PBML

logo

The Prague Bulletin of Mathematical Linguistics
NUMBER ??? AUGUST 2012 1–11

Better Splitting Algorithms for Parallel Corpus Processing

Lane Schwartz

Air Force Research Laboratory
Human Effectiveness Directorate
Wright-Patterson AFB, OH USA

Abstract

Each iteration of minimum error rate training involves re-translating a development set. Distributing this work across computational nodes can speed up translation time, but in practice some parts may take much longer to complete than others, leading to computational slack time. To address this problem, we develop three novel algorithms for distributing translation tasks in a parallel computing environment, drawing on research in parallel machine scheduling. We present results showing a substantial speedup in overall decoding time.

1. Introduction

The task of translation involves translating a source language document $\mathbf{f}$ into target language $\boldsymbol{e}$. Most popular statistical translation techniques select the best translation $\hat{e}$ for source sentence f according to a linear combination of models $\boldsymbol{\phi}$ using a set of model weights $\boldsymbol{\lambda}$ (Och and Ney, 2002).

$$\hat{e} = \arg\max_{e} \sum_{i} \lambda_i \phi_i(e, f) \tag{1}$$

Values for $\boldsymbol{\lambda}$ are obtained by optimizing an objective function such as BLEU (Papineni et al., 2001) against a development set, most commonly using minimum error rate training (MERT) (Och, 2003). Each iteration of MERT requires this development set to be re-translated using a new set of $\boldsymbol{\lambda}$ weights. MERT is one of the slowest components in a typical machine translation training pipeline, and translating the development set is nearly always the slowest step in MERT. We now examine techniques

 Corresponding author: Lane.Schwartz@wpafb.af.mil
Cite as: Lane Schwartz. Better Splitting Algorithms for Parallel Corpus Processing. The Prague Bulletin of Mathematical Linguistics No. ???, 2012, pp. 1–11. doi: HERE-WILL-BE-A-DOI-NUMBER

for speeding up the translation process by splitting a source document into parts and distributing the translation of those parts across parallel computational nodes.

Ideally, all parts should take the same amount of time to translate. While naive splitting techniques reduce the time required for each translation iteration by splitting the work between n computational nodes, in practice some parts may take much longer to complete than others. This can lead to significant computational slack time. To address this problem, we develop three novel algorithms for splitting translation tasks in a parallel computing environment, drawing on research in parallel machine scheduling.

2. Related Work

Research into parallel machine scheduling problems constitutes a wide and well-studied field, ranging through various disciplines of engineering, manufacturing, and management in addition to computer science and applied mathematics (Cheng and Sin, 1999), spanning a wide range of scheduling techniques (Panwalkar and Iskander, 1977).

We now briefly examine the existing research most relevant to our task. Hu (1961) and Graham (1966; 1969) develop various list scheduling algorithms. This family of algorithms prioritizes jobs into a queue, then assigns jobs to machines in queue order. This approach attempts to evenly balance the load on each execution host (De and Morton, 1980; Cheng and Sin, 1999). Both Algorithm 1 below and the techniques we develop in Section 3 fall into this family of algorithms.

Algorithm 1 Split input text into n parts such that each part contains the same number of lines. In cases where the total number of lines is not evenly divisible by n, the last part will contain fewer lines than each of the other parts.

```
function NAIVE-SPLIT(n,input)
    s ← input.length
    ℓ ← ⌈s/n⌉
    for p ← 0...(n − 1) do
        i ← ℓ × p
        for j ← i...min(i + ℓ − 1,s − 1) do
            output[p].append(input[j])
        end for
    end for
    return output
end function
```

While the models in Equation 1 could, in theory, condition on previously translated sentences, in practice virtually no widely used models do so. It is therefore very

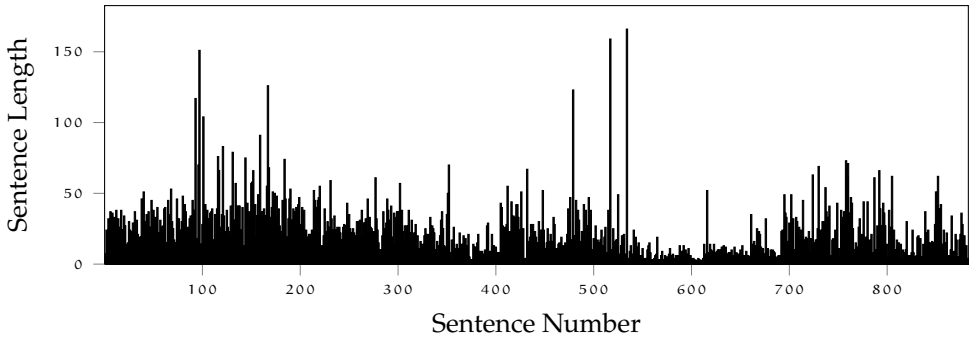


Figure 1: Sentence length (in words) of sentences from the NIST OpenMT 2008 Urdu-English task.

straightforward to split the data into n parts, and translate each part independently on n computational nodes.

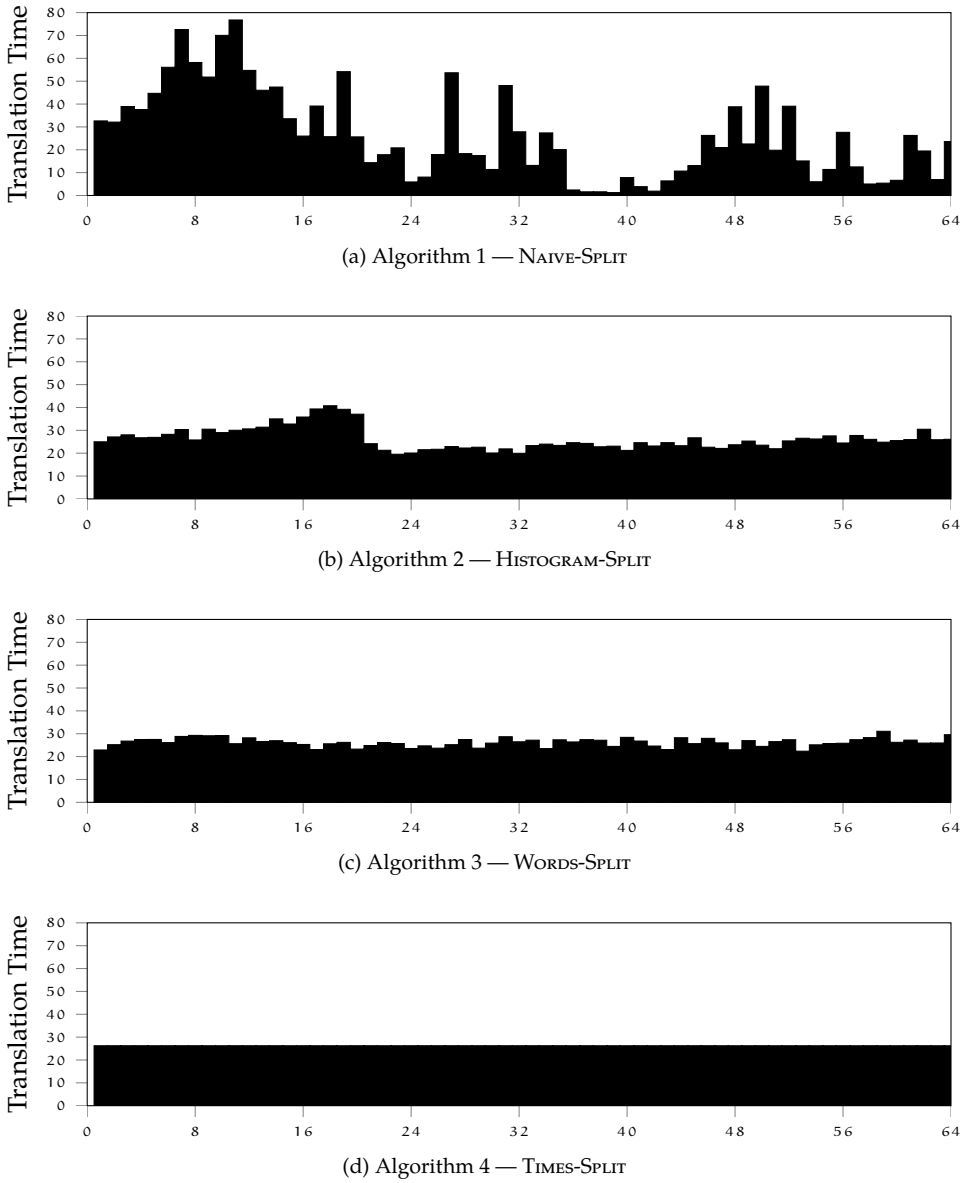
3. Better Splitting Algorithms

We begin by examining the development set for the NIST OpenMT 2008 Urdu-English task. When tuning the model weights λ, this development set will be translated numerous times. We observe in Figure 1 that the number of words in each sentence varies widely and unevenly throughout the corpus.

Using Algorithm 1, scripts included with Moses (Koehn et al., 2007) split the corpus into n parts of of ℓ or fewer lines each; ℓ is the smallest integer greater than or equal to s/n, where s is the total number of input sentences. The first ℓ lines comprise the first part, the next ℓ lines comprise the second part, and so on. Thus, each part contains exactly ℓ lines, with the possible exception of the last part, which contains fewer than ℓ lines when s is not evenly divisible by n.

Figure 2a shows the amount of time taken to translate each part of the development set, split using Algorithm 1 into 64 parts, using Moses configured with a 5-gram language model. We observe that the shape of Figure 2a generally matches that of Figure 1. There is substantial variance in translation time between the parts in Figure 2a, with some parts taking nearly 80 seconds and many others finishing in well under 10 seconds. We observe that this disparity is largely due to an imbalance of short versus long sentences between the parts. Because short sentences take less time to translate than long sentences, parts assigned mostly short sentences finish much faster than parts that are assigned many longer sentences.

To remedy this imbalance, we propose Algorithm 2. Prior to splitting the data into parts, Algorithm 2 begins by examining the number of words in each sentence.



(a) Algorithm 1 — NAIVE-SPLIT

(b) Algorithm 2 — HISTOGRAM-SPLIT

(c) Algorithm 3 — WORDS-SPLIT

(d) Algorithm 4 — TIMES-SPLIT

Figure 2: Total translation time (in **seconds**) for each part when data from the NIST OpenMT 2008 Urdu-English task is split into 64 parts using each of four algorithms.

Algorithm 2 Split input text into n parts to balance the histograms of line lengths for all parts.

```
function HISTOGRAM-SPLIT(n,input)
    for i ← 0 ... ( input.length −1) do
        sentence[i].length ← input[i].length
        sentence[i].index ← i
    end for
    SORT(sentence) {|x, y| x.length ⇔ y.length}
                                              ▷ Sort sentences by length
    p ← n
    for i ← 0 ... (input.length −1) do
        if p < n then
            p ← p + 1
        else
            p ← 0
        end if
        output[p].append(input[sentence[i].index])
    end for
    return output
end function
```

Sentences are sorted according to length, then assigned in turns to parts. This round-robin distribution of sentences into parts results in the sentence length histograms for each part being approximately equal. While Algorithm 2 attempts to balance short and long sentences across parts, we nevertheless observe non-trivial imbalance in translation times across parts in Figure 2b.

To improve this remaining imbalance, we propose Algorithm 3. In Algorithm 3, sentences are sorted by length into a queue, with longest sentences at the head of the queue. Initially, no sentences have been assigned to any part. The longest sentence, at the head of the queue, is assigned first to a part. As each sentence is assigned to a part, the total number of words assigned to that part is recorded. Each subsequent sentence is removed from the queue and assigned to the part with the least work assigned to it, as measured by number of words. In Figure 2c, we observe that most of the imbalance in translation times across parts has been resolved.

When assigning sentences to jobs, we would ideally like to know how long each sentence will take to process. Algorithms 2 and 3 use the number of words in each sentence as a proxy for processing time. During MERT, the same set of development sentences are translated multiple times. Since each decoding process differs only by the λ weights used, it is reasonable to expect little variation in decoding runtime for any given sentence across all MERT runs. With this in mind, we record the time required to translate each sentence during the first iteration of MERT. In subsequent

Algorithm 3 Split input text into n parts to balance the number of words for all parts.

```
function WORDS-SPLIT(n,input)
    for i ← 0 ... ( input.length −1) do
        sentence[i].length ← input[i].length
        sentence[i].index ← i
    end for
    SORT(sentence) {|x, y| y.length ⇔ x.length}
                                        ▷ Sort sentences by length, in reverse order
    for i ← 0 ... ( input.length −1) do
        p ← LEAST(words)
                                                ▷ Find partition with fewest words
        output[p].append(input[sentence[i].index])
        words[p] ←words[p] + sentence[i].length
    end for
    return output
end function
```

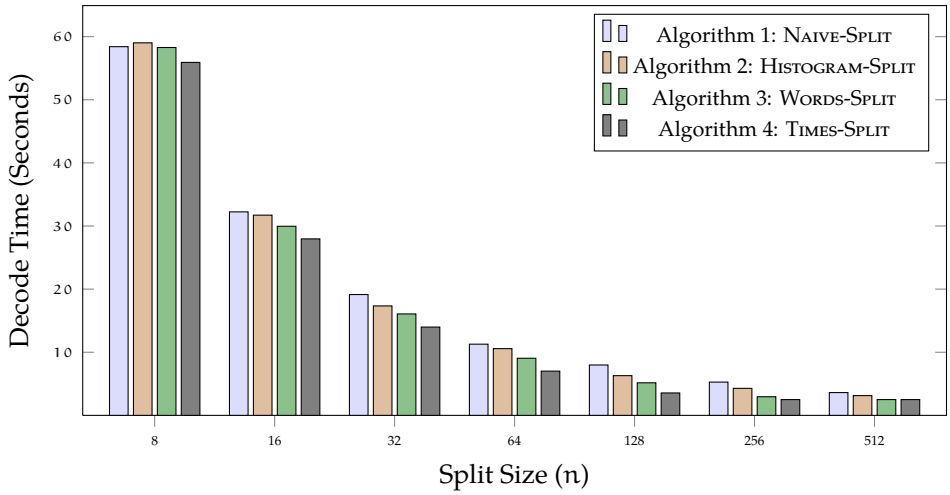
iterations, Algorithm 4 uses the time recorded to translate a sentence as an estimate of the time it will take to translate that sentence again. Algorithm 4 differs from Algorithm 3 by sorting using these times instead of sentence length. We see in Figure 2d that the time imbalance between parts is virtually non-existent, with all times now within 0.01 seconds of each other.

4. Experimental Configuration

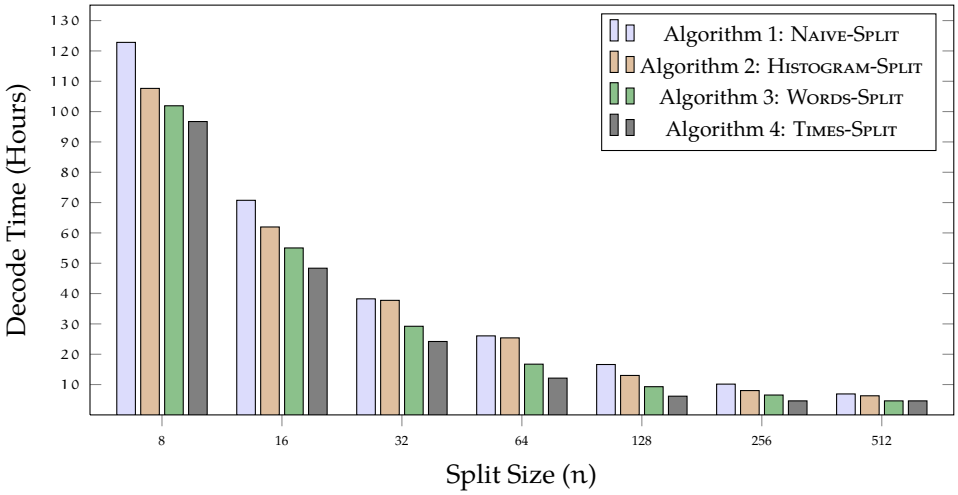
To observe the effects of splitting algorithms on decoding speed, we translated the NIST OpenMT 2008 development set of Urdu-English data using Moses in a parallel computing cluster, distributing work using the Sun Grid Engine. We ran two decoding setups: a standard configuration using a 5-gram language model, and a much slower syntactic LM configuration following Schwartz et al. (2011).

Figure 2 shows the per-part translation times for all parts of the development set when $n = 64$. In Figure 3, we examine the per-part translation times for only the slowest of n translation jobs in each configuration for various values of n, ranging from 2–512. In all configurations, we see that Algorithm 4 provides the fastest performance.

Another metric to use in examining our algorithms is total computational slack time. During MERT, computational slack time arises when some parts of the development set finish translating faster than others. Figure 4 lists the decoding times of the fastest and slowest translation jobs for parts split using each of the four algorithms for various values of n, ranging from 2–512. We see the total cumulative slack time for each of these conditions in Figure 5.



(a) Decoding times in **seconds** for decoder configured using a 5-gram language model.



(b) Decoding times in **hours** for decoder configured using a syntactic language model (Schwartz et al., 2011) in addition to a 5-gram language model.

Figure 3: Decoding times for the slowest translation job in a translation task split into n decoding jobs using various splitting algorithms (Naive-Split, Histogram-Split, Words-Split, and Times-Split).

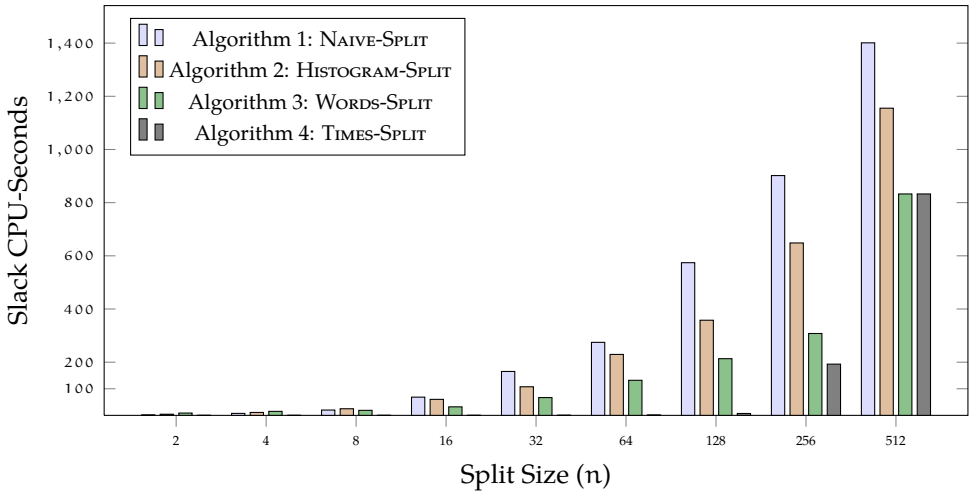
Split Size (n)	Algorithm 1 Naive-Split		Algorithm 2 Histogram-Split		Algorithm 3 Words-Split		Algorithm 4 Times-Split	
	Min	Max	Min	Max	Min	Max	Min	Max
2	222.9	224.4	221.5	225.8	219.3	228.0	*223.7*	***223.7***
4	109.2	113.7	110.0	114.6	108.7	115.6	*111.8*	***111.8***
8	51.4	58.4	52.2	59.0	53.2	58.3	*55.9*	***55.9***
16	24.9	32.2	25.0	31.7	25.3	30.0	27.9	**28.0**
32	11.3	19.1	11.9	17.3	11.7	16.1	*14.0*	***14.0***
64	5.4	11.3	5.4	10.6	5.7	9.1	*7.0*	***7.0***
128	1.3	8.0	2.2	6.3	2.3	5.2	*3.5*	***3.5***
256	0.3	5.3	0.7	4.3	0.8	3.0	1.7	**2.5**
512	0.0	3.6	0.2	3.1	0.3	**2.5**	0.6	**2.5**

(a) Decoding times in **seconds** for decoder configured using a 5-gram language model.

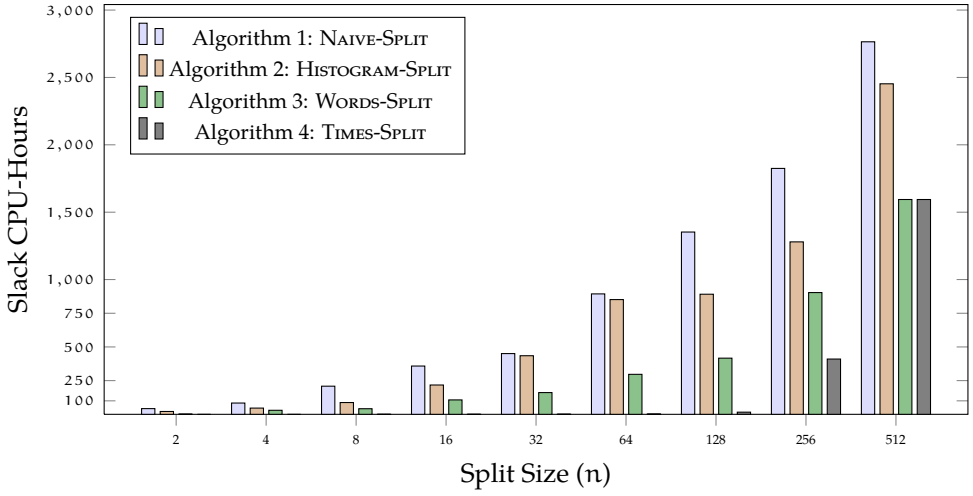
Split Size (n)	Algorithm 1 Naive-Split		Algorithm 2 Histogram-Split		Algorithm 3 Words-Split		Algorithm 4 Times-Split	
	Min	Max	Min	Max	Min	Max	Min	Max
2	365.7	408.0	376.3	397.5	386.1	387.6	*386.9*	***386.9***
4	176.1	214.4	186.8	205.0	184.6	200.9	*193.4*	***193.4***
8	84.6	122.8	84.8	107.6	88.4	101.9	*96.7*	***96.7***
16	40.8	70.8	40.5	62.0	45.3	55.0	*48.4*	***48.4***
32	19.2	38.3	18.8	37.8	20.5	29.2	*24.2*	***24.2***
64	9.2	26.1	9.2	25.4	9.4	16.7	*12.1*	***12.1***
128	2.7	16.6	4.1	13.0	3.7	9.3	5.9	**6.2**
256	0.7	10.2	1.3	8.0	1.4	6.6	2.9	**4.6**
512	0.0	6.9	0.3	6.3	0.6	**4.6**	1.1	**4.6**

(b) Decoding times in **hours** for decoder configured using a syntactic language model (Schwartz et al., 2011) addition to a 5-gram language model.

Figure 4: Decoding times for the fastest (min) and slowest (max) decoding jobs when a translation task is split into n decoding jobs. **Bold** indicates fastest max time at that split. *Italics* indicate balanced task times, corresponding to zero slack time (see Figure 5).



(a) Slack **CPU-Seconds** for decoder configured using a 5-gram language model.



(b) Slack **CPU-Hours** for decoder configured using a syntactic language model (Schwartz et al., 2011) in addition to a 5-gram language model.

Figure 5: Cumulative slack CPU time for n processing cores when processing a parallel translation task split into n jobs using various splitting algorithms. Slack CPU time is caused when some jobs finish before others. Zero slack time indicates conditions where all jobs complete simultaneously.

Algorithm 4 Split input text into n parts to balance the estimated translation time of all parts.

```
function TIMES-SPLIT(n,input,estimate)
    for i ← 0 ... ( input.length −1) do
        sentence[i].time ← estimate[i]
        sentence[i].index ← i
    end for
    SORT(sentence) {|x, y| y.time ⇔ x.time}
                                        ▷ Sort sentences by time, in reverse order
    for i ← 0 ... ( input.length −1) do
        p ← LEAST(times)
                                                ▷ Find partition with least time
        output[p].append(input[sentence[i].index])
        times[p] ← times[p] + sentence[i].time
    end for
    return output
end function
```

5. Conclusion

While statistical translation models could, in theory, condition on previously translated sentences, in practice virtually no widely used models do so. Translation is therefore embarrassingly parallel — a document to be translated can be split into n parts, with each part translated independently a different computational node.

While such splitting is commonly performed, a suboptimal naive splitting technique (Algorithm 1) is used by all translation software of which we are aware. In this work we have presented three more effective corpus splitting algorithms (Algorithms 2, 3 and 4), enabling substantial speed-ups in parallel decoding time at no additional cost.

We observe that while Algorithm 2 fails to improve over Algorithm 1 for a standard Moses configuration for small values of n, for values of $n > 8$, and for all values of n using the slow syntactic language model, Algorithm 2 represents a clear improvement. Results for Algorithm 3 show further speedups over Algorithms 1 and 2 in most configurations. Algorithm 4 is the clear winner, nearly eliminating slack time in many cases.

While the most effective algorithm (Algorithm 4) requires per-sentence decode times from previous decodes, in most realistic settings, Algorithm 3 provides much of the benefit of Algorithm 4 in terms of decreased computational slack time while requiring little changes to existing decoding scripts which use the naive Algorithm 1.

Implementations in Ruby and Java of all four splitting algorithms are provided at `http://github.com/dowobeha/balance-corpus`.

Bibliography

Cheng, T.C.E. and C.C.S. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47:271–292, 1999.

De, Prabuddha and Thomas E. Morton. Scheduling to minimum makespan on unequal parallel processors. *Decision Sciences*, 11(4):586–602, October 1980.

Graham, Ron L. Bounds on certain multiprocessing timing anomalies. *The Bell Systems Technical Journal*, 45(9):1563–1581, November 1966.

Graham, Ron L. Bounds on certain multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, March 1969.

Hu, T.C. Parallel sequencing and assembly line problems. *Operations Research*, 9(6):841–848, 1961.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007.

Och, Franz. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Sapporo, Japan, July 2003.

Och, Franz and Hermann Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 295–302, Philadelphia, Pennsylvania, July 2002.

Panwalkar, S.S. and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1): 45–61, 1977.

Papineni, Kishore, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Toulouse, France, 2001.

Schwartz, Lane, Chris Callison-Burch, William Schuler, and Stephen Wu. Incremental syntactic language models for phrase-based translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Techologies*, pages 620–631, Portland, Oregon, USA, June 2011.

Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force. Cleared for public release on 19 Jan 2011. Originator reference number RH-11-107802. Case number 88ABW-2011-0185. Thanks to Grant Erdmann and Tim Anderson, and to the anonymous reviewers.