Efficiency-based evaluation of aligners for industrial applications*

Antonio Toral
School of Computing
Dublin City University
Dublin, Ireland
atoral@computing.dcu.ie

Marc Poch
IULA, Universitat
Pompeu Fabra
Barcelona, Spain
marc.pochriera@upf.edu

Pavel Pecina
Faculty of Mathematics and
Physics, Charles University
Prague, Czech Republic
pecina@ufal.mff.cuni.cz

Gregor Thurmair
Linguatec GmbH
Munich, Germany
g.thurmair@linguatec.de

Abstract

This paper presents a novel efficiency-based evaluation of sentence and word aligners. This assessment is critical in order to make a reliable use in industrial scenarios. The evaluation shows that the resources required by aligners differ rather broadly. Subsequently, we establish limitation mechanisms on a set of aligners deployed as web services. These results, paired with the quality expected from the aligners, allow providers to choose the most appropriate aligner according to the task at hand.

1 Introduction

Aligners refer in this paper to tools that, given a bilingual corpus, identify corresponding pairs of linguistic items, be they sentences (sentence aligners) or words (word aligners). Alignment is a key component in corpus-based multilingual applications. First, alignment is one of the most time-consuming tasks in building Machine Translation (MT) systems. In terms of quality, good alignment is decisive for the final quality of the MT system; bad alignment decreases MT quality and inflates the phrase table with spurious translations with very low probabilities, which reduces system performance. Finally, for terminology acquisition, the choice of a good aligner determines whether the results of a term extraction tool are usable or not; alignment quality on phrase level differs from less than 5% (usable) to more than 40% (unusable) error rate (Aleksic and Thurmair, 2012).

The performance of aligners is commonly evaluated extrinsically, i.e. by measuring their impact in the result obtained by a MT system that uses the aligned corpus (Abdul-Rauf et al., 2010; Lardilleux and Lepage, 2009; Haghighi et al., 2009). Intrinsic evaluations have also been carried out, mainly by measuring the Alignment Error Rate (AER), precision and recall (von Waldenfels, 2006; Varga et al., 2005; Moore, 2002; Haghighi et al., 2009). Intrinsic evaluation is less popular due to two reasons (Fraser and Marcu, 2007): (i) it requires a gold standard and (ii) the correlation between AER and MT quality is very low. Both types of evaluation have, however, a common aspect; they focus on measuring the quality of the output produced by aligners. Conversely, seldom if at all has it been considered to assess the efficiency of aligners, i.e. to measure the computational resources consumed (e.g. execution time, use of memory). However, this assessment is critical if the aligners are to be exploited in an industrial scenario.

This work is part of a wider project, whose objective is to automate the stages involved in the acquisition, production, updating and maintenance of language resources required by MT systems. This is done by creating a platform, designed as a dedicated workflow manager, for the composition of a number of processes for the production of language resources, based on combinations of different web services.

The present work builds upon (Toral et al., 2011), where we presented a web service architecture for sentence and word alignment. Here we extend this proposal by evaluating the efficiency of the aligners integrated, and subsequently im-

*We would like to thank Daniel Varga and Adrien Lardilleux for their feedback on Hunalign and Anymalign, respectively. We would like to thank Joachim Wagner for his help on using the cluster. This research has been partially funded by the EU project PANACEA (7FP-ITC-248064).

proving the architecture by implementing limitation mechanisms that take into account the results.

2 Evaluation

We have integrated a range of state-of-the-art sentence and word aligners into the web service architecture. The sentence aligners included are Hunalign (Varga et al., 2005), GMA[1] and BSA (Moore, 2002). As for word aligners, they are GIZA++ (Och and Ney, 2003), BerkeleyAligner (Haghighi et al., 2009) and Anymalign (Lardilleux and Lepage, 2009). For a detailed description of the integration please refer to (Toral et al., 2011).

In order to evaluate the efficiency of the aligners, we have run them over different amounts of sentences of a bilingual corpus (from 5k to 100k adding 5k at a time for sentence alignment and from 100k to 1.7M adding 100k at a time for word alignment). For all the experiments we use sentences from the Europarl English–Spanish corpus,[2] which contains over 1.7M sentence pairs. The aligners are executed using the default values for their parameters. All the experiments have been run in a cluster node with 2 Intel Xeon X5670 6-core CPUs and 96 GB of RAM. The OS is GNU/Linux. The resources consumed have been measured using the following parameters of the GNU command `time`:

- `%S` (CPU-seconds used by the system on behalf of the process) plus `%T` (CPU-seconds that the process used directly), to measure the execution time. We limit our experiments to 100k seconds.
- `%M` (maximum resident set size of the process during its lifetime, in Kilobytes), to measure the memory used.

Figure 1 shows the execution times (logarithmic scale) of the sentence aligners. It emerges that the time required by GMA is considerable higher compared to the other two aligners (e.g., for 45k sentences GMA takes approximately 16 and 20 times longer than BSA and Hunalign, respectively). The gap grows exponentially with the input size.

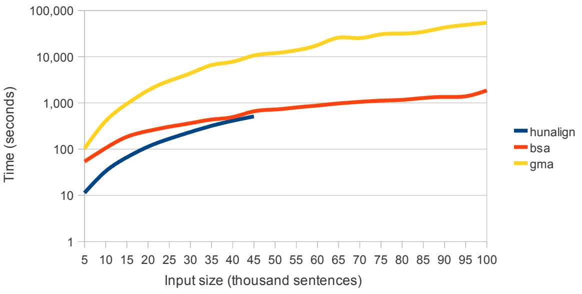

Figure 1: Execution time for sentence aligners

Figure 2 shows the memory consumed by the sentence aligners. Hunalign has a steeper curve (for 45k sentences, Hunalign uses 6 and 4 times more memory than BSA and GMA, respectively). In fact Hunalign was not able to align inputs of more than 45k sentences due to memory issues.[3] Table 1 contains all the measurements for sentence alignment.

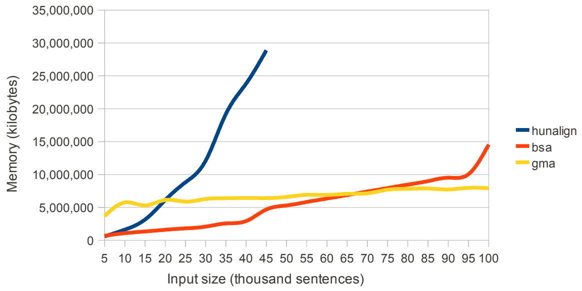

Figure 2: Memory used by sentence aligners

	Time (seconds)			Memory (M bytes)		
i	hun	bsa	gma	hun	bsa	gma
5	11	54	103	584	684	3,677
10	33	105	405	1,616	1,079	5,749
15	66	185	950	3,146	1,337	5,305
20	113	247	1,866	6,115	1,597	6,126
25	168	305	3,004	8,803	1,807	5,878
30	234	364	4,370	12,104	2,070	6,276
35	319	436	6,578	19,211	2,559	6,390
40	412	494	7,775	23,827	2,919	6,433
45	510	659	10,609	28,892	4,679	6,415
50	-	721	11,947	-	5,297	6,594
55	-	797	13,768	-	5,824	6,915
60	-	878	17,780	-	6,347	6,888
65	-	973	25,787	-	6,872	7,061
70	-	1,053	25,251	-	7,415	7,143
75	-	1,120	30,513	-	7,940	7,692
80	-	1,165	31,591	-	8,469	7,832
85	-	1,277	34,664	-	8,991	7,872
90	-	1,348	42,720	-	9,518	7,730
95	-	1,391	48,823	-	10,043	7,969
100	-	1,863	54,350	-	14,537	7,911

Table 1: Detailed results for sentence aligners. **i** input sentences (thousand), **hun** hunalign

Figure 3 shows the execution times for word aligners. GIZA++ is the most efficient word aligner, consistently across the different inputs.

[1] `http://nlp.cs.nyu.edu/GMA/`

[2] `http://www.statmt.org/europarl/`

[3] A constant in the source code of Hunalign establishes the maximum amount of memory it will use, by default 4GB; we increased it to 64GB. Moreover, it can split the input into smaller chunks with partialAlign (it cuts the data into chunks of approximately 5,000 sentences each, based on hapax clues found on each side), however we did not use this preprocessing tool but only the aligner itself.

The performance of Berkeley is similar to that of GIZA++ for the first runs but the difference of execution time grows with the size of the input. There are no results for Berkeley for over 1,1M sentences as the time limit is exceeded. Finally, the behaviour of Anymalign does not correlate at all with the size of the input. This has to do with the very nature of this aligner.[4]

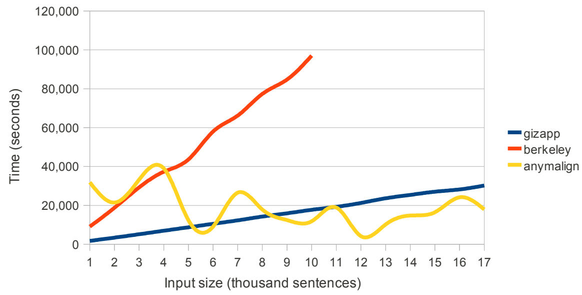

Figure 3: Execution time for word aligners

Figure 4 shows the memory required by word aligners. Berkeley consistently requires more memory than both GIZA++ and Anymalign. The requirements of GIZA++ and Anymalign are similar, although slightly lower for the latter. Table 2 contains all the measurements for word alignment.

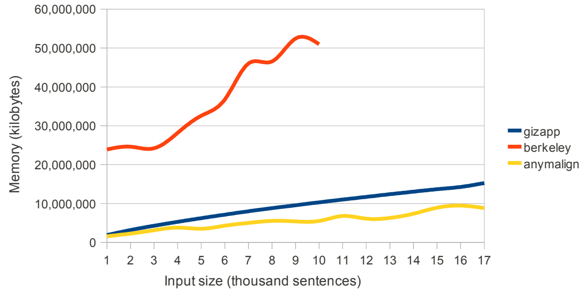

Figure 4: Memory used by word aligners

3 Limiting web services

The previous section has shown that the computational resources required by state-of-the-art aligners are very different. These resources are limited and must be taken into account when they are being shared by users using web services.

We have studied ways on establishing limitations for the aligners deployed as web services. Two kinds of limitations are explored and implemented: (i) the number of concurrent executions and (ii) the input size allowed for each aligner.

The web services are developed using Soaplab2.[5] This tool allows to deploy web services on top of command-line applications by writing files that describe the parameters of these services in ACD format.[6] Soaplab2 then converts the ACD files to XML metadata files which contain all the necessary information to provide the services. The Soaplab server is a web application run by a server container (Apache Tomcat[7] in our setup) which is in charge of providing the services using the generated metadata.

	Time (k seconds)			Memory (M bytes)		
i	giz	brk	any	giz	brk	any
1	1.7	9,0	31,9	1,894	23,906	1,582
2	3.4	18,8	21,4	3,181	24,619	2,277
3	5.1	29,2	33,2	4,293	24,222	3,142
4	6,9	37,3	39,0	5,292	28,190	3,818
5	8,7	43,6	12,4	6,245	32,586	3,525
6	10,5	58,0	9,0	7,144	36,773	4,304
7	12,3	66,2	26,5	8,008	45,999	5,017
8	14,2	77,3	17,8	8,807	46,545	5,531
9	15,9	84,7	12,4	9,565	52,437	5,407
10	17,7	97,0	11,8	10,313	50,977	5,522
11	19,3	-	18,9	11,030	-	6,800
12	21,2	-	4,1	11,713	-	6,107
13	23,6	-	10,1	12,403	-	6,301
14	25,4	-	14,8	13,057	-	7,382
15	27,0	-	16,5	13,688	-	8,931
16	28,2	-	24,2	14,272	-	9,469
17	30,2	-	17,9	15,270	-	8,860

Table 2: Detailed results for word aligners. **i** input sentences (hundred thousand), **giz** GIZA++, **brk** Berkeley, **any** Anymalign

Figure 5 shows the diagram of the program flow for web services that incorporates limitation mechanisms.[8] The modules are the following:

- *tool.acd* (e.g. `bsa.acd`), contains the metadata of the web service in ACD format.
- `ws.sh`, controls other modules that implement the waiting and execution mechanisms.
- `init_ws.sh`, contains the code that implements the limitation on the number of concurrent executions and waiting queue. The web service is in waiting state while it is executing this script.
- *tool.sh* (e.g. `bsa.sh`), executes the tool. The web service is in executing state while it is executing this script.
- `ws_vars.sh`, contains all the variables used by the different web services.
- `ws_common.sh`, contains code routines shared by different web services.

[4] Anymalign runs are random, its stop criterion can be based on the number of alignments it finds per second, we set this parameter to the most conservative value supported, i.e. 1 alignment per second.

[5] `http://soaplab.sourceforge.net/soaplab2/`

[6] `http://soaplab.sourceforge.net/soaplab2/MetadataGuide.html`

[7] `http://tomcat.apache.org/`

[8] The code is available under the GPL-v3 license at `BLIND`

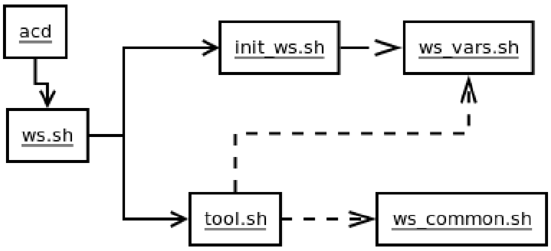

Figure 5: Diagram of the program flow

3.1 Limitation of concurrent executions

The limitation of concurrent executions is controlled by two variables, MAX_WS_WAIT and MAX_WS_EXE, set in ws_vars.sh. They hold the maximum number of web services that can be concurrently waiting and executing, respectively.

The following actions are carried out when a web service is executed. First, *tool.acd* calls ws.sh. This one calls sequentially two scripts: init_ws.sh and *tool.sh*. init_ws.sh checks if the waiting queue is full and aborts the execution if so. Otherwise it puts the execution in waiting state and checks periodically whether the execution queue is full. When there is a free execution slot, init_ws.sh exits returning the control to ws.sh, which changes the state to executing and calls *tool.sh*.

3.2 Limitation of input size

The limitation of input/output data size can be performed at three levels: Tomcat, Soaplab and web service. Tomcat provides a parameter, MaxPostSize, which indicates the maximum size of the POST in bytes that will be processed. Soaplab allows us to put a size limit (in bytes) to the output of web services using a property. The user can establish a general limit that applies to every web service, and/or specific limits that apply to any web service in particular.

Both these methods allow us to limit the input/output of web services in bytes. However, limiting the size according to different metrics might be useful. For example, the inputs of aligners are usually measured in number of sentences (rather than number of bytes). Limits of number of input sentences have been established at the web service level for each aligner following the results obtained in the evaluation (Section 2). Variables with the desired maximum input size in number of sentences have been added for each aligner in ws_vars.sh. A function included in ws_common.sh checks the size of the input whenever an aligner is executed.

4 Conclusions

This paper has presented, to the best of our knowledge, the first efficiency-based evaluation of sentence and word aligners. This assessment is critical in order to make a reliable use in industrial scenarios, especially when they are offered as services. The evaluation has showed that the resources required by aligners differ rather broadly. These results, paired with the quality expected from the aligners, allow providers to choose the most appropriate aligner according to the task at hand.

References

Abdul-Rauf, S., M. Fishel, P. Lambert, S. Noubours, and R. Sennrich. 2010. Evaluation of Sentence Alignment Systems (Project at the Fifth Machine Translation Marathon).

Aleksic, V. and G. Thurmair. 2012. Rule-based MT system adjusted for narrow domain (ACCURAT Deliverable D4.4.). Technical report.

Fraser, A. and D. Marcu. 2007. Measuring Word Alignment Quality for Statistical Machine Translation. *Computational Linguistics*, 33:293–303.

Haghighi, A., J. Blitzer, J. DeNero, and D. Klein. 2009. Better word alignments with supervised ITG models. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 923–931.

Lardilleux, A. and Y. Lepage. 2009. Sampling-based multilingual alignment. In *Proceedings of RANLP*, pages 214–218, Borovets, Bulgaria.

Moore, R. C. 2002. Fast and accurate sentence alignment of bilingual corpora. In *Proceedings of AMTA*, pages 135–144.

Och, F. J. and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29:19–51.

Toral, A., P. Pecina, A. Way, and M. Poch. 2011. Towards a User-Friendly Webservice Architecture for Statistical Machine Translation in the PANACEA project. In *Proceedings of EAMT*, pages 63–72, Leuven, Belgium.

Varga, D., L. Németh, P. Halácsy, A. Kornai, V. Trón, and V. Nagy. 2005. Parallel corpora for medium density languages. In *Proceedings of RANLP*, pages 590–596, Borovets, Bulgaria.

von Waldenfels, R. 2006. Compiling a parallel corpus of slavic languages. Text strategies, tools and the question of lemmatization in alignment. In *Beiträge der Europäischen Slavistischen Linguistik*, pages 123–138.