Anubis - speeding up Computer-Aided Translation

Rafał Jaworski

Adam Mickiewicz University
Poznań, Poland
rjawor@amu.edu.pl

Abstract. In this paper, the idea of Computer-Aided Translation is first introduced and a modern approach to CAT is then presented. Next, we provide a more detailed description of one of the state-of-art CAT systems - memoQ. Then, the author's approach to the idea - the Anubis system - is described and evaluated. While Anubis is comparable to memoQ in terms of the precision of provided translation memory matches, it outperforms memoQ when it comes to the speed of searching the translation memory. In the experiments carried out, Anubis turned out to be over 430% faster than memoQ. The result was achieved thanks to the author's algorithm of minimizing the size of translation memory index, so it can be stored in computer's RAM memory.
The paper is divided in two parts: the first describes the field of CAT, where the Anubis system can be applied. The second gives a detailed description of Anubis itself, proving its potential usefulness.

1 Introduction

In the modern world, we observe an increasing use of AI techniques. Can these techniques prove useful in practical applications? This paper focuses on one specific AI mechanism, associated with the domain of machine translation - Computer Aided Translation (abbreviated CAT). It is designed to faciliate the work of a human translator whose task is to translate a document from one language (called the source language) to another (target language). The machine's task in the process is providing suggestions for translation of each sentence to be translated. Such suggestions are then reviewed by the human translator and are used to produce the final translation. From the technical point of view, the main difficulty of the problem lies in the process of generating suggestions for translation of a sentence. To deal with it, most CAT systems incorporate so called translation memories - databases containing previously translated sentences. More specifically, a translation memory is a set of pairs of sentences, where the first sentence is in the source language and the second - in the target language. During the translation process, given an input sentence in source language, the system looks for a similar one in the translation memory. If such a sentence is found, its translation is returned as a suggestion for translation of the input sentence. If not, the sentence is translated by the human translator and added to the translation memory to enrich it.

2 CAT history

The early beginnings of what is today called Computer-Aided Translation date back to 1980s ([1]) when systems of this class were developed in Japan. Japanese computer companies (Fujitsu, Hitachi, NEC, Sharp, Toshiba) worked on software faciliating the process of translation primarily in the directions: Japanese-English and English-Japanese (though other languages were also taken into consideration). The systems relied on automatic translations which were corrected by human translators in the process of post-editing. Machine translation carried out by the systems was based either on a direct word-to-word transfer or on a very superficial lexicographical analysis. Interestingly, these systems tended to focus on a specific domain of texts. The benefits of this focus included lower costs of lexical resources preparation (due to smaller volume of dictionaries), faster translation (for the same reason) and higher precision of translations. The most popular domains the systems focused on were computer science and information technology.
A notable milestone in the history of computer-aided translation was the creation of the ALPS system in 1981 (as described in [1]). ALPS was the first CAT system designed for personal computers and released to the market. It offered the following functionalities:

- multilingual word-processing
- automatic dictionary
- terminology consultation
- interactive translation
- repetitions extraction

Especially the last feature is worth mentioning as it was a very early concept of a translation memory. All translations carried out by a translator were stored in a so called "repetitions file". While working on a new translation, a translator could compare the new sentence to those in the repetitions file. The process was facilited by automatic search of sentence fragments.
Sadly, ALPS did not turn out to be profitable. Nevertheless, several other systems were developed shortly after. By the end of 1980s, the translators had realised how much they could benefit from using a computer as a translation tool. Not only did the CAT systems provide tools for the translation process, but also faciliated word processing and managament of the work. The class of such systems is now called "Translation workstations". The earliest vendors of translation workstations, as stated in [1], were:

- Trados (Translator's Workbench, still developed)
- STAR AG (Transit)
- IBM (the TranslationManager, no longer marketed)
- the Eurolang Optimizer (also no longer available).

More recently, in the 1990s and 2000s many more appeared:

- Atril (Déjà Vu)

- SDL (the SDLX system)
- Xerox (XMS)
- Terminotix (LogiTerm)
- MultiCorpora (MultiTrans)
- Champollion (WordFast)
- MetaTexis
- ProMemoria
- Kilgray Translation Technologies (memoQ system)

Nowadays CAT tools are widely popular among freelance translators and translation agencies.

3 Key concepts

The field of Computer-Aided Translation incorporates a variety of concepts and techniques. It is difficult to find standards regarding the design and functionalites of CAT systems. Nevertheless, it is important to distinguish CAT from other related fields, such as machine translation (MT), natural language processing (NLP) or more general field of artificial intelligence.
This section lists key concepts and definitions that clarify what should and what should not be called CAT. The definitions are inspired by those found in [2].

Computer-Aided Translation (also called Computer-Assisted Translation) is a term used to describe computer techniques used to faciliate the process of translation.

Machine Assisted Human Translation (MAHT) in CAT is the work of a human translator on the translation process. The human translator is the performer of translations while the computer plays a supportive role. This relation is a crucial characteristic of MAHT. There exists an approach where these roles are reversed - Human Assisted Machine Translation (HAMT) in which the human helps the computer in carrying out the translations. This, however, is closely related to machine translation, and is not a part of CAT.

Machine Translation (MT) is an action of fully automatic text translation. The translation is carried out entirely by the computer with no help of human translators whatsoever. Even though MT is not a proper part of CAT, MT systems are sometimes used in CAT systems to provide rough suggestions of translation. Human translator is then responsible for carrying out the post-editing. Such a hybrid technique can be regarded as a CAT technique.

Translation Workbench also known as MAHT Workbench or Integrated Translation System is a piece of computer software offering a variety of CAT techniques along with utilities faciliating the work with text in general.

Translation Memory (TM) is a database of previously carried out translations. It is assumed that TM contains only high-quality translations which can be reused in future. The reuse of translations by means of Translation Memory is the most widely recognized and appreciated feature of CAT. It reflects the natural work process of a translator before the era of computerization, when instead of using databases, translators took notes of phrases and sentences to use them later in their work. Apart from cost saving (once translated sentence does not need to be translated again), Translation Memories grant the consistency of translations, which in some cases is essential.

Terminology consultation is a mechanism of automated dictionary lookups during text translation. It is a widely popular CAT mechanism applied in a majority of CAT systems. During translation of a sentence, a translator is provided with dictionary matches of words or phrases that appeared in the sentence. Typically, multiple dictionaries are searched for terminology matches. These dictionaries are divided into two categories: built-in dictionaries and user-created glossaries. The first are usually vast and comprehensive general dictionaries while the latter serve for storing more specialistic terms. User-created glossaries have similar effect as translation memories - they allow to reduce the cost of translation as well as to ensure consistency. The dictionaries and glossaries are most useful when the translation memory fails to provide a good suggestion.

Text aligning is a process of creating translation memories out of previously translated documents. Professional translators who do not use a CAT system often store their translations in the form of pairs of documents (either in electronic form or on paper). In order to use these translations as a translation memory, the documents need to be aligned. The first stage of the text alignment procedure is importing two monolingual documents into the computer and splitting them into sentences. This process can be automated as automatic sentence splitting procedures usually prove effective. The next step is sentence alignment, i.e. determining, which sentences are each other's translation. There are automatic procedures to perform this task but their results often need correction. Therefore, computer software faciliating manual alignment correction has been developed and included in some CAT systems.

As the latter sections will be dealing with details regarding the new CAT algorithm, some concepts from the field of computer science and natural language processing will also be defined in this section.

Hashing (in computer science) is a procedure carried out on a larger set of data in order to produce its shorter version. This short, simple dataset is called a hash. Hash is typically used to identify the original data.

Lemmatization (in natural language processing) is a process of substituting a word with its base dictionary form, called lemma. Lemmatization must be done

with the use of a dictionary.

Stemming (in natural language processing) is a process of substituting a word with its shorter version, called stem. Stem does not necessarily has to be a dictionary word. Stemming is often done without the use of a dictionary, by simply removing word's flexion suffixes.

4 CAT usability studies

4.1 CAT on the market

In recent years, CAT tools have revolutionized the market of translations (see for instance [3] or [4]). As they allow for reducing the cost of translation, translation agencies using CAT can offer their services at lower prices. Moreover, consistency of translations granted by CAT became not only a desired but demanded quality. As a result, CAT allows the agencies to offer better translations for lower prices and be far more competitive than companies preferring the traditional work model.
However, many translators are still resentful towards CAT. Some agencies need to enforce the usage of CAT tools by obligating the translators to that. Similarily, it is not true that all the freelance translators use CAT. They argue that CAT can generate additional work as post-editing of a sentence with simultaneous terminology consultation and possible other MAHT procedures can take longer than translating this sentence "from scratch" without any help. Where does the truth lie? How helpful are CAT systems on average?
Several studies have been made to find answers to these questions. The popularity of CAT tools and the fact that agencies using them are more competitive on the market suggests that CAT indeed faciliates the process of translation.

4.2 Individual CAT productivity study

As for freelance translators, results and conclusions of the analysis described in [5] can provide answers to the questions posted above. The author of [5] is a professional translator who had been working in this profession before CAT tools became widely popular. During the CAT revolution she became familiar with several tools, as different clients demanded the use of different systems. The author agrees that using CAT tools increases productivity but decided to measure the profits precisely. This is an essential calculation as the majority of clients demand prices reduction for translations with translation memories. It must be known weather the profit from using CAT tools can make up for the reduction of translation prices.
The author of the article was using three different CAT tools: DejaVu (version X), Trados (version 6) and a custom made tool developed by one of her clients.

In DejaVu and Trados she was using her own translation memories, accumulated over the years, totalling approximately 150 000 entries each. The jobs carried out by the author were split into two categories: full and reduce priced. In the first case the client did not demand price reduction for using a translation memory and the translator was using her own resources. In the second case, the client provided the translator with a specialized translation memory, required its use and demanded price reduction for sentences found in the memory. Consequently, with respect to the CAT tool used and price reduction policy, the jobs were split into the following categories:

1. Trados full price
2. Trados reduced price
3. DejaVu full price
4. DejaVu reduced price
5. Custom tool full price
6. Custom tool reduced price
7. No CAT used

The amount of translation work taken into consideration in this study is shown in Table 1. Translation productivity was measured in the unit of words per hour.

Table 1. The amount of work in productivity study.

CAT tool	**No of projects**	**Total word count**	**Total time (h)**
Trados	36	158940	192.8
DejaVu	25	42525	74.35
Custom CAT tool	26	155023	271.2
No CAT tool	3	2617	6.5

The baseline speed of translation with no CAT tool used was **250** words/hour. The calculated productivity for each translation tools is shown in Table 2. The

Table 2. Translation productivity for different CAT tools.

CAT tool	**Productivity (words/hour)**
Trados (full and reduced)	824.3
Trados (full price)	424.5
Trados (reduced price)	1104.3
DejaVu (full and reduced)	571.9
Custom CAT tool (only reduced)	571.6
No CAT tool	250

results are very impressive and show a considerable profit from using CAT tools. It is understandable that the productivity is higher when the client provides a specialized translation memory. The quality of a translation memory is equally important as the power of a CAT tool. Specialized TMs, dedicated for a specific translation task, have a good chance of providing good translation suggestions. In the light of these facts, the client's demand for lower price is justified.
The study also showed that using a CAT tool with a translation memory consisting of translations accumulating over time has an advantage over translating with no help of such a tool.
What follows from this study is that Computer-Aided Translation is a practical, useful technique, worth focusing on.

5 The memoQ CAT system

In this section we describe the memoQ system [6] by Kilgray Translation Technologies. Its translation memory searching module will serve as a baseline for the evaluation of Anubis system.

5.1 Overview

MemoQ is a modern Computer-Aided Translation tool which can be considered as a full translator's workbench. The system was first introduced in 2006. Thanks to its robustness and customizability, the system has gained much popularity and is still in the process of development. Its features include:

- Translation memory
- Terminology base
- Automatic quality checks (based on TM, term bases, spellcheckers and many others)
- Translation editor interface
- Translation memory editor
- Formatting tags analysis
- Text aligner
- Real-time preview for .doc, .docx, .ppt, .pptx, .html and XML file formats
- Handling TMX, XLIFF, bilingual DOC/RTF document formats
- Compatibility with Trados, WordFast and STAR Transit document workflows
- Customizable add-in architecture

Out of the above features, one of the most important factors that build the strength of memoQ is the integration with other popular CAT tools. It faciliates the transition from these tools to memoQ and thus helps to acquire former Trados or WordFast users.

5.2 The memoQ translation memory

Just as in the majority of CAT tools on the market, the key functionality of the memoQ system is a translation memory. It is searched for so called "100% matches" (sentences in the TM identical to the one being translated) as well as for "fuzzy matches" (TM sentences similar to the translated sentence in terms of a fuzzy similarity measure). The system provides a percentage similarity score along with every TM match. The user can set a similarity threshold for TM matches. The matches with scores below the threshold will not be returned as suggestions for translation.
A distinguishing feature of memoQ translation memory are so called "101% matches". The artificial score of 101% is assigned to 100% matches which are found in the same context in TM and in the translated document. For those matches the translator is guaranteed that they require the minimal amount of reviewing. This context matching is referred to as "ICE matching".
Another interesting technique used in memoQ is "SPICE matching". It provides another possibility for a TM match to achieve 101% score. SPICE matching is executed when the translated document is in the XML format. The "over-perfect" score is given to 100% matching segments that refer to the same information in terms of XML attributes and their values. This situation is common e.g. in localisation documents.
The memoQ translation memory has, however, a drawback - it is not well optimized for search speed. This is mainly due to the fuzzy sentence similarity measure which is not the fastest known algorithm for sentence searching. In practice, CAT technique's speed is one of the key factors that decide if it is usable. A translator has to receive the translation suggestions and perform post-editing within the time he or she would have translated the sentence manually. Hence, the speed of sentence searching is crucial, especially when having to deal with translation memories of a considerable size.

6 Anubis sentence search algorithm

In order to speed up searching for sentences in a translation memory, we developed the Anubis system. Its goal is to find in a TM all sentences similar to a given sentence in the shortest possible time. As in other TM search systems, each TM match is assigned a percentage similarity score.
When designing the new algorithm, we decided to take advantage of the achievements in the fields of natural language processing and approximate string matching. This section describes the search algorithm, revealing its main idea - the use of custom designed, compact index.

6.1 The Anubis system

The algorithm was implemented in the Anubis system. Anubis is used to manage a translation memory for the use of Computer-Aided Translation and to perform

TM searches. Technically, Anubis is based on the described suffix-array-based index stored in RAM memory. The system carries out the following operations:

- Adding a sentence pair (example) to the RAM-based index.
- Storing the RAM index to the hard disk.
- Restoring the RAM index from the hard disk.
- Searching the index for examples whose source sentence is similar to a given input sentence.

The typical use case of the system is divided into three stages: preparation, system initialization and translation. The preparation consists of the following steps:

1. Building the index from previously collected translation memory (stored for example in the TMX format).
2. Storing the index into hard disk.

Once the translation memory is stored on the hard disk, the system is able to load it on demand. Loading of the translation memory takes place during the initialization of the Anubis system. The initialization is done in the following steps:

1. Restore the index from hard disk.
2. Initialize system's NLP tools (such as the stemmer, mentioned in the Section 2.1).

Because of the possibility of sharing resources (such as translation memory, stemmer, etc.), Anubis is especially efficient when designed in a client-server architecture. In such architecture, Anubis enables multiple clients to search for sentences in the translation memory as well as add new sentences to it. The typical translation stage (run on a client by a human translator) would be:

1. Get translation suggestions for current sentence.
2. Carry out the translation with the help of suggestions.
3. Add the newly translated sentence to the translation memory for future reuse.
4. Repeat steps 1-3 until the end of the document.

The following subsections will give a detailed description of the procedures that the translation memory search algorithm consists of.

6.2 Sentence hash

The procedure **hash** is invoked on a sentence and returns its hashed version. The first step of the procedure is removal of all punctuation and technical characters from the stentence. In the next step, stop words (such as functional words) are discarded using predefined stop words list. Finally, all the sentence words are stemmed (by the means of a stemmer dedicated for the sentence language). After these operations, a sequence of word stems (tokens) containing only the most significant information is obtained.

6.3 Suffix array

A suffix array is a data structure widely used in the area of approximate string matching (see [7] and [8]). For a given set of strings (or sentences) suffix array is an index, consisting of all suffixes that can be obtained from the strings. For example, the following sentences:

1. Operation is finished
2. Sun is shining
3. I am tall

would produce the following suffixes:

- operation, is, finished
- is, finished
- finished
- sun, is, shining
- is, shining
- shining
- i, am, tall
- am, tall
- tall

The suffixes are then sorted in lexicographical order and put into a so called suffix array, along with the id of the sentence the suffix originates from and the offset of the suffix. The resulting suffix array for the above example is shown in Table 3. The algorithm described in this paper uses its own method of coding

Table 3. Example suffix array.

Suffix	**Sentence id**	**Offset**
am, tall	3	1
finished	1	2
i, am, tall	3	0
is, finished	1	1
is, shining	2	1
operation, is, finished	1	0
shining	2	2
sun, is, shining	2	0
tall	3	2

such a suffix array in order to reduce memory usage. The method uses the general idea of Huffman's coding algorithm (described in [9]). It assignes numeric codes to each sentence and maintaines a code-token dictionary. This technique proves to be effective, as the ratio of distinct tokens divided by total number

of tokens in a corpus is often small (see [10] for ratio of 0.033). This ratio gets even smaller when each token is stemmed before processing. In an experimental corpus of 3 593 227 words, the number of distinct stemmed tokens was 17 001, resulting in a ratio of 0.005. Because integers are less memory consuming than strings representing the tokens and the dictionary takes up a relatively small amount of memory, this method leads to a significant memory usage decrease.

6.4 Adding to the index

Another procedure that has to be carried out before searching is adding a sentence to the index. Procedure **indexAdd** takes two parameters: the sentence and its unique id. It takes advantage of a simple hash-based **dictionary**, capable of storing, retrieving and creating codes for tokens. The index is represented by the object **array**.
The procedure first generates the hashed version of the sentence. Then, every token from the hash is substituted with a code. Finally, the procedure generates all the suffixes from the modified hash and adds them to the index with the sentence's id and offset parameters. These additions preserves the suffix array sorting. The procedure is described in Figure 1.

Algorithm 1: adding to the index

```
procedure indexAdd(s,id)
    h := hash(s)
    for all (Token t in h)
        code := dictionary.get(t)
        if (code == null)
            code := dictionary.createNewCode(t)
        t := code  //substitute a token with its code
    for (i = 0 to length(h))
        array.addSuffix(h.subsequence(i,length(h)),id, i)
end procedure
```

Fig. 1. Algorithm for adding a single sentence to the index

6.5 Searching the index

The algorithm for searching the index uses a procedure called **getLongestCommonPrefixes** and an object **OverlayMatch**. Both of them will be defined in this section before introducing the main search algorithm.
The procedure **getLongestCommonPrefixes**, described in Figure 2, takes one parameter - a series of tokens - and returns a set of suffixes from the array having

the longest common prefix with the input series. It takes advantage of the array's method **subArray** which returns the set of suffixes from the array beginning with a given series of tokens. The method subArray is optimized (by means of the binary search).

Algorithm 2: getLongestCommonPrefixes procedure

```
procedure getLongestCommonPrefixes(h)
    longestPrefixesSet := empty set
    pos := 0
    currentScope := array
    while(not empty(currentScope) and pos < length(h))
        currentScope := currentScope.subArray(h.subSequence(0,pos))
        if (size(currentScope) > 0)
            longestPrefixesSet := currentScope
        pos := pos + 1
    return longestPrefixesSet
end procedure
```

Fig. 2. The getLongestCommonPrefixes procedure

The **OverlayMatch** object holds information about the degree of similarity between the searched sentence and one of the sentences in the suffix-array-based index. Each sentence candidate found in the index has its own OverlayMatch object which is used to assess its similarity to the searched sentence. The object is described in Figure 3. For example, if the searched sentence (pattern)

The OverlayMatch object definition

```
object OverlayMatch {
    patternMatches - a list of disjunctive intervals,
                     representing the overlay of the searched sentence
    exampleMatches - a list of disjunctive intervals,
                     representing the overlay of the candidate sentence
}
```

Fig. 3. The OverlayMatch object

is: "There is a small book on the table" and the candidate sentence (example) is: "I know there is a small pen on the desk", the OverlayMatch object would be:

patternMatches : { [0,3]; [5,6] }
exampleMatches : { [2,5]; [7,8] }

The main search algorithm - the procedure **search** - takes one parameter: a series of tokens and returns a map of candidate sentence ids and their OverlayMatch objects. It is described in Figure 4.

Algorithm 3: The main search procedure

```
procedure search(h)
    for(i = 0 to length(h))
        longestPrefixes := getLongestPrefixes(h.subSequence(i,length(h)))
        for all (Suffix suffix in longestPrefixes)
            prefixLength := longestPrefixes.getPrefixLength()
            currentMatch := matchesMap.get(suffix.id)
            currentMatch.addExampleMatch(suffix.offset, suffix.offset+prefixLength)
            currentMatch.addPatternMatch(i, i+prefixLength)
end procedure
```

Fig. 4. The search procedure

6.6 Computing score

For a given input sentence (pattern) and a candidate sentence (example) having an OverlayMatch object, the similarity score is computed using the following formula:

$$score = \frac{\sum_{i=0}^{n} patternMatches[i].length + \sum_{i=0}^{m} exampleMatches[i].length}{length(pattern) + length(example)}$$

where:

- $patternMatches[k].length$ is the length (in tokens) of k-th interval
- $length(pattern)$ is the length of the pattern

For the example used to present the OverlayMatch object in section 2.4, the computed score would be:

$\frac{(4+2)+(4+2)}{8+10} \approx 66.6\%$

7 Evaluation

Anubis was evaluated in order to measure the precision of its translation suggestions and the speed of TM searching. For comparison, the system memoQ, described in Section 5 was selected.

7.1 Precision evaluation

Precision evaluation procedure was aimed at determining wheather Anubis is able to provide as valuable translation suggestions as memoQ. As there is no good way of scoring the value of suggestions automatically, human translators were involved in the evaluation process.
The experiment was carried out on a genuine translation memory, coming from a translator who built it during his work. Using this kind of TM makes this evaluation significantly more credible than that using bilingual corpora acquired from the Internet. These corpora are often the results of text alignment, not sentence-by-sentence translation. For that reason, they differ from real-life translation memories (e.g. bilingual corpora tend to contain longer sentences).
The statistics of the translation memory used in the experiment are presented in Table 4.
The evaluation procedure was split into three phases: preparation, translation

Table 4. Experiment's translation memory statistics.

Attribute	**Value**
Source language	Polish
Target language	English
Translation unit count	215 904
Polish word count	3 194 713
English word count	3 571 598
Total word count	6 766 311

and annotation. The preparation phase involved selecting at random 1500 Polish sentences from the translation memory (let's denote it as set $TEST$). During the translation phase, Anubis and memoQ provided translation suggestions for every sentence from $TEST$ using the procedure described in Figure 5.
In this procedure, each sentence is translated using the translation memory it came from, so it is expected to appear as one of possibly many suggestions. If a translation system returns an empty set of suggestions, it will signal the pathological situation in which a 100% match is not found by the system. If the returned *suggestions* set contains only one element, the perfect match, it will be intepreted as the situation where the system could not find a good match. Finally, if the *suggestions* set contains more than one suggestion, they are (except for the first one, which is the 100% match) the proper suggestions. In the report only the first proper suggestion is included along with its similarity score. The translation procedure was run both by Anubis and memoQ. The results of these runs are shown in Table 5. "Reported errors" correspond to the number of error reports generated by the Translation procedure described in Figure 5. Both systems did not commit any error and successfully detected all the 100% matches.

Translation procedure

```
for all sentence in TEST
    suggestions = getTranslationSuggestions(sentence)
    if (size(suggestions) == 0)
        report('error!')
    else if (size(suggestions) == 1)
        report('no translation found')
    else
        suggestions.remove(0)
    print(sentence,suggestions[0])
```

Fig. 5. Translation procedure

Table 5. Translation phase statistics.

	memoQ	**Anubis**
Sentences analyzed	1500	1500
Reported errors	0	0
Translated sentences	1156	962
Knock-outs	285	91
Awarded better scores	491	374
Common translations	871	
including:		
- identical translations	444	
- different translations	427	
Scores correlation (Pearson)	0.541	
Scores correlation (Spearman)	0.553	

"Translated sentences" count indicates the number of sentences for which at least one proper suggestion has been found (i.e. the *suggestions* set contained more than 1 element). The results were comparable, though memoQ translated 20% more sentences. This was due to the fact that Anubis had a 50% similarity threshold set, whereas memoQ did not.

A situation in which system A translated a sentence and system B did not translate this sentence was called system's A "knock-out". In the light of the previous figure, indicating the total number of translations, it is not surprising that memoQ scored more "knock-outs" than Anubis.

These "knock-outs", however, were further analyzed to determine if the translation suggestions corresponding to them were valuable. This analysis was performed by two human translators, who were given a set of sentence pairs. Each pair constituted of the source sentence and the suggestion provided by a CAT system. For each pair a translator was to answer a yes/no question: would the provided suggestion be helpful in translating the source sentence into the target language? This procedure was the first step of the experiment's annotation

phase. The results of the "knock-outs" analysis are shown in Table 6. This study

Table 6. "Knock-out" statistics.

	Annotator 1	**Annotator 2**
Total memoQ knock-outs	285	
Valuable memoQ knock-outs	162	246
Total Anubis knock-outs	91	
Valuable Anubis knock-outs	34	45

indicates that the similarity threshold for Anubis can be set to a lower value.
In the translation phase, the scores of suggestions provided by the two systems were also taken into consideration. The correlation of these scores (treated as random variables) was measured using statistical methods (see [11]). The coefficient values around 0.5 indicate some correlation between the scores. MemoQ was the system that was awarding better scores in general for the same suggestions.
The two systems provided identical translation suggestions for 444 out of total 871 common translations. The remaining 427 translations were judged by two human translators in the key step of the annotation phase. The translators were given a set of 3-tuples: source sentence, translation from system A, translation from system B. In order to obtain more balanced results, the order of translations was changed randomly. For each tuple, the annotators were to assign one of the possible scores

- first system's suggestion is better
- second system's suggestion is better
- both translations are of equal value

The criterion of the suggestion's value was: "how helpful is the suggestion in translating the source sentence into the target language?"
The results of this annotation are shown in Table 7. The inter-annotator agree-

Table 7. Suggestion precision statistics.

	Annotator 1	**Annotator 2**
Total translations	427	
Anubis wins	**156**	**103**
MemoQ wins	150	96
Draws	121	228

ment was: the annotators agreed in 262 cases, disagreed 165 times out of which

only 24 were strong disagreements (one annotator assigned a win to one system and the other annotator to the other system).
These results show that suggestions generated by the two systems are roughly comparable.

7.2 Speed evaluation compared to memoQ

Apart from the precision, the most important characteristic of a CAT system is its speed. During the translation phase of the above experiment, the translation speed was measured. The tests were run on a machine with Intel Core 2 Duo 2.0 GHz CPU and 3GB RAM memory. The results of the speed test are shown in Table 8. These results show a considerable advantage of Anubis. The speed

Table 8. MemoQ speed test results.

	memoQ	**Anubis**
Total translations	1500	
Translation time [s]	414.6	**94.7**
Average speed [sent/s]	3.618	**15.846**

of suggestion generation can be a crucial characteristic of a CAT system in the context of translation memory servers. As the volume of collected translation memories grows, translation agencies try to realize an idea of centered TM, shared among their translators. On the other hand, freelance translators often team up in communities and share their translation resources (an example of such a project is Wordfast VLTM - Very Large Translation Memory [12]).
These tendencies lead to creation of vast translation memories, shared among multiple users. In this architecture, it is essential to have an effective translation memory search algorithm. The algorithm will have to be able to deal with a large data set in a short time, as multiple users working simultaneously will query the TM often. Thus, the idea of speeding up translation memory searching is a key idea for next generation CAT tools.

8 Conclusions

This paper presented the idea od Computer-Aided Translation, which is gaining more and more popularity in today's market of translations. The technique is known to improve the efficiency of human translator's work. A good CAT system provides precise translation suggestions based on the translation memory. The other key factor that determines the system's usability is the speed of suggestions generation. Although nowadays this CAT technique is often wrapped up in a whole translator's workbench (like in the memoQ system), translation

memory searching is still the key mechanism that is used to help the translator. A novel approach to translation memory building, storing and searching was presented in this paper. We propose the system Anubis, based on state-of-art techniques of approximate string searching and natural language processing. The results of the first tests show that the precision of translation suggestions generated by Anubis can be compared to the memoQ system. However, Anubis offers much higher speed of translations, especially for large translation memories which are the future of CAT.
The system Anubis is still in the process of development. The study on Computer-Aided Translation along with the system's evaluation indicate that Anubis has a chance to become a usable, helpful tool for translators.

References

1. Hutchins, J.: Machine translation: a concise history. Computer aided translation: Theory and practice, ed. Chan Sin Wai. Chinese University of Hong Kong (2007)
2. Palacz, B.: A comparative study of cat tools (maht workbenches) with translation memory components. Master thesis written under guidance of prof. Włodzimierz Sobkowiak, Adam Mickiewicz University (2003)
3. Twiss, G.: A comparative study of cat tools (maht workbenches) with translation memory components. proz.com The translator workspace (2006)
4. Craciunescu, O., Gerding-Salas, C., Stringer-O'Keeffe, S.: Machine translation and computer-assisted translation: a new way of translating? The Translation Journal Volume: 8 Issue: 3 (2004)
5. Vallianatou, F.: Cat tools and productivity: Tracking words and hours. The Translation Journal Volume: 9 Issue: 4 (2005)
6. multiple: Kilgray translation technologies: memoq translator pro. (http://kilgray.com/products/memoq/)
7. Navarro, G., Baeza-yates, R., Sutinen, E., Tarhio, J.: Indexing methods for approximate string matching. IEEE Data Engineering Bulletin **24** (2000) 2001
8. Navarro, G.: A guided tour to approximate string matching. ACM Computing Surveys (CSUR) Volume 33 Issue 1, March 2001 (2001)
9. Huffman, D.: A method for the construction of minimum-redundancy codes. Proceedings of the I.R.E., pp. 1098–1102 (1952)
10. Tufiş, D., Irimia, E.: Roco-news: A hand validated journalistic corpus of romanian (2006)
11. Stigler, S.M.: Francis galton's account of the invention of correlation. Statistical Science (1989)
12. multiple: Wordfast community: Very large translation memory project. (http://www.wordfast.com/)