Fast, Scalable Phrase-Based SMT Decoding

Anonymous ACL submission

Abstract

The utilization of statistical machine translation (SMT) has grown enormously over the last decade, many using open-source software developed by the NLP community. As commercial utilization has increased, there has been a pressing need that is optimized for their requirements. Specifically, faster phrase-based decoding, and more efficient utilization of modern multicore servers.

We present in this paper a re-assessment of the major components of phrase-based decoding and decoder implementation with particular emphasis on speed and scalability to multicore machines. The result is a drop-in replacement for the Moses decoder which is up to fifteen times faster and scales almost linearly with the number of cores. Furthermore, the decoder makes less search errors than the current Moses decoder.

1 Introduction

SMT has been one of the outstanding success story from the NLP community in the last decade, progressing from a mostly research discipline to public useability via services such as Google Translate, Microsoft Translator Hub, as well as services and products built around offline products such as Language Weaver and the open-source Moses toolkit. The latter has spawned a cottage industry encompassing a range of organizations and services from small language service providers seeking to reduce translation cost, to large intergovermental organizations such as the EU and the UN that require high volume, high quality translation.

For high volume users, decoding is a largest and most critical part of the translation process which needs to be fast and efficient. However, it has been noticed that the Moses decoder, amongst others, is unable to efficiently use multiple CPU cores that are now common on modern servers (reviewed paper, github discussion). That is, the time taken to decode a test set does not substantial decrease when more cores are used, in fact, decoding time may increase when more cores are added. The problem will continue to grow as the commercial use of SMT increases and the number of CPU cores increases.

There have be speculation on the causes of the inefficiency as well as potential remedies. This paper is the first we know of that seeks to tackle this problem head on. We present an phrase-based decoder that is not only significantly faster than the Moses baseline for single-threaded operation, but is able to scale run multiple threads on multicore machines with only a slightly loss in linear speed. Model scores and functionality are compatible with Moses to aid comparison and ease of transition for users. All source code will be made available under an open-source license.

1.1 Prior Work

There are a number of open-source SMT projects, most includes a decoder. The most well known is Moses, which supports phrase-based models, hierarchical phrase-based as well as various syntax-based models. Joshua also supports hiearchical and syntax models and has recently supported phrase-based models. Phrasal supports a number of variants of the phrase-based model. CDEC supports hierarchical and syntactic models.

A number of the decoders support multithreading whilst others use alternative methods such as Hadoop or external scripts to parallelize decoding. We shall investigate the efficiency of using parallelizing decoding using the multi-processor approach. None of the decoder focus on multi-threads decoding.

(Recently reviewed) describes running multiple processes of the Moses decoder for increased speed.

Other prior work look to optimizing specific components of decoding. (Liang and Chiang) describes the cube-pruning and cube-growing algorithm for decoding which allows the tradeoff between speed and translation quality to the adjusted with a simgle parameter. (KenLM) and (DALM) describes fast, efficient datastructures for language models. (Zen) describes an implementation of a phrase-table for an SMT decoder that is loaded on demand, reducing the initial loading time and memory requirements. (CompactPT) extends this by compressing the on-disk phrase table and lexicalized re-ordering model resulting in impressive speed gains over previous work.

(mtplz) is perhaps closest in intent to this work. This takes a wholistic approach to decoding, describing a novel decoding algorithm which is fis focused on better decoding speed. It also describes a number of implementation details for faster decoding. However, the decoding algorithm is only able to incorporate one stateful feature function which precludes some of the useful decoding configurations which contains multiple stateful feature functions. It does not include a load-on-demand phrase table, therefore, cannot be used in a commercial environment where phrase-table has not be filtered with a know test set for any realistic size phrase-table. Neither did this paper analyze the scalability of their work to multicore servers.

The rest of the paper will be broken up into the following sections. Next, we will describe the phrase-based model and the major implementation components, with particular emphasis on decoding time shortcomings. We will then describe modifications to improve decoding speed and present results. We conclude in the last section discuss suggested improvements and future work.

2 Phrase-Based Model

The objective of decoding is to find the target translation with the maximum probability, given a source sentence. That is, for a source sentence s, the objective is to find a target translation $\hat{t}$ which has the highest conditional probability $p(t|s)$. Mathematically, this is written as:

$$\hat{t} = \arg\max_t p(t|s) \quad (1)$$

where the *arg max* function is the search. The log-linear model generalizes Equation 1 to include more component models and weighting each model according to the contribution of each model to the total probability.

$$p(t|s) = \frac{1}{Z} \exp(\sum_m \lambda_m h_m(t,s)) \quad (2)$$

where λ_m is the weight, and h_m is the feature function, or 'score', for model m. Z is the partition function which can be ignored for optimization.

2.1 Beam Search

A translation of a source sentence is created by applying a series of translation rules which together translate each source word once, and only once. Each partial translation is called a *hypothesis*, which is created by applying a rule to an existing hypothesis. This process is called *hypothesis expansion* and starts with a hypothesis that has translated no source word and ends with a completed hypothesis that has translated all source words. The highest-scoring completed hypothesis, according to the model score, is returned as most probable translation, $\hat{t}$. Incomplete hypotheses are referred to as partial hypotheses.

Each rule translates a contiguous sequence of source words but successive translation options do not have to be adjacent on the source side, depending on the distortion limit. However, the target output is constructed strictly left-to-right from the target string of successive translation options. Therefore, successive translation options which are not adjacent and monotonic in the source causes translation reordering.

A beam search algorithm is used to create the completed hypothesis set efficiently. Partial hypotheses are organized into stacks where each stack holds a number of comparable hypotheses. Hypotheses in the same stack have the same coverage cardinality $|C|$, where C is the coverage set, $C \subseteq \{1, 2, ...|s|\}$ of the number of source words translated. Therefore, $|s| + 1$ number of stacks are created for the decoding of a sentence s.

There are three main optimization to the search that we shall investigate. Firstly, the search creates and destroy a large number of hypothesis objects in memory which puts a heavy burden on the operating system. We shall optimize the search algorithm to use memory pools and object pools, replacing the operating system's general purpose memory management with our own application-aware management.

In multiprocessor servers, the CPU cache is attached to each processor and each core. If a sentences is being decoded on one CPU is switched to another, the CPU cache on the new CPU must be repopulated, slowing down decoding. We will therefore investigate binding threads to specific cores.

Lastly, we shall investigate different stack configurations other than coverage cardinality to see whether they can improve speed and translation quality.

2.2 Feature Functions

Features functions are the h_m in Equation 2, calculating a score for each hypothesis.

The standard feature functions in the phrase-based model include:

1. log transforms translation model probabilities, $p_{TM}(t|s)$ and $p_{TM}(s|t)$, and word-based translation probabilities $p_w(t|s)$ and $p_w(s|t)$,
2. log transforms of the lexicalized re-ordering probabilities,
3. log transforms of the target language model probability $p(t)$,
4. a distortion penalty
5. a phrase-penalty,
6. a word penalty,
7. an unknown word penalty.

The first three feature functions frequently trained on data and require the feature to read the model from files. The other feature functions do not require model files. We shall investigate the first two feature functions for optimization.

2.3 Translation Model

For any realistic sized phrase-based model to be used in an online situation, memory and loading time constraints requires us to use load-on-demand phrase-table implementations. Moses contains a number of such implementations with different performance characterics, we show the time take to decode 800,000 sentences for the fastest two in Figure 1. From this, it appears

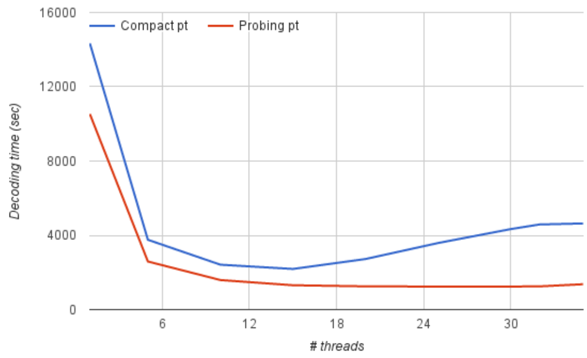

Figure 1: Moses decoding time (in sec) with two different phrase-table implementations

that the Probing phrase-table (Bogochev, 2013) has the fastest translation rule lookup, especially with large number of cores, therefore, we will concentrate exclusively on this implementation in our work as this . We will look at two main areas for phrase-table optimization specifically in relation to the Probing implementation.

We will focus on two main optimizations that differs from the current Probing phrase-table implementation and that have application to other implementations. Firstly, we will look at the **caching** of often used translation rules. The default caching framework save the most recently looked up rules. However, this caching mechanism is actually slower than re-querying the phrase-table, Table **??**. We shall explore a simpler caching mechanism that creates the cache at the start of decoding and remain static thereafter.

Secondly, the Probing phrase-table use a simple **compression** algorithm to compress the target side of the translation rule. Compression was championed being the main factor for the speed of the Compact phrase-table but as we saw in Figure 1, it may come at a cost when used with higher number of cores. We shall therefore take the opposite approach to Junczys-Dowmunt (2012) and explore the decoding speed gain by disabling compression.

2.4 Lexicalized Reordering Model

The lexicalized reordering model is trained on parallel data, usually requiring random lookups of the model file during decoding. However, the key to the lookup are the source and target phrase of each translation rule. We shall investigate the advantages of storing the mode data within the translation rule.

3 Experimental Setup

We trained a phrase-based system using the Moses toolkit with standard settings. The training data consisted of most of the publicly available Arabic-English data from Opus (Jrg Tiedemann, 2012,) containing over 69 million parallel sentences, and tuned on a held out set. The phrase-table was then pruned, keeping only the top 100 entries per source phrase, according to $p(t|s)$. All models files were then binarized; the language models were binaized using KenLM (Heafield, 2011), the phrase table using the Probing phrase-table, lexicalized reordering model using the compact datastructure. These binary formats were choosen for their best-in-class multithreaded performance. Table 1 gives details of the resultant sizes of the model files. For verification with a different dataset, we also occasionally used a second system trained on the French-English Europarl corpus (2m parallel sentences). For testing decoding

	ar-en	fr-en
Phrase table	17	5.8
Language model	3.1	1.8
Lex-re model	2.3	637MB

Table 1: Model sizes in GB

speed, we used a subset of the training data, Table 2. The two test scenarios have differing characterics that we are interested in analyzing, ar-en have short sentences with large models while fr-en have overly long sentences with smaller models. Where we need to compare model scores, we used held out test sets.

	ar-en	fr-en
For speed testing		
Set name	Subset of training data	
# sentences	800k	200k
# words	5.8m	5.9m
Avg words/sent	7.3	29.7
For model score testing		
Set name	OpenSubtitles	newstest2011
# sentences	2000	3003
# words	14,620	86,162
Avg words/sent	7.3	28.7

Table 2: Test sets

Standard Moses phrase-based configurations are used, except that we use the cube-pruning algorithm (Chiang, 2007) with a pop-limit of 400, rather than the basic phrase-based algorithm. The cube-pruning algorithm is often employ by users who require fast decoding as it gives them the ability to trade speed with translation quality with a simple pop-limit parameter.

As a baseline, we use the latest version of the Moses decoder taken from the github repository.

For all experiments, we used a Dell PowerEdge R620 server with 16 cores, 32 hyper-threads, split over 2 physical processors (Intel Xeon E5-2650 @ 2.00GHz). The server has 380GB RAM. The operating system was Ubuntu 14.04, the code was compiled with gcc 4.8.4 and Boost library 1.59.

4 Results

4.1 Optimizing Memory

We create a dynamic memory pool which can grow as more memory is requested. The memory is not released, instead the pool can be reset in order for the memory to be re-used. We instantiate two pools for each thread, one which is never reset and another which is reset after the decoding each sentence. Objects are created in either pool according to their life cycle.

For critical objects with high churn such as the hypotheses, thread-specific LIFO queues are used to recycle objects which are no longer used. This not only reduces memory wastage but re-uses recent objects which are likely to be in the CPU cache.

Over 24% of the Moses decoder running time is spent on memory management and this increases when more threads are used, Table 3, dampening the scalability of the decoder. By contrast, our decoder spends 11% on memory management and does not significantly increase with more cores.

	Moses		Our Work	
# threads	1	32	1	32
Memory	24%	39%	11%	13%
LM	12%	2%	47%	38%
Phrase-table	9%	5%	2%	4%
Lex RO	8%	2%	2%	2%
Search	2%	0%	14%	19%
Misc/Unknown	45%	39%	24%	29%

Table 3: Profile of %age decoding time

Figure 2 compares the decoding time for Moses and our decoder, using the same models, parameters and test set. Our decoder is over 3 times faster with one thread, and 4.7% faster using all cores.

Like Moses, performance actually worsens after approximately 15 cores, however, the problem is not as pronounced. This gives us a better foundation on which to build further innovations for fast, multi-core decoding.

Figure 2: Decoding time of Moses and our decoder, using the same models

4.2 Stack Configuration

The most popular stack configuration for phrase-based model, as implemented in Pharaoh, Moses and Joshua, has been by coverage cardinality, ie. hypotheses that have translated the same number of source words are stored in the same stack. There have been research into other stack layouts such as (Ortiz-Martínez et al., 2006), and it has been noted that the decoder in (Brown et al., 1993) uses coverage stacks, as opposed to coverage cardinality.

We also note that distortion limit which constrains hypothesis extension is dependent on the hypothesis' coverage vector, C *and* the end position of most recent source word that has been translated, e. The distortion limit must be checked for every instance of a hypothesis and translation rule, Figure 3. However, by separating hypotheses into set of hypotheses (*'ministacks'*) according to coverage and end position, the distortion limit only needs to be checked for each ministack, Figure 4. Furthermore, stack pruning is done on each of these hypotheses set therefore, changing how hypotheses are grouped can affect model scores. We therefore looked at the effects of three stack configurations:

1. coverage cardinality,
2. coverage,

for all $hypo$ in $stack_{|C|}$ **do**
  for all translation rules **do**
    if can-expand($C(hypo)$, $e(hypo)$, translation rule range) **then**
      expand hypo with translation rule $\rightarrow$ new hypo
      add new hypo to next stack
    end if
  end for
end for

Figure 3: Hypothesis Expansion with Cardinality Stacks

for all $ministack_{C,e}$ in $stack_{|C|}$ **do**
  for all translation rules **do**
    if can-expand(C, e, translation rule range) **then**
      for all $hypo$ in $ministack_{C,e}$ **do**
        expand hypo with translation rule $\rightarrow$ new hypo
        add new hypo to next ministack
      end for
    end if
  end for
end for

Figure 4: Hypothesis Expansion with Coverage & End Position Stacks

3. coverage and end position of most recent translated source word.

Table 4 and Figure 5 present the tradeoff between decoding time and average model at various pop-limits.

As can be seen, the model scores for all stack configurations are identical for low pop-limits parameters but grouping hypotheses into coverage & end position produces higher model scores for higher pop-limits. It is also slower but the time/quality tradeoff is better overall with this stack configuration. For lower pop-limits this configuration is slightly slower, but not by much, therefore, we shall stick with this Configuration for the remainder of the paper.

The cube-pruning algorithm contain a further priority queue which is attached to hypotheses sets which can be independent of how hypotheses are grouped. In the Moses implementation, the priority queue is also attached to the coverage cardinality, Figure 6. Again, we experiment with diffirent queue configurations, having separate queues for

Pop-limit	Cardinality		Coverage		Coverage & end pos	
	Time	Score	Time	Score	Time	Score
100	73	-8.64513	75	-8.64513	72	-8.64513
500	237	-8.59563	225	-8.59563	229	-8.59612
1,000	416	-8.58700	397	-8.58700	423	-8.58700
5,000	1930	-8.58165	1931	-8.58165	2153	-8.58098
10,000	3619	-8.58133	3630	-8.58133	4576	-8.58015
15,000	4830	-8.58130	5001	-8.58130	7156	-8.57999
20,000	5849	-8.58130	5916	-8.58130	9583	-8.57994

Table 4: Decoding time (in secs with 32 threads) and average model scores for different stack configurations

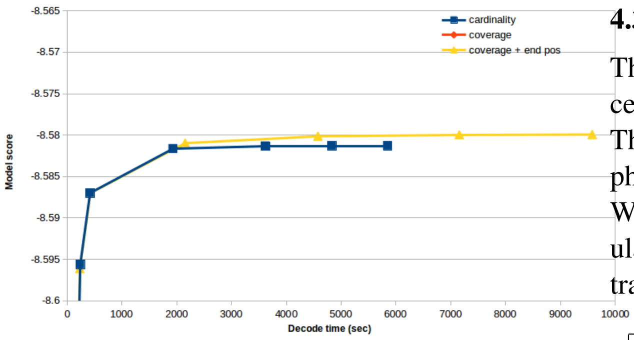

Figure 5: Trade-off between decoding time average model scores for different stack configurations

initialize $queue_{|C|}$
for 1 to pop-limit **do**
  get best $item$ in $queue_{|C|}$
  create hypo from $item$, coverage C_{new}
  add hypo to $stack_{|C_{new}|}$
  create next $items$
  add new $items$ to $queue_{|C|}$
end for

Figure 6: Cube Pruning with Cardinality Stacks

each cardinality, coverage, and coverage & end position. The stack configuration remained constant (coverage & end position with pop-limit of 400). From the results in Table 5, using finer grain

Queue configuration	Time	Score
Cardinality	192	-8.59922
Coverage	2,413	-8.58635
Coverage & end position	7,472	-8.58263

Table 5: Decoding time (in secs with 32 threads) and average model scores for different queue configurations

queues does results in better model scores but it is significantly slower to decode.

4.3 Translation Model

The Moses translation model caches the most recently queried translation rules for later re-use. This has been shown to perform badly for fast phrase-tables such as the Probing phrase-table. We explore a simple caching mechanism that populates the cache during loading with rules that translates the most common source phrases. The

Cache size	Decoding Time	Cache Hit %age
Before caching	229	N/A
0	239 (+4.4%)	0%
1,000	213 (-7.0%)	11%
2,000	204 (-10.9%)	13%
4,000	205 (-10.5%)	14%
10,000	207 (-9.7%)	17%

Table 6: Decoding time (in secs with 32 threads) for varying cache sizes

static cache does not require the overhead of managing the most recently queried lookups but there is still some overhead in using a cache. Overall however, there was over a 10% decrease in decoding time using the optimum cache size, Table 6.

In the second optimatization, we disable the compression. This increase the size of the binary files from 17GB to 23GB but the time saved not needing to decompress the data resulted in a 1.5% decrease in decoding time with 1 thread and nearly 7% when the CPU is saturated, Table 7.

# threads	Compressed pt	Non-compressed pt
1	3052	3006 (-1.5%)
5	756	644 (-14.8%)
10	372	362 (-2.7%)
15	284	250 (-12.0%)
20	244	227 (-7.0%)
25	218	209 (-4.1%)
30	206	192 (-6.8%)
35	203	189 (-6.9%)

Table 7: Decoding time (in secs with 32 threads) for compressed and non-compressed phrase-tables

4.4 Lexicalized Reordering Model

The lexicalized reordering model assign a probability to a translation rule, given the relative ordering of the rule in the hypothesis. Using the model can significantly increase translation quality at the cost of significantly longer decoding time. Efforts such as Junczys-Dowmunt (2012) have been made to increase the speed of the model, with some success, Figure **??**.

We take a different approach by integrating the lexicalized model file into the translation model, reducing the need to query another model file. This resulted in a large decrease in decoding time, especially with high number of cores, Figure 7. Integrating the lexicalized reordering model into

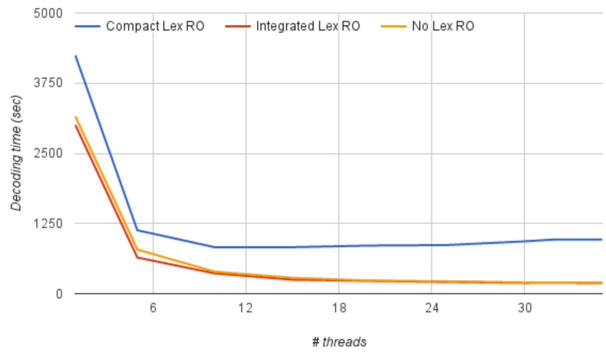

Figure 7: Decoding time (in sec) with Compact Lexicalized Reordering, and integrated into a model the phrase-table

the translation model decreases decoding time by 29% with a single core but it is over 5 times faster using all cores. In fact, the decoding time with the integrated model is similar to that *without* a lexilized reordering model. Critically for systems with large number of cores, it enables the decoder to continue to scale, making efficient use of all available cores, while using a separate lexicalized reordering model decoding time flatten out and actually worsens after approximately 10 threads.

4.5 Core Affinity

The phrase-based decoding algorithm requires fast access to memory in order to Core affinity enables the binding of specific threads to specific CPU cores.

5 BLAH BLAH

The following instructions are directed to authors of papers submitted to and accepted for publication in the ACL 2016 proceedings. All authors are required to adhere to these specifications. Authors are required to provide a Portable Document Format (PDF) version of their papers. The proceedings will be printed on A4 paper. Authors from countries where access to word-processing systems is limited should contact the publication chairs as soon as possible. Grayscale readability of all figures and graphics will be encouraged for all accepted papers (Section 6.8).

Submitted and camera-ready formatting is similar, however, the submitted paper should have:

1. Author-identifying information removed
2. A ‘ruler’ on the left and right margins
3. Page numbers
4. A confidentiality header.

In contrast, the camera-ready **should not have** a ruler, page numbers, nor a confidentiality header. By uncommenting `\aclfinalcopy` at the top of this document, it will compile to produce an example of the camera-ready formatting; by leaving it commented out, the document will be anonymized for initial submission. Authors should place this command after the `\usepackage` declarations when preparing their camera-ready manuscript with the ACL 2016 style.

6 General Instructions

Manuscripts must be in two-column format. Exceptions to the two-column format include the title, as well as the authors’ names and complete addresses (only in the final version, not in the version submitted for review), which must be centered at the top of the first page (see the guidelines in Subsection 6.4), and any full-width figures or tables. Type single-spaced. Do not number the pages in the camera-ready version. Start all pages directly under the top margin. See the guidelines later regarding formatting the first page.

The maximum length of a manuscript is eight (8) pages for the main conference, printed single-sided, plus two (2) pages for references (see Section 7 for additional information on the maximum number of pages).

By uncommenting `\aclfinalcopy` at the top of this document, it will compile to produce an example of the camera-ready formatting; by leaving it commented out, the document will be anonymized for initial submission. When you first create your submission on softconf, please fill in your submitted paper ID where `***` appears in the `\def\aclpaperid{***}` definition at the top.

The review process is double-blind, so do not include any author information (names, addresses) when submitting a paper for review. However, you should maintain space for names and addresses so that they will fit in the final (accepted) version. The ACL 2016 LaTeX style will create a titlebox space of 2.5in for you when `\aclfinalcopy` is commented out.

6.1 The Ruler

The ACL 2016 style defines a printed ruler which should be presented in the version submitted for review. The ruler is provided in order that reviewers may comment on particular lines in the paper without circumlocution. If you are preparing a document without the provided style files, please arrange for an equivalent ruler to appear on the final output pages. The presence or absence of the ruler should not change the appearance of any other content on the page. The camera ready copy should not contain a ruler. (LaTeX users may uncomment the `\aclfinalcopy` command in the document preamble.)

Reviewers: note that the ruler measurements do not align well with lines in the paper — this turns out to be very difficult to do well when the paper contains many figures and equations, and, when done, looks ugly. Just use fractional references (e.g., the first line on this page is at mark 096.5), although in most cases one would expect that the approximate location will be adequate.

6.2 Electronically-available resources

ACL provides this description in LaTeX2e (`acl2016.tex`) and PDF format (`acl2016.pdf`), along with the LaTeX2e style file used to format it (`acl2016.sty`) and an ACL bibliography style (`acl2016.bst`) and example bibliography (`acl2016.bib`). These files are all available at `acl2016.org/index.php?article_id=9`. We strongly recommend the use of these style files, which have been appropriately tailored for the ACL 2016 proceedings.

6.3 Format of Electronic Manuscript

For the production of the electronic manuscript, you must use Adobe's Portable Document Format (PDF). This format can be generated from postscript files: on Unix systems, you can use `ps2pdf` for this purpose; under Microsoft Windows, you can use Adobe's Distiller, or if you have cygwin installed, you can use `dvipdf` or `ps2pdf`. Note that some word processing programs generate PDF that may not include all the necessary fonts (esp. tree diagrams, symbols). When you print or create the PDF file, there is usually an option in your printer setup to include none, all, or just non-standard fonts. Please make sure that you select the option of including ALL the fonts. *Before sending it, test your* PDF *by printing it from a computer different from the one where it was created.* Moreover, some word processors may generate very large postscript/PDF files, where each page is rendered as an image. Such images may reproduce poorly. In this case, try alternative ways to obtain the postscript and/or PDF. One way on some systems is to install a driver for a postscript printer, send your document to the printer specifying "Output to a file", then convert the file to PDF.

For reasons of uniformity, Adobe's **Times Roman** font should be used. In LaTeX2e this is accomplished by putting

```
\usepackage{times}
\usepackage{latexsym}
```

in the preamble.

Print-outs of the PDF file on A4 paper should be identical to the hardcopy version. If you cannot meet the above requirements about the production of your electronic submission, please contact the publication chairs above as soon as possible.

6.4 The First Page

Center the title, author name(s) and affiliation(s) across both columns (or, in the case of initial submission, space for the names). Do not use footnotes for affiliations. Use the two-column format only when you begin the abstract.

Title: Place the title centered at the top of the first page, in a 15 point bold font. (For a complete guide to font sizes and styles, see Table 9.) Long titles should be typed on two lines without a blank line intervening. Approximately, put the title at 1in from the top of the page, followed by a blank line, then the author name(s), and the affiliation(s) on the following line. Do not use only initials for given names (middle initials are allowed). Do not format surnames in all capitals (e.g., "Mitchell," not "MITCHELL"). The affiliation should contain the author's complete address, and if possible, an electronic mail address. Leave about 0.75in between the affiliation and the body of the first page.

Command	Output	Command	Output
{\"a}	ä	{\c c}	ç
{\^e}	ê	{\u g}	ğ
{\`i}	ì	{\l}	ł
{\.I}	İ	{\~n}	ñ
{\o}	ø	{\H o}	ő
{\'u}	ú	{\v r}	ř
{\aa}	å	{\ss}	ß

Table 8: Example commands for accented characters, to be used in, e.g., BIBTEX names.

Abstract: Type the abstract at the beginning of the first column. The width of the abstract text should be smaller than the width of the columns for the text in the body of the paper by about 0.25in on each side. Center the word **Abstract** in a 12 point bold font above the body of the abstract. The abstract should be a concise summary of the general thesis and conclusions of the paper. It should be no longer than 200 words. The abstract text should be in 10 point font.

Text: Begin typing the main body of the text immediately after the abstract, observing the two-column format as shown in the present document. Do not include page numbers in the camera-ready manuscript.

Indent when starting a new paragraph. For reasons of uniformity, use Adobe's **Times Roman** fonts, with 11 points for text and subsection headings, 12 points for section headings and 15 points for the title. If Times Roman is unavailable, use **Computer Modern Roman** (LATEX2e's default; see section 6.3 above). Note that the latter is about 10% less dense than Adobe's Times Roman font.

6.5 Sections

Headings: Type and label section and subsection headings in the style shown on the present document. Use numbered sections (Arabic numerals) in order to facilitate cross references. Number subsections with the section number and the subsection number separated by a dot, in Arabic numerals.

Citations: Citations within the text appear in parentheses as (Gusfield, 1997) or, if the author's name appears in the text itself, as Gusfield (1997). Using the provided LATEX style, the former is accomplished using `\cite` and the latter with `\shortcite` or `\newcite`. Collapse multiple citations as in (Gusfield, 1997; Aho and Ullman, 1972); this is accomplished with the provided style using commas within the `\cite` command, e.g., `\cite{Gusfield:97,Aho:72}`. Append lowercase letters to the year in cases of ambiguities. Treat double authors as in (Aho and Ullman, 1972), but write as in (Chandra et al., 1981) when more than two authors are involved.

References: We recommend including references in a separate `.bib` file, and include an example file in this release (`naalhlt2016.bib`). Some commands for names with accents are provided for convenience in Table 8. References stored in the separate `.bib` file are inserted into the document using the following commands:

```
\bibliography{acl2016}
\bibliographystyle{acl2016}
```

References should appear under the heading **References** at the end of the document, but before any Appendices, unless the appendices contain references. Arrange the references alphabetically by first author, rather than by order of occurrence in the text. Provide as complete a reference as possible, using a consistent format, such as the one for *Computational Linguistics* or the one in the *Publication Manual of the American Psychological Association* (American Psychological Association, 1983). Authors' full names rather than initials are preferred. You may use **standard** abbreviations for conferences[1] and journals[2].

Appendices: Appendices, if any, directly follow the text and the references (but see above). Letter them in sequence and provide an informative title: **Appendix A. Title of Appendix**.

Acknowledgment sections should go as a last (unnumbered) section immediately before the references.

6.6 Footnotes

Footnotes: Put footnotes at the bottom of the page. They may be numbered or referred to by asterisks or other symbols.[3] Footnotes should be separated from the text by a line.[4] Footnotes should be in 9 point font.

6.7 Graphics

Illustrations: Place figures, tables, and photographs in the paper near where they are first discussed, rather than at the end, if possible. Wide illustrations may run across both columns and

[1] https://en.wikipedia.org/wiki/List_of_computer_science_conference_acronyms

[2] http://www.abbreviations.com/jas.php

[3] This is how a footnote should appear.

[4] Note the line separating the footnotes from the text.

Type of Text	Font Size	Style
paper title	15 pt	bold
author names	12 pt	bold
author affiliation	12 pt	
the word “Abstract”	12 pt	bold
section titles	12 pt	bold
document text	11 pt	
abstract text	10 pt	
captions	9 pt	
caption label	9 pt	bold
bibliography	10 pt	
footnotes	9 pt	

Table 9: Font guide.

should be placed at the top of a page. Color illustrations are discouraged, unless you have verified that they will be understandable when printed in black ink.

Captions: Provide a caption for every illustration; number each one sequentially in the form: “**Figure 1:** Figure caption.”, “**Table 1:** Table caption.” Type the captions of the figures and tables below the body, using 9 point text. Table and Figure labels should be bold-faced.

6.8 Accessibility

In an effort to accommodate the color-blind (as well as those printing to paper), grayscale readability for all accepted papers will be encouraged. Color is not forbidden, but authors should ensure that tables and figures do not rely solely on color to convey critical distinctions. Here we give a simple criterion on your colored figures, if your paper has to be printed in black and white, then you must assure that every curves or points in your figures can be still clearly distinguished.

7 Length of Submission

The ACL 2016 main conference accepts submissions of long papers and short papers. Long papers may consist of up to eight (8) pages of content, plus unlimited pages for references. Upon acceptance, final versions of long papers will be given one additional page (up to 9 pages with unlimited pages for references) so that reviewers' comments can be taken into account. Short papers may consist of up to four (4) pages of content, plus unlimited pages for references. Upon acceptance, short papers will be given five (5) pages in the proceedings and unlimited pages for references. For both long and short papers, all illustrations and appendices must be accommodated within these page limits, observing the formatting instructions given in the present document. Papers that do not conform to the specified length and formatting requirements are subject to be rejected without review.

8 Double-blind review process

As the reviewing will be blind, the paper must not include the authors' names and affiliations. Furthermore, self-references that reveal the author's identity, e.g., “We previously showed (Smith, 1991) ...” must be avoided. Instead, use citations such as “Smith previously showed (Smith, 1991) ...” Papers that do not conform to these requirements will be rejected without review. In addition, please do not post your submissions on the web until after the review process is complete (in special cases this is permitted: see the multiple submission policy below).

We will reject without review any papers that do not follow the official style guidelines, anonymity conditions and page limits.

9 Multiple Submission Policy

Papers that have been or will be submitted to other meetings or publications must indicate this at submission time. Authors of papers accepted for presentation at ACL 2016 must notify the program chairs by the camera-ready deadline as to whether the paper will be presented. All accepted papers must be presented at the conference to appear in the proceedings. We will not accept for publication or presentation papers that overlap significantly in content or results with papers that will be (or have been) published elsewhere.

Preprint servers such as arXiv.org and ACL-related workshops that do not have published proceedings in the ACL Anthology are not considered archival for purposes of submission. Authors must state in the online submission form the name of the workshop or preprint server and title of the non-archival version. The submitted version should be suitably anonymized and not contain references to the prior non-archival version. Reviewers will be told: “The author(s) have notified us that there exists a non-archival previous version of this paper with significantly overlapping text. We have approved submission under these circumstances, but to preserve the spirit of blind review, the current submission does not reference the non-archival version.” Reviewers are free to do what they like with this information.

Authors submitting more than one paper to ACL must ensure that submissions do not overlap significantly ($> 25\%$) with each other in content or results. Authors should not submit short and long versions of papers with substantial overlap in their original contributions.

Acknowledgments

Do not number the acknowledgment section. This section should not be presented for the submission version.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

American Psychological Association. 1983. *Publications Manual*. American Psychological Association, Washington, DC.

Nikolay Bogochev. 2013. The Paradox of Overfitting. Master's thesis, University of Edinburgh, UK.

Peter F. Brown, Stephen A. Della-Pietra, Vincent J. Della-Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation. *Computational Linguistics*, 19(2):263–313.

Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. 1981. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, United Kingdom, July.

Marcin Junczys-Dowmunt. 2012. A space-efficient phrase table implementation using minimal perfect hash functions. In Petr Sojka, Ales Hork, Ivan Kopecek, and Karel Pala, editors, *15th International Conference on Text, Speech and Dialogue (TSD)*, volume 7499 of *Lecture Notes in Computer Science*, pages 320–327. Springer.

Daniel Ortiz-Martínez, Ismael García-Varea, and Francisco Casacuberta. 2006. Generalized stack decoding algorithms for statistical machine translation. In *Proceedings on the Workshop on Statistical Machine Translation*, pages 64–71, New York City, June. Association for Computational Linguistics.