed the use of semantic or syntactic information, but their methods are inefficient for large TMs [3–8].

Macklovitch and Russell [4] showed that using NLP techniques like named entity recognition and morphological processing can improve matching in TM, whilst Somers [5] highlighted the need for more sophisticated matching techniques that include linguistic knowledge like inflection paradigms, synonyms and grammatical alterations. Both Planas and Furuse [3] and Hodász and Pohl [6] proposed to use lemmas and parts of speech along with surface form comparison. Hodász and Pohl [6] extended the matching process to a sentence skeleton where noun phrases are either tagged by a translator or by a heuristic NP aligner developed for English-Hungarian translation. Planas and Furuse [3] tested a prototype model on 50 sentences from the software domain and 75 sentences from a journal with TM sizes of 7,192 and 31,526 segments respectively. A fuzzy match retrieved was considered usable if less than half of the words required editing to match the input sentence. The authors concluded that the approach gives more usable results compared to Trados Workbench, an industry standard commercial system, used as a baseline.

Pekar and Mitkov [7] presented an approach based on syntactic transformation rules. They evaluated the proposed method using a query sentence and found that the syntactic rules help in retrieving better segments. The use of alignments between source and target at word, phrase or character level was proposed in [9] as a way of improving matching.

Recent work focused on approaches which use paraphrasing in TM matching and retrieval [1,10,11]. Utiyama et al. [10] developed a method which relies on a finite state transducer limited to exact matches only. Gupta and Orăsan [1]

[1] https://github.com/rohitguptacs/TMAdvanced.

proposed a paraphrasing approach based on greedy approximation and dynamic programming. Gupta et al. [11] showed that post-editing time is reduced if paraphrases are used in the TM matching. Timonera and Mitkov [12] show clause splitting as a preprocessing stage improves the matching and retrieval.

The approach proposed in this paper is similar to the one described in [1] but instead of using greedy approximation we use dynamic programming which optimises edit-distance globally and makes the approach more simple and efficient.

3 Our Approach

In this section, we present our approach to include paraphrasing in the TM matching and retrieval process. In order to be able to compare our approach with the one proposed in [1], we use the same settings for the experiments. In the rest of the paper we will refer to the method proposed in [1] as DPGA (Dynamic Programming and Greedy Approximation) and the method proposed in this paper as DP (Dynamic Programming only).

3.1 Paraphrase Corpus and Classification

We use the PPDB 1.0 paraphrase database [13] for our work. This database contains lexical, phrasal and syntactic paraphrases automatically extracted using a large collection of parallel corpora. The paraphrases in this database are constructed using a bilingual pivoting method. The paraphrase database is available in six sizes (S, M, L, XL, XXL, XXXL) where S is the smallest and XXXL is the largest; we use size L (lexical and phrasal) in this research.

We use the four types proposed in [1] for classifying paraphrases on the basis of the number of words they contain (common part is shown in **bold** and can be removed after considering the context when computing edit-distance):

1. Paraphrases having one word in both the source and target sides, e.g. "period" ⇔ "duration"
2. Paraphrases having multiple words on both sides but differing in one word only, e.g. "in **the period**" ⇔ "during **the period**"
3. Paraphrases having multiple words, but the same number of words on both sides, e.g. "laid down **in article**" ⇔ "set forth **in article**"
4. Paraphrases in which the number of words in the source and target differ, e.g. "**a reasonable period** of time to" ⇔ "**a reasonable period** to"

The classification of paraphrases helps to implement the Type 1 and Type 2 paraphrases more efficiently (see Sect. 3.4 for further details).

3.2 Matching Steps

There are two options for incorporating paraphrasing in a typical TM matching pipeline: to paraphrase the input or to paraphrase the TM. For our approach we have chosen to paraphrase the TM on fly because it allows retrieval of matches in real time.

Our approach is also inspired by the one proposed in [1] and comprises of the following steps:

1. Read the Translation Memories available
2. Classify and store all paraphrases for each segment in the TM in their reduced forms according to the types presented in Sect. 3.1
3. Read the file that needs to be translated
4. For each segment in the input file
   (a) retrieve the potential segments for paraphrasing in the TM according to the filtering steps of Sect. 3.3
   (b) search for the most similar segment based on the approach described in Sect. 3.4
   (c) retrieve the most similar segment if it is above a predefined threshold

3.3 Filtering

Before processing begins, several filtering steps are applied to each input segment. The purpose of this filtering process is to remove unnecessary candidates and speed up the processing. These steps, based on [1], are as follows:

1. lenFilter: Filter out segments based on length. If segments differ considerably in length, the edit-distance will also differ correspondingly. In our case, TM segments are discarded if the TM segments are shorter than 39 % of the input or vice-versa.
2. simFilter: Filter out segments based on baseline edit-distance similarity. TM segments which have a similarity below a certain threshold will be removed. In our case, the threshold was set to 39 %.
3. maxFilter: Next, after filtering the candidates with the above two steps we sort the remaining segments in decreasing (non-increasing) order of baseline edit-distance similarity and pick the top 100 segments.
4. beamFilter: Finally, segments within a certain range of similarity with the most similar segment are selected for paraphrasing. Here, the range adopted is 35 %. This means that if the most similar segment has 95 % similarity, segments with a similarity below 60 % will be discarded.

3.4 Edit-Distance with Paraphrasing Computation

For our implementation, we use the basic edit-distance procedure [2], which is word-based edit-distance with cost 1 for insertion, deletion and substitution. We obtain the similarity between two segments by normalising edit-distance with the length of the longer segment.

We have employed the basic edit-distance as a baseline and adapted it to incorporate paraphrasing. When edit-distance is calculated, the paraphrases of Types 1 and 2 can be implemented in a more efficient manner than for paraphrases of Types 3 and 4. They have the unique property that they can be reduced to single word paraphrases by removing the other matching words. The

basic procedure works by comparing each token one by one in the input segment with each token in the TM segment. This procedure makes use of previous edit-distance computations to optimise the edit-distance globally (for the whole sentence). In our dynamic programming approach, at every step we consider the best matching path.

Table 1. Edit-distance calculation

i	j	0	1	2	3	4	5	6	10	7	8	9	11
		#	the	period duration time	laid	down	referred	to	in under	provided	for	by	article
0	#	0	1	2	3	4	3	4	5	3	4	5	6
1	the	1	0	1	2	3	2	3	4	2	3	4	5
2	period	2	1	0	1	2	1	2	3	1	2	3	4
3	referred	3	2	1	1	2	0	1	2	1	2	3	3
4	to	4	3	2	2	**2**	1	**0**	1	2	2	3	2
5	in	5	4	3	3	3	2	1	**0**	3	3	**3**	1
6	article	6	5	4	4	4	3	2	1	4	4	4	0

Algorithm 1. Edit-Distance with Paraphrasing using Dynamic Programming

procedure EDIT-DISTANCE($InputSegment, TMLattice$)
  $M \leftarrow$ length of $TMLattice$ ▷ number of nodes in a TM lattice
  $N \leftarrow$ length of $InputSegment$ ▷ number of nodes in a input segment
  Initialise D, a two dimensional array of size $M \times N$ initialized with distance from null # ▷ See Table 1, all values at $i = 0$ and all values at $j = 0$
  for $j \leftarrow 1...M$ **do**
    for $i \leftarrow 1...N$ **do**
      $cost \leftarrow 1$ ▷ substitution cost
      if $TMLattice_j = InputSegment_i$ **then**
        $cost \leftarrow 0$ ▷ substitution cost if matches
      if $cost = 1$ **and** $TMLattice_j$ is a TM token **then** ▷ condition to avoid paraphrasing the paraphrase
        $OneWordPP \leftarrow$ get Type 1 and Type 2 paraphrases associated with TMToken
        if $InputToken \in OneWordPP$ **then** ▷ applying type 1 and type 2 paraphrasing
          $cost \leftarrow 0$
      $P \leftarrow$ previous indices of paths at $TMLattice_j$ ▷ get indices of previous nodes connecting to the present node $TMLattice_j$
      $D[i, j] \leftarrow minimum(D[i-1, j] + 1, D[i, k] + 1, D[i-1, k] + cost$ for all $k \in P)$ ▷ store minimum edit distance by considering insertion, deletion or substitution for all paths
  $k \leftarrow$ index of minimum edit distance path at last node
  $N \leftarrow$ length of $InputSegment$ ▷ number of tokens in a input segment
  Return $D[k, N]$ ▷ Return minimum edit-distance

Table 1 illustrates the edit-distance calculation of the first five tokens of the Input and TM segment with paraphrasing of the example given in Fig. 1. The second column represents the input segment (*the period referred to in article*) and

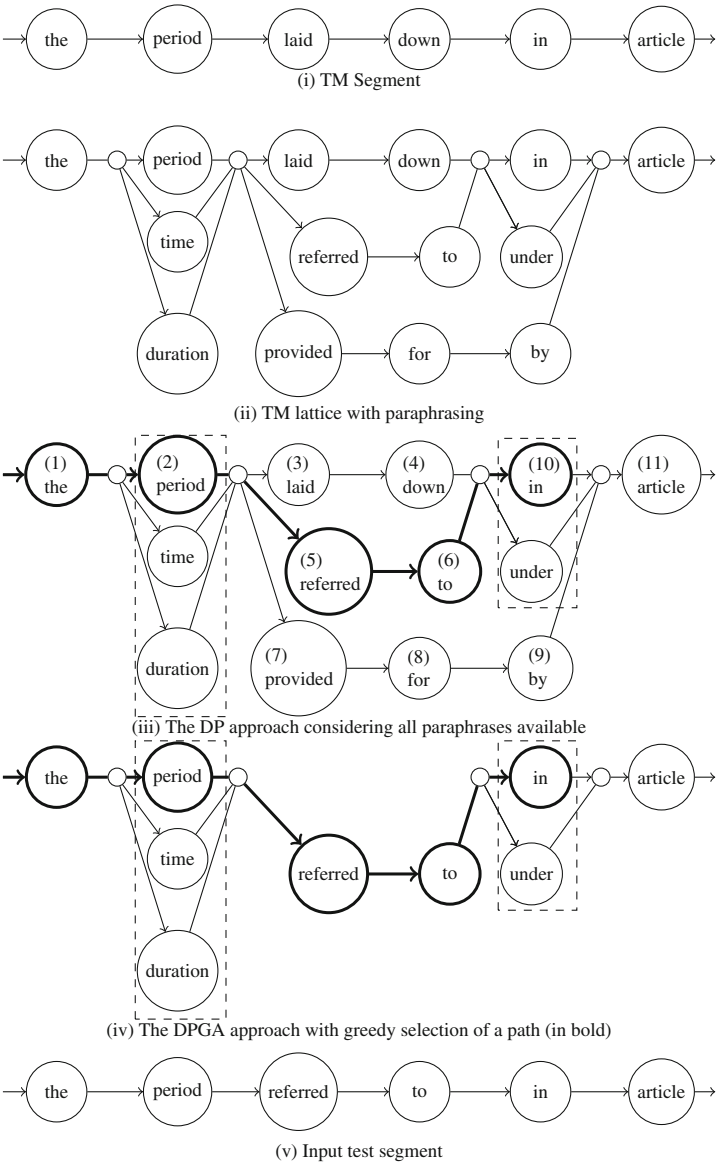

Fig. 1. (i) TM segment, (ii) TM lattice after considering paraphrases, (iii) The DP approach considering all paraphrases available (numbers show values of j as given in Table 1, dashed area indicates Types 1 and 2 paraphrasing), (iv) DPGA with greedy selection, (v) Input test segment

the second row represents the TM segment (*the period laid down in article*) along with the paraphrases. Figure 1(v) shows the input test segment and Fig. 1(ii) shows the TM lattice. In Algorithm 1, *InputSegment* is the segment that to be

translated and *TMLattice* is the TM lattice (TM segment with paraphrasing). Table 1 shows the edit-distance calculation of the first five tokens of the input segment and TM lattice. In Table 1, if a word from the input segment matches any of the words "period", "time" or "duration", the cost of substitution will be 0. In Algorithm 1, *line 14* gets the previous indices at every step.

For example, in Table 1, when executing the token 'in' the algorithm will consider previous indices of 'down' and 'to' (see lattice in Fig. 1 (ii)) and store the minimum edit-distance path in *line 15*. As we can see in Table 1, the column 'in' is updated after considering both column 'down' and 'to' and column 'article' is updated after considering column 'in' and 'by'. In contrast to the DP approach, the DPGA approach makes a greedy selection and will update only on the bases of selected paraphrases and not all paraphrase paths as shown in Fig. 1.

3.5 Computational Considerations

The time complexity of the basic edit-distance procedure is $O(m\,n)$ where m and n are lengths of source and target segments, respectively. After employing paraphrasing of Type 1 and Type 2, the complexity increases to $O(m\,n\,log(p))$, where $p-1$ is the number of additional paraphrases of Type 1 and Type 2 per token of TM segment. Employing paraphrasing of Type 3 and Type 4 further increases the edit-distance complexity to $O(m\,n(log(p)+q\,l))$, where q is the number of Type 3 and Type 4 paraphrases stored per token and l is the average length of a Type 3 and Type 4 paraphrase. The time complexity of the DPGA approach is $O(l\,m\,n(log(p)+q))$, which is slightly more compared to $O(m\,n(log(p)+q\,l))$ complexity the DP approach. However, in practice we have not observed much difference in the speed of both approaches.

If we consider, Type 1 and Type 2 paraphrases in the same manner (comparing sequentially instead of searching in a list) as Type 3 and Type 4, the time complexity of the DP approach can be simply written as $O(k\,n)$, where k is the number of nodes in the TM lattice and n is the number of nodes in the input segment.

4 Experiments and Results

In this section, we present our experiments and the results obtained. For our experiments we used the English-French, English-German and English-Spanish pairs of the 2014 release of the DGT-TM corpus [14]. From this corpus, we filtered out segments of fewer than seven words and more than 40 words; the remaining pairs were used to create the TM and Test dataset. The test sets for all language pairs contain 20,000 randomly selected unique segments and the rest are used as the TM. The statistics of the datasets are given in Table 2.

When we use paraphrasing in the matching and retrieval process, the fuzzy match score of a paraphrased segment is increased, which results in the retrieval of more segments at a particular threshold. This increment in retrieval can be classified into two types: without changing the top rank and by changing the

Table 2. DGT-TM corpus statistics

	English-German		English-French		English-Spanish	
	TM	Test set	TM	Test set	TM	Test set
Segments	204,776	20,000	204,713	20,000	202,700	20,000
Source words	4,179,007	382,793	4,177,332	382,358	4,140,473	383,694
Target words	3,833,088	343,274	4,666,196	407,495	4,783,178	433,450

top rank. For example, for a particular input segment, we have two segments: A and B in the TM. Using simple edit-distance, A has a 65 % and B has a 60 % fuzzy score; the fuzzy score of A is better than that of B. As a result of using paraphrasing, we observe two types of score changes:

1. the score of A is still better than or equal to that of B, for example, A has 85 % and B has 70 % fuzzy score;
2. the score of A is less than that of B, for example, A has 75 % and B has 80 % fuzzy score.

In the first case, paraphrasing does not supersede the existing model and just facilitates it by improving the fuzzy score so that the top segment ranked using edit distance gets retrieved. However, in the second case, paraphrasing changes the ranking and now the top-ranked segment is different. In this case, the paraphrasing model supersedes the existing simple edit distance model. This second case also gives a different reference with which to compare. In the experiments reported below, we take the top segment retrieved using simple edit distance as a reference against the top segment retrieved using paraphrasing and compare to determine which one is better.

We performed experiments using both approaches (DPGA and DP). For both of them, we conducted experiments in two different preprocessing settings: In setting 1 (S1), we do not remove any punctuation marks and in the preprocessing step we perform only tokenization; In setting 2 (S2), along with tokenization, we also remove punctuation marks in the preprocessing stage. The DGT-TM dataset is of legal genre and contains many punctuation marks. We deleted the punctuation marks to see how it affects the baseline edit-distance and the edit-distance with paraphrasing. In S2, punctuation marks are also removed from the target side.[2]

Table 3 presents our results. We measure an increase in the number of segments in an interval when using paraphrasing as well as quality of those segments against simple edit-distance. Table 3 shows similarity threshold intervals (TH) for TM (the threshold intervals shown are on the basis of edit-distance with paraphrasing similarity), the total number of segments retrieved

[2] We have used Stanford tokenizer on the English side and tokenizer provided with Moses [15] on the target side. The source (English) tokenization is used for matching and target language tokenization is used when calculating BLEU score.

Table 3. Results on English-German (DE), English-French (FR) and English-Spanish (ES)

			DP Approach				DPGA Approach			
		TH	**100**	**[85, 100)**	**[70, 85)**	**[55, 70)**	**100**	**[85, 100)**	**[70, 85)**	**[55, 70)**
DE	S1	EditRetrieved	6629	1193	1029	1259	6629	1193	1029	1259
		+ParaRetrieved	54	114	146	445	44	88	89	244
		%Improve	0.81	9.56	14.19	35.35	0.66	7.38	8.65	19.38
		RankCh	7	8	27	217	4	5	21	115
		METEOR-EditRankCh	83.82	57.42	24.05	26.47	75.69	63.78	25.11	27.74
		METEOR-ParaRankCh	**90.76**	**64.15**	**33.88**	**26.89**	**89.93**	49.97	**28.53**	27.58
		BLEU-EditRankCh	72.79	38.21	12.13	15.38	65.26	50.27	6.67	17.69
		BLEU-ParaRankCh	**84.15**	**49.11**	**14.44**	14.40	**73.79**	23.68	**8.08**	15.61
	S2	EditRetrieved	6767	1078	889	1070	6767	1078	889	1070
		+ParaRetrieved	60	123	150	380	49	98	99	224
		%Improve	0.89	11.41	16.87	35.51	0.72	9.09	11.14	20.93
		RankCh	8	15	36	176	5	9	30	118
		METEOR-EditRankCh	80.01	57.79	37.99	24.90	69.20	49.19	38.58	25.00
		METEOR-ParaRankCh	**90.27**	**67.34**	**43.95**	**27.59**	**88.21**	**54.42**	**38.68**	**28.23**
		BLEU-EditRankCh	64.97	50.22	25.20	13.37	49.34	36.06	23.06	13.02
		BLEU-ParaRankCh	**84.90**	**54.43**	**29.67**	**14.51**	**77.85**	**36.38**	**23.48**	**14.99**
FR	S1	EditRetrieved	6611	1197	1040	1257	6611	1197	1040	1257
		+ParaRetrieved	54	116	141	443	44	90	87	241
		%Improve	0.82	9.69	13.56	35.24	0.67	7.52	8.37	19.17
		RankCh	7	8	26	217	4	5	20	115
		METEOR-EditRankCh	94.88	45.92	34.91	32.26	92.26	54.38	39.96	33.47
		METEOR-ParaRankCh	**96.17**	**70.23**	**42.92**	**32.70**	91.44	**61.28**	39.37	33.35
		BLEU-EditRankCh	78.21	17.81	20.06	19.75	79.60	28.87	18.76	20.87
		BLEU-ParaRankCh	**87.56**	**46.58**	**24.67**	18.15	69.32	27.42	17.19	17.50
	S2	EditRetrieved	6750	1082	894	1078	6750	1082	894	1078
		+ParaRetrieved	60	125	141	377	49	100	91	223
		%Improve	0.89	11.55	15.77	34.97	0.73	9.24	10.18	20.69
		RankCh	8	15	33	177	5	9	28	118
		METEOR-EditRankCh	81.73	50.35	49.02	31.45	71.18	41.02	50.27	31.17
		METEOR-ParaRankCh	**92.87**	**66.37**	**58.60**	**33.99**	**85.27**	**52.54**	**58.60**	**35.77**
		BLEU-EditRankCh	63.44	33.50	32.84	19.56	48.88	15.76	32.25	18.18
		BLEU-ParaRankCh	**84.80**	**47.38**	**39.70**	**20.02**	**68.68**	**28.31**	**37.85**	**18.77**
ES	S1	EditRetrieved	6620	1187	1028	1233	6620	1187	1028	1233
		+ParaRetrieved	54	115	141	433	44	89	86	234
		%Improve	0.82	9.69	13.72	35.12	0.66	7.50	8.37	18.98
		RankCh	7	8	29	209	4	5	21	110
		METEOR-EditRankCh	85.85	52.10	36.62	33.80	73.62	51.61	35.72	34.85
		METEOR-ParaRankCh	**88.45**	**64.83**	**44.48**	**34.58**	**83.36**	49.02	**44.94**	**35.93**
		BLEU-EditRankCh	72.79	24.74	17.49	19.20	55.75	38.20	14.49	20.11
		BLEU-ParaRankCh	**78.09**	**44.48**	**24.22**	19.09	**75.49**	18.61	**24.13**	19.48
	S2	EditRetrieved	6757	1076	885	1050	6757	1076	885	1050
		+ParaRetrieved	60	125	142	374	49	100	92	221
		%Improve	0.89	11.62	16.05	35.62	0.73	9.29	10.40	21.05
		RankCh	8	15	34	179	5	9	28	118
		METEOR-EditRankCh	82.11	58.01	49.32	35.13	69.89	51.35	49.58	35.46
		METEOR-ParaRankCh	**87.12**	**67.94**	**60.42**	**36.07**	**82.52**	**66.11**	**60.39**	**37.14**
		BLEU-EditRankCh	61.84	37.02	33.42	18.84	37.04	29.67	29.29	18.37
		BLEU-ParaRankCh	**73.83**	**47.07**	**44.08**	**19.82**	**69.89**	**44.80**	**42.02**	**19.83**

using the baseline approach (EditRetrieved), the additional number of segments retrieved using the paraphrasing approaches (+ParaRetrieved), the percentage increase in retrieval obtained over the baseline (%Improve), and the number of segments that changed their ranking and rose to the top because of paraphrasing (RankCh). BLEU-ParaRankCh and METEOR-ParaRankCh represent the BLEU score [16] and METEOR [17] score over translations retrieved by the DP approach for segments which changed their ranking and come up in

the threshold interval because of paraphrasing and BLEU-EditRankCh and METEOR-EditRankCh represent the BLEU score and METEOR score on corresponding top translations retrieved by the baseline approach. Table 3 also shows results obtained by the DPGA approach.

The DP approach presented in this paper retrieves more matches than the DPGA approach for all language pairs. We see that the DP approach retrieves better results compared to using simple edit-distance for all language pairs in both settings.

Table 3 shows that for S1 (English-German), the DPGA approach does not retrieve better results compared to simple edit-distance for threshold interval [85, 100) and [55, 70). We have observed that removing punctuation marks in the preprocessing stage not only increases the retrieval but also increases the improvement in retrieval using paraphrases. Table 3 shows that there is an improvement around 9.56 % improvement for S1 and 11.41 % for S2 in the interval [85, 100) for the DP approach on English-German. Table 3 also suggests that the DPGA approach is more sensitive to preprocessing performed which can be seen in the difference between S1 and S2 results for all language pairs. The quality of the retrieved segments is influenced more for the DPGA approach compared to the DP approach.

5 Conclusion

In this paper, we presented our new dynamic programming based approach to include paraphrasing in the process of translation memory matching and retrieval. Using the DP approach, depending on the preprocessing settings, we observed an increase in retrieval around 9 % to 16 % for threshold intervals [100, 85) or [85, 70). We observe that the number of matches increased when using paraphrasing in every interval and also have better quality compared to those retrieved by simple edit-distance. The DP approach also yields better results compared to the DPGA approach for all three language pairs.

Acknowledgement. The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme FP7/2007-2013/ under REA grant agreement No. 317471.

References

1. Gupta, R., Orăsan, C.: Incorporating paraphrasing in translation memory matching and retrieval. In: Proceedings of the European Association of Machine Translation (EAMT-2014) (2014)
2. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Sov. Phys. Dokl. **10**, 707–710 (1966)
3. Planas, E., Furuse, O.: Formalizing translation memories. In: Proceedings of the 7th Machine Translation Summit, pp. 331–339 (1999)

4. Macklovitch, E., Russell, G.: What's been forgotten in translation memory. In: White, J.S. (ed.) AMTA 2000. LNCS (LNAI), vol. 1934, pp. 137–146. Springer, Heidelberg (2000)
5. Somers, H.: Translation memory systems. Comput. Transl.: Transl. Guide **35**, 31–48 (2003)
6. Hodász, G., Pohl, G.: MetaMorpho TM: a linguistically enriched translation memory. In: International Workshop, Modern Approaches in Translation Technologies (2005)
7. Pekar, V., Mitkov, R.: New generation translation memory: content-sensivite matching. In: Proceedings of the 40th Anniversary Congress of the Swiss Association of Translators, Terminologists and Interpreters (2007)
8. Mitkov, R.: Improving third generation translation memory systems through identification of rhetorical predicates. In: Proceedings of LangTech 2008 (2008)
9. Clark, J.P.: System, method, and product for dynamically aligning translations in a translation-memory system, 5 February 2002. US Patent 6,345,244
10. Utiyama, M., Neubig, G., Onishi, T., Sumita, E.: Searching translation memories for paraphrases. In: Machine Translation Summit XIII, pp. 325–331 (2011)
11. Gupta, R., Orăsan, C., Zampieri, M., Vela, M., Van Genabith, J.: Can translation memories afford not to use paraphrasing? In: Proceedings of EAMT (2015)
12. Timonera, K., Mitkov, R.: Improving translation memory matching through clause splitting. In: Proceedings of the Workshop on Natural Language Processing for Translation Memories (NLP4TM), Hissar, Bulgaria, pp. 17–23 (2015)
13. Ganitkevitch, J., Benjamin, V.D., Callison-Burch, C.: PPDB: the paraphrase database. In: Proceedings of NAACL-HLT, Atlanta, Georgia, pp. 758–764 (2013)
14. Steinberger, R., Eisele, A., Klocek, S., Pilos, S., Schlüter, P.: DGT-TM: a freely available translation memory in 22 languages. In: LREC, pp. 454–459 (2012)
15. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al.: Moses: open source toolkit for statistical machine translation. In: Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, pp. 177–180. Association for Computational Linguistics (2007)
16. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the ACL, pp. 311–318 (2002)
17. Denkowski, M., Lavie, A.: Meteor universal: language specific translation evaluation for any target language. In: Proceedings of the EACL 2014 Workshop on Statistical Machine Translation (2014)
18. Gupta, R.: Use of language technology to imporve matching and retrieval in translation memory. Ph.D. thesis, University of Wolverhampton (2016)