Linear Text Segmentation Using Affinity Propagation

Anna Kazantseva
School of Electrical Engineering
and Computer Science,
University of Ottawa
ankazant@site.uottawa.ca

Stan Szpakowicz
School of Electrical Engineering
and Computer Science,
University of Ottawa &
Institute of Computer Science,
Polish Academy of Sciences
szpak@site.uottawa.ca

Abstract

This paper presents a new algorithm for linear text segmentation. It is an adaptation of Affinity Propagation, a state-of-the-art clustering algorithm in the framework of factor graphs. Affinity Propagation for Segmentation, or *APS*, receives a set of pairwise similarities between data points and produces segment boundaries and segment *centres* – data points which best describe all other data points within the segment. *APS* iteratively passes messages in a cyclic factor graph, until convergence. Each iteration works with information on all available similarities, resulting in high-quality results. *APS* scales linearly for realistic segmentation tasks. We derive the algorithm from the original Affinity Propagation formulation, and evaluate its performance on topical text segmentation in comparison with two state-of-the art segmenters. The results suggest that *APS* performs on par with or outperforms these two very competitive baselines.

1 Introduction

In complex narratives, it is typical for the topic to shift continually. Some shifts are gradual, others – more abrupt. *Topical text segmentation* identifies the more noticeable topic shifts. A topical segmenter's output is a very simple picture of the document's structure. Segmentation is a useful intermediate step in such applications as subjectivity analysis (Stoyanov and Cardie, 2008), automatic summarization (Haghighi and Vanderwende, 2009), question answering (Oh, Myaeng, and Jang, 2007) and others. That is why improved quality of text segmentation can benefit other language-processing tasks.

We present Affinity Propagation for Segmentation (*APS*), an adaptation of a state-of-the-art clustering algorithm, Affinity Propagation (Frey and Dueck, 2007; Givoni and Frey, 2009).[1] The original AP algorithm considerably improved exemplar-based clustering both in terms of speed and the quality of solutions. That is why we chose to adapt it to segmentation. At its core, *APS* is suitable for segmenting any sequences of data, but we present it in the context of segmenting documents. *APS* takes as input a matrix of pairwise similarities between sentences and, for each sentence, a *preference* value which indicates an *a priori* belief in how likely a sentence is to be chosen as a segment centre. *APS* outputs segment assignments and segment centres – data points which best explain all other points in a segment. The algorithm attempts to maximize *net similarity* – the sum of similarities between all data points and their respective segment centres.

APS operates by iteratively passing messages in a factor graph (Kschischang, Frey, and Loeliger, 2001) until a good set of segments emerges. Each iteration considers all similarities – takes into account all available information. An iteration includes sending at most $O(N^2)$ messages. For the majority of realistic segmentation tasks, however, the upper bound is $O(MN)$ messages, where M is a constant. This is more computationally expensive than the requirements of locally informed segmentation algorithms such as those based on HMM or CRF (see Section 2), but for a globally-informed algorithm the requirements are very reasonable. *APS* is an instance of loopy-belief propagation (belief propagation on cyclic graphs) which has

[1] An implementation of *APS* in Java, and the data sets, can be downloaded at ⟨www.site.uottawa.ca/∼ankazant⟩.

Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, pages 284–293,
Edinburgh, Scotland, UK, July 27–31, 2011.

been used to achieved state-of-the-art performance in error-correcting decoding, image processing and data compression. Theoretically, such algorithms are not guaranteed to converge or to maximize the objective function. Yet in practice they often achieve competitive results.

APS works on an already pre-compiled similaritiy matrix, so it offers flexibility in the choice of similarity metrics. The desired number of segments can be set by adjusting preferences.

We evaluate the performance of *APS* on three tasks: finding topical boundaries in transcripts of course lectures (Malioutov and Barzilay, 2006), identifying sections in medical textbooks (Eisenstein and Barzilay, 2008) and identifying chapter breaks in novels. We compare *APS* with two recent systems: the Minimum Cut segmenter (Malioutov and Barzilay, 2006) and the Bayesian segmenter (Eisenstein and Barzilay, 2008). The comparison is based on the WindowDiff metric (Pevzner and Hearst, 2002). *APS* matches or outperforms these very competitive baselines.

Section 2 of the paper outlines relevant research on topical text segmentation. Section 3 briefly covers the framework of factor graphs and outlines the original Affinity Propagation algorithm for clustering. Section 4 contains the derivation of the new update messages for APSeg. Section 5 describes the experimental setting, Section 6 reports the results, Section 7 discusses conclusions and future work.

2 Related Work

This sections discusses selected text segmentation methods and positions the proposed *APS* algorithm in that context.

Most research on automatic text segmentation revolves around a simple idea: when the topic shifts, so does the vocabulary (Youmans, 1991). We can roughly subdivide existing approaches into two categories: locally informed and globally informed.

Locally informed segmenters attempt to identify topic shifts by considering only a small portion of complete document. A classical approach is TextTiling (Hearst, 1997). It consists of sliding two adjacent windows through text and measuring lexical similarity between them. Drops in similarity correspond to topic shifts. Other examples include text segmentation using Hidden Markov Models (Blei and Moreno, 2001) or Conditional Random Fields (Lafferty, McCallum, and Pereira, 2001). Locally informed methods are often very efficient because of lean memory and CPU time requirements. Due to a limited view of the document, however, they can easily be thrown off by short inconsequential digressions in narration.

Globally informed methods consider "the big picture" when determining the most likely location of segment boundaries. Choi (2000) applies divisive clustering to segmentation. Malioutov and Barzilay (2006) show that the knowledge about long-range similarities between sentences improves segmentation quality. They cast segmentation as a graph-cutting problem. The document is represented as a graph: nodes are sentences and edges are weighted using a measure of lexical similarity. The graph is cut in a way which maximizes the net edge weight within each segment and minimizes the net weight of severed edges. Such *Minimum Cut* segmentation resembles *APS* the most among others mentioned in this paper. The main difference between the two is in different objective functions.

Another notable direction in text segmentation uses generative models to find segment boundaries. Eisenstein and Barzilay (2008) treat words in a sentence as draws from a multinomial language model. Segment boundaries are assigned so as to maximize the likelihood of observing the complete sequence. Misra et al. (2009) use a Latent Dirichlet allocation topic model (Blei, Ng, and Jordan, 2003) to find coherent segment boundaries. Such methods output segment boundaries and suggest lexical distribution associated with each segment. Generative models tend to perform well, but are less flexible than the similarity-based models when it comes to incorporating new kinds of information.

Globally informed models generally perform better, especially on more challenging datasets such as speech recordings, but they have – unsurprisingly – higher memory and CPU time requirements.

The *APS* algorithm described in this paper combines several desirable properties. It is unsupervised and, unlike most other segmenters, does not require specifying the desired number of segments as an input parameter. On each iteration it takes into account the information about a large portion of the docu-

ment (or all of it). Because *APS* operates on a precompiled matrix of pair-wise sentence similarities, it is easy to incorporate new kinds of information, such as synonymy or adjacency. It also provides some information as to what the segment is about, because each segment is associated with a segment centre.

3 Factor graphs and affinity propagation for clustering

3.1 Factor graphs and the max-sum algorithm

The *APS* algorithm is an instance of belief propagation on a cyclic factor graph. In order to explain the derivation of the algorithm, we will first briefly introduce factor graphs as a framework.

Many computational problems can be reduced to maximizing the value of a multi-variate function $F(x_1, \ldots, x_n)$ which can be approximated by a sum of simpler functions. In Equation 1, H is a set of discrete indices and f_h is a local function with arguments $X_h \subset \{x_1, \ldots, x_n\}$:

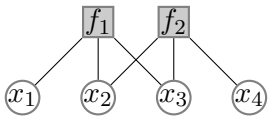
$$F(x_1, \ldots, x_n) = \sum_{h \in H} f_h(X_h) \tag{1}$$

Factor graphs offer a concise graphical representation for such problems. A global function F which can be decomposed into a sum of M local function f_h can be represented as a bi-partite graph with M function nodes and N variable nodes ($M = |H|$). Figure 1 shows an example of a factor graph for $F(x_1, x_2, x_3, x_4) = f_1(x_1, x_2, x_3) + f_2(x_2, x_3, x_4)$. The factor (or function) nodes are dark squares, the variable nodes are light circles.

The well-known max-sum algorithm (Bishop, 2006) seeks a configuration of variables which maximizes the objective function. It finds the maximum in acyclic factor graphs, but in graphs with cycles neither convergence nor optimality are guaranteed (Pearl, 1982). Yet in practice good approximations can be achieved. The max-sum algorithm amounts to propagating messages from function nodes to variable nodes and from variable nodes to function nodes. A message sent from a variable node x to a function node f is computed as a sum of the incoming messages from all neighbours of x other than f (the sum is computed for each possible value of x):

$$\mu_{x \to f} = \sum_{f' \in N(x) \setminus f} \mu_{f' \to x} \tag{2}$$

Figure 1: Factor graph for $F(x_1, x_2, x_3, x_4) = f_1(x_1, x_2, x_3) + f_2(x_2, x_3, x_4)$.



$N(x)$ is the set of all function nodes which are x's neighbours. The message reflects the evidence about the distribution of x from all functions which have x as an argument, except for the function corresponding to the receiving node f.

A message $\mu_{f \to x}$ from function f to variable x is computed as follows:

$$\mu_{f \to x} = \max_{N(f) \setminus x} (f(x_1, \ldots, x_m) + \sum_{x' \in N(f) \setminus x} \mu_{x' \to f}) \tag{3}$$

$N(f)$ is the set of all variable nodes which are f's neighbours. The message reflects the evidence about the distribution of x from function f and its neighbours other than x.

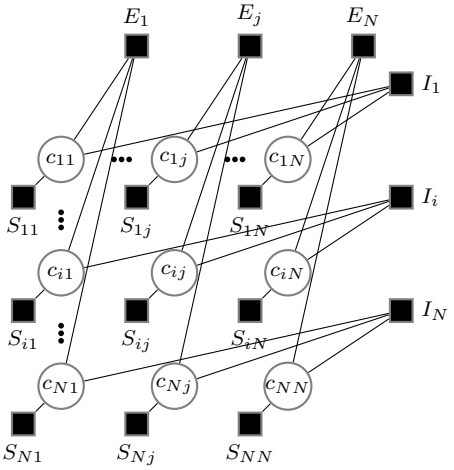
A common message-passing schedule on cyclic factor graphs is *flooding*: iteratively passing all variable-to-function messages, then all function-to-variable messages. Upon convergence, the summary message reflecting final beliefs about the maximizing configuration of variables is computed as a sum of all incoming function-to-variable messages.

3.2 Affinity Propagation

The *APS* algorithm described in this paper is a modification of the original Affinity Propagation algorithm intended for exemplar-based clustering (Frey and Dueck, 2007; Givoni and Frey, 2009). This section describes the binary variable formulation proposed by Givoni and Frey, and lays the groundwork for deriving the new update messages (Section 4).

Affinity Propagation for exemplar-based clustering is formulated as follows: to cluster N data points, one must specify a matrix of pairwise similarities $\{\mathcal{SIM}(i,j)\}_{i,j \in \{1,\ldots,N\}, i \neq j}$ and a set of self-similarities (so-called *preferences*) $\mathcal{SIM}(j,j)$ which reflect *a priori* beliefs in how likely each data point is to be selected as an exemplar. Preference values occupy the diagonal of the similarity matrix. The algorithm then assigns each data point to an exemplar so as to maximize net similarity – the sum of

Figure 2: Factor graph for affinity propagation.



similarities between all points and their respective exemplars; this is expressed by Equation 7. Figure 2 shows a schematic factor graph for this problem, with N^2 binary variables. $c_{ij} = 1$ iff point j is an exemplar for point i. Function nodes E_j enforce a *coherence constraint*: a data point cannot exemplify another point unless it is an exemplar for itself:

$$E_j(c_{1j}, \ldots, c_{Nj}) = \begin{cases} -\infty & \text{if } c_{jj} = 0 \ \wedge \ c_{ij} = 1 \\ & \text{for some } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

An I node encodes a *single-cluster constraint*: each data point must belong to exactly one exemplar – and therefore to one cluster:

$$I_i(c_{i1}, \ldots, c_{iN}) = \begin{cases} -\infty & \text{if } \sum_j c_{ij} \neq 1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

An S node encodes user-defined similarities between data-points and candidate exemplars ($\mathcal{SIM}(i,j)$ is the similarity between points i and j):

$$S_{ij}(c_{ij}) = \begin{cases} \mathcal{SIM}(i,j) & \text{if } c_{ij} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Equation 7 shows the objective function which we want to maximize: a sum of similarities between data points and their exemplars, subject to the two constraints (coherence and single-cluster per point).

$$S(c_{11}, \ldots, c_{NN}) = \sum_{i,j} S_{i,j}(c_{ij}) + \sum_i I_i(c_{i1}, \ldots, c_{iN}) + \sum_j E_j(c_{1j}, \ldots, c_{Nj}) \tag{7}$$

According to Equation 3, the computation of a single factor-to-variable message involves maximizing over 2^n configurations. E and I, however, are binary constraints and evaluate to $-\infty$ for most configurations. This drastically reduces the number of configurations which can maximize the message values. Given this simple fact, Givoni and Frey (2009) show how to reduce the necessary update messages to only two types of scalar ones: *availabilities* (α) and *responsibilities* (ρ).[2]

A responsibility message ρ_{ij}, sent from a variable node c_{ij} to function node E_j, reflects the evidence of how likely j is to be an exemplar for i given all other potential exemplars:

$$\rho_{ij} = \mathcal{SIM}(i,j) - \max_{k \neq j}(\mathcal{SIM}(i,k) + \alpha_{ik}) \tag{8}$$

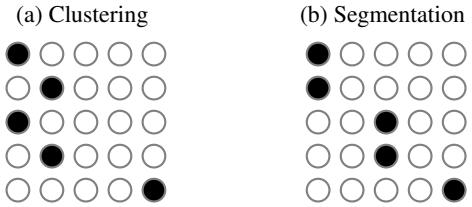
An availability message α_{ij}, sent from a function node E_j to a variable node c_{ij}, reflects how likely point j is to be an exemplar for i given the evidence from all other data points:

$$\alpha_{ij} = \begin{cases} \sum_{k \neq j} \max[\rho_{kj}, 0] & \text{if } i = j \\ \min[0, \rho_{jj} + \sum_{k \notin \{i,j\}} \max[\rho_{kj}, 0]] & \text{if } i \neq j \end{cases} \tag{9}$$

Let $\gamma_{ij}(l)$ be the message value corresponding to setting variable c_{ij} to l, $l \in \{0, 1\}$. Instead of sending two-valued messages (corresponding to the two possible values of the binary variables), we can send the difference for the two possible configurations: $\gamma_{ij} = \gamma_{ij}(1) - \gamma_{ij}(0)$ – effectively, a log-likelihood ratio.

[2]Normally, each iteration of the algorithm sends five types of two-valued messages: to and from functions E and I and a message from functions S. Fortunately, the messages sent to and from E factors to the variable nodes subsume the three other message types and it is not necessary to compute them explicitly. See (Givoni and Frey, 2009, p.195) for details.

Figure 3: Examples of valid configuration of hidden variables $\{c_{ij}\}$ for clustering and segmentation.



The algorithm converges when the set of points labelled as exemplars remains unchanged for a predetermined number of iterations. When the algorithm terminates, messages to each variable are added together. A positive final message indicates that the most likely value of a variable c_{ij} is 1 (point j is an exemplar for i), a negative message indicates that it is 0 (j is not an exemplar for i).

4 Affinity Propagation for Segmentation

This section explains how we adapt the Affinity Propagation clustering algorithm to segmentation.

In this setting, sentences are data points and we refer to exemplars as *segment centres*. Given a document, we want to assign each sentence to a segment centre so as to maximize net similarity.

The new formulation relies on the same underlying factor graph (Figure 2). A binary variable node c_{ij} is set to 1 iff sentence j is the segment centre for sentence i. When clustering is the objective, a cluster may consist of points coming from anywhere in the data sequence. When segmentation is the objective, a segment must consist of a solid block of points around the segment centre. Figure 3 shows, for a toy problem with 5 data points, possible valid configurations of variables $\{c_{ij}\}$ for clustering (3a) and for segmentation (3b).

To formalize this new linearity requirement, we elaborate Equation 4 into Equation 10. E_j evaluates to $-\infty$ in three cases. Case 1 is the original coherence constraint. Case 2 states that no point k may be in the segment with a centre is j, if k lies before the start of the segment (the sequence $c_{(s-1)j} = 0$, $c_{sj} = 1$ necessarily corresponds to the start of the segment). Case 3 handles analogously the end of the segment.

$$E_j = \begin{cases} -\infty & \textbf{1}.\ \text{if } c_{jj} = 0 \ \wedge\ c_{ij} = 1 \text{ for some } i \neq j \\ & \textbf{2}.\ \text{if } c_{jj} = 1 \ \wedge\ c_{sj} = 1 \ \wedge\ c_{(s-1)j} = 0 \\ & \wedge\, c_{kj} = 1 \text{ for some } s < j,\ k < s-1 \\ & \textbf{3}.\ \text{if } c_{jj} = 1 \ \wedge\ c_{ej} = 1 \ \wedge\ c_{(e+1)j} = 0 \\ & \wedge\, c_{kj} = 1 \text{ for some } e > j,\ k > e+1 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The *E* function nodes are the only changed part of the factor graph, so we only must re-derive α messages (availabilities) sent from factors *E* to variable nodes. A function-to-variable message is computed as shown in Equation 11 (elaborated Equation 3), and the only incoming messages to E nodes are responsibilities (ρ messages):

$$\mu_{f\to x} = \max_{N(f)\setminus x}(f(x_1,\ldots,x_m) + \sum_{x'\in N(f)\setminus x} \mu_{x'\to f}) = \tag{11}$$

$$\max_{c_{ij},\, i\neq j}((E_j(c_{1j},\ldots,c_{Nj}) + \sum_{c_{ij},\, i\neq j} \rho_{ij}(c_{ij})))$$

We need to compute the message values for the two possible settings of binary variables – denoted as $\alpha_{ij}(1)$ and $\alpha_{ij}(0)$ – and propagate the difference $\alpha_{ij} = \alpha_{ij}(1)$ - $\alpha_{ij}(0)$.

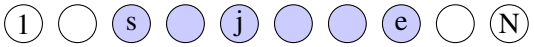
Consider the case of factor E_j sending an α message to the variable node c_{jj} (i.e., $i = j$). If $c_{jj} = 0$ then point j is not its own segment centre and the only valid configuration is to set all other c_{ij} to 0:

$$\alpha_{jj}(0) = \max_{c_{ij}, i\neq j}(E_j(c_{1j},\ldots,c_{Nj}) + \sum_{c_{ij}, i\neq j} \rho_{ij}(c_{ij})) \tag{12}$$

$$= \sum_{i\neq j} \rho_{ij}(0)$$

To compute $\alpha_{ij}(1)$ (point j is its own segment centre), we only must maximize over configurations which will not correspond to cases 2 and 3 in Equation 10 (other assignments are trivially non-optimal because they would evaluate E_j to $-\infty$). Let the start of a segment be s, $1 \leq s < j$ and the end of the segment be e, $j+1 < e \leq N$. We only need to consider configurations such that all points between s and e are in the segment while all others are not.

The following picture shows a valid configuration.[3]



To compute the message $\alpha_{ij}(1), i = j$, we have:

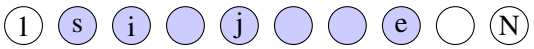
$$\alpha_{jj}(1) = \max_{s=1}^{j}[\sum_{k=1}^{s-1}\rho_{kj}(0) + \sum_{k=s}^{j-1}\rho_{kj}(1)] + \quad (13)$$
$$\max_{e=j}^{N}[\sum_{k=j+1}^{e}\rho_{kj}(1) + \sum_{k=e+1}^{N}\rho_{kj}(0)]$$

Subtracting Equation 12 from Equation 13, we get:

$$\alpha_{jj} = \alpha_{jj}(1) - \alpha_{jj}(0) = \quad (14)$$
$$\max_{s=1}^{j}(\sum_{k=s}^{j-1}\rho_{kj}) + \max_{e=j}^{N}(\sum_{k=j+1}^{e}\rho_{kj})$$

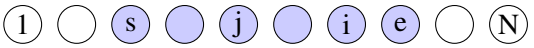
Now, consider the case of factor E_j sending an α message to a variable node c_{ij} other than segment exemplar j (i.e., $i \neq j$). Two subcases are possible: point i may lie before the segment centre j ($i < j$), or it may lie after the segment centre ($i > j$).

The configurations which may maximize $\alpha_{ij}(1)$ (the message value for setting the hidden variable to 1) necessarily conform to two conditions: point j is labelled as a segment centre (c_{jj} = 1) and all points lying between i and j are in the segment. This corresponds to Equation 15 for $i < j$ and to Equation 16 for $i > j$. Pictorial examples of corresponding valid configurations precede the equations.



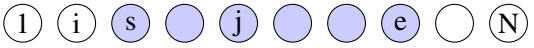
$$\alpha_{ij,\, i<j}(1) = \max_{s=1}^{i}[\sum_{k=1}^{s-1}\rho_{kj}(0) + \sum_{k=s}^{i-1}\rho_{kj}(1)] + \quad (15)$$
$$\sum_{k=i+1}^{j}\rho_{kj}(1) + \max_{e=j}^{N}[\sum_{k=j+1}^{e}\rho_{kj}(1) + \sum_{k=e+1}^{N}\rho_{kj}(0)]$$

[3]Variables c_{ij} set to 1 are shown as shaded circles, to 0 – as white circles. Normally, variables form a column in the factor graph; we transpose them to save space.

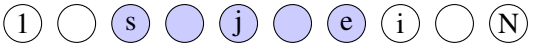


$$\alpha_{ij,\, i>j}(1) = \max_{s=1}^{j}[\sum_{k=1}^{s-1}\rho_{kj}(0) + \sum_{k=s}^{j-1}\rho_{kj}(1)] + \quad (16)$$
$$\sum_{k=j}^{i-1}\rho_{kj}(1) + \max_{e=i}^{N}[\sum_{k=i+1}^{e}\rho_{kj}(1) + \sum_{k=e+1}^{N}\rho_{kj}(0)]$$

To compute the message value for setting the hidden variable c_{ij} to 0, we again distinguish between $i < j$ and $i > j$ and consider whether c_{jj} = 1 or c_{jj} = 0 (point j is / is not a segment centre). For c_{jj} = 0 the only optimal configuration is c_{ij} = 0 for all $i \neq j$. For c_{jj} = 1 the set of possible optimal configurations is determined by the position of point i with respect to point j. Following the same logic as in the previous cases we get Equation 17 for $i < j$ and Equation 18 for $i > j$.



$$\alpha_{ij}(0) = \max(\sum_{k\notin i,j}\rho_{kj}(0), \quad (17)$$
$$\sum_{k=1}^{i-1}\rho_{kj}(0) + \max_{s=i+1}^{j}[\sum_{k=i+1}^{s-1}\rho_{kj}(0) + \sum_{k=s}^{j-1}\rho_{kj}(1)] +$$
$$\rho_{jj}(1) + \max_{e=j}^{N}[\sum_{k=j+1}^{e}\rho_{kj}(1) + \sum_{k=e+1}^{N}\rho_{kj}(0)])$$



$$\alpha_{ij}(0) = \max(\sum_{k\notin i,j}\rho_{kj}(0), \quad (18)$$
$$\max_{s=1}^{j}[\sum_{k=1}^{s-1}\rho_{kj}(0) + \sum_{k=s}^{j-1}\rho_{kj}(1)] +$$
$$\rho_{jj}(1) + \max_{e=j}^{i-1}[\sum_{k=j+1}^{e}\rho_{kj}(1) + \sum_{k=e+1}^{i-1}\rho_{kj}(0)]$$
$$\sum_{k=i+1}^{N}\rho_{kj}(0))$$

Due to space constraints, we will omit the details of subtracting Equation 17 from 15 and Equation 18 from 16. The final update rules for both $i < j$ and

Algorithm 1 Affinity Propagation for Segmentation

1: **input**: 1) a set of pairwise similarities $\{\mathcal{SIM}(i,j)\}_{(i,j)\in\{1,\ldots,N\}^2}$, $\mathcal{SIM}(i,j) \in \mathbb{R}$; 2) a set of preferences (self-similarities) $\{\mathcal{SIM}(i,i)\}_{i\in\{1,\ldots,N\}}$ indicating *a priori* likelihood of point i being a segment centre

2: **initialization**: $\forall i,j : \alpha_{ij} = 0$ (set all availabilities to 0)

3: **repeat**

4: iteratively update responsibilities (ρ) and availabilities (α)

5:
$$\forall i,j : \rho_{ij} = \mathcal{SIM}(i,j) + \max_{k\neq j}(\mathcal{SIM}(i,k) - \alpha_{ik})$$

6:
$$\forall i,j : \alpha_{ij} = \begin{cases} \max_{s=1}^{j}(\sum_{k=s}^{j-1}\rho_{kj}) + \max_{e=j}^{N}(\sum_{k=j+1}^{e}\rho_{kj}) & \text{if } i=j \\ \min[\max_{s=1}^{i}\sum_{k=s}^{i-1}\rho_{kj} + \sum_{k=i+1}^{j}\rho_{kj} + \max_{e=j}^{N}\sum_{k=j+1}^{e}\rho_{kj}, & \\ \quad \max_{s=1}^{i}\sum_{k=s}^{i-1}\rho_{kj} + \min_{s=i+1}^{j}\sum_{k=i+1}^{s-1}\rho_{kj}] & \text{if } i<j \\ \min[\max_{s=1}^{j}\sum_{k=s}^{j-1}\rho_{kj} + \sum_{k=j}^{i-1}\rho_{kj} + \max_{e=i}^{N}\sum_{k=i+1}^{e}\rho_{kj}, & \\ \quad \min_{e=j}^{i-1}\sum_{k=e+1}^{i-1}\rho_{kj} + \max_{e=i}^{N}\sum_{k=i+1}^{e}\rho_{kj}] & \text{if } i>j \end{cases}$$

7: **until** convergence

8: compute the final configuration of variables: $\forall i, j$ j is the exemplar for i iff $\rho_{ij} + \alpha_{ij} > 0$

9: **output**: exemplar assignments

$i > j$ appear in Algorithm 1, where we summarize the whole process.

The equations look cumbersome but they are trivial to compute. Every summand corresponds to finding the most likely start or end of the segment, taking into account *fixed* information. When computing messages for any given sender node, we can remember the maximizing values for neighbouring recipient nodes. For example, after computing the availability message from factor E_j to c_{ij}, we must only consider one more responsibility value when computing the message from E_j to variable $c_{(i+1)j}$. The cost of computing a message is thus negligible.

When the matrix is fully specified, each iteration requires passing $2N^2$ messages, so the algorithm runs in $O(N^2)$ time and requires $O(N^2)$ memory (to store the similarities, the availabilities and the responsibilities). When performing segmentation, however, the user generally has some idea about the average or maximum segment length. In such more realistic cases, the input matrix of similarities is sparse – it is constructed by sliding a window of size M. M usually needs to be at least twice the maximum segment length or thrice the average segment length. Each iteration, then, involves sending $2MN$ messages and the storage requirements are also $O(MN)$.

As is common in loopy belief propagation algorithms, both availability and responsibility messages are dampened to avoid overshooting and oscillating. The dampening factor is λ where $0.5 \leq \lambda < 1$.

$$newMsg = \lambda * oldMsg + (1-\lambda)newMsg \quad (19)$$

The *APS* algorithm is unsupervised. It only benefits

from a small development set to fine-tune a few parameters: preference values and the dampening factor. *APS* does not require (nor allow) specifying the number of segments beforehand. The granularity of segmentation is adjusted through preference values; this reflect how likely each sentence is to be selected as a segment centre. (This translates into the cost of adding a segment.)

Because each message only requires the knowledge about one column or row of the matrix, the algorithm can be easily parallelized.

5 Experimental Setting

Datasets. We evaluate the performance of the *APS* algorithm on three datasets. The first, compiled by Malioutov and Barzilay (2006), consists of manually transcribed and segmented lectures on Artificial Intelligence, 3 development files and 19 test files. The second dataset consists of 227 chapters from medical textbooks (Eisenstein and Barzilay, 2008), 5 of which we use for development. In this dataset the gold standard segment boundaries correspond to section breaks specified by the authors. The third dataset consists of 85 works of fiction downloaded from Project Gutenberg, 3 of which are used for development. The segment boundaries correspond to chapter breaks or to breaks between individual stories. They were inserted automatically using HTML markup in the downloaded files.

The datasets exhibit different characteristics. The lecture dataset and the fiction dataset are challenging because they are less cohesive than medical textbooks. The textbooks are cognitively more difficult to process and the authors rely on repetition of terminology to facilitate comprehension. Since lexical repetition is the main source of information for text segmentation, we expect a higher performance on this dataset. Transcribed speech, on the other hand, is considerably less cohesive. The lecturer makes an effort to speak in "plain language" and to be comprehensible, relying less on terminology. The use of pronouns is very common, as is the use of examples.

Repeated use of the same words is also uncommon in fiction. In addition, the dataset was compiled automatically using HTML markup. The markup is not always reliable and occasionally the e-book proofreaders skip it altogether, which potentially adds noise to the dataset.

Baselines. We compare the performance of *APS* with that of two state-of-the-art segmenters: the Minimum Cut segmenter (Malioutov and Barzilay, 2006) and the Bayesian segmenter (Eisenstein and Barzilay, 2008). The authors have made Java implementations publicly available. For the Minimum Cut segmenter, we select the best parameters using the script included with that distribution. The Bayesian segmenter automatically estimates all necessary parameters from the data.

Preprocessing and the choice of similarity metric. As described in Section 4, the *APS* algorithm takes as inputs a matrix of pairwise similarities between sentences in the document and also, for each sentence, a preference value.

This paper focuses on comparing globally informed segmentation algorithms, and leaves for future work the exploration of best similarity metrics. To allow fair comparison, then, we use the same metric as the Minimum Cut segmenter, cosine similarity. Each sentence is represented as a vector of token-type frequencies. Following (Malioutov and Barzilay, 2006), the frequency vectors are smoothed by adding counts of words from the adjacent sentences and then weighted using a tf.idf metric (for details, see *ibid.*) The similarity between sentence vectors s_1 and s_2 is computed as follows:

$$cos(s_1, s_2) = \frac{s_1 \bullet s_2}{||s_1|| \times ||s_2||} \tag{20}$$

The representation used by the Bayesian segmenter is too different to be incorporated into our model directly, but ultimately it is based on the distribution of unigrams in documents. This is close enough to our representation to allow fair comparison.

The fiction dataset consists of books: novels or collections of short stories. Fiction is known to exhibit less lexical cohesion. That is why – when working on this dataset – we work at the paragraph level: the similarity is measured not between sentences but between paragraphs. We use this representation with all three segmenters.

All parameters have been fine-tuned on the development portions of the datasets. For *APS* algorithm *per se* we needed to set three parameters: the size of the sliding window for similarity computations, the dampening factor λ and the preference values. The

	BayesSeg	MinCutSeg	*APS*
AI	0.443	0.437	**0.404**
Clinical	**0.353**	0.382	0.371
Fiction	0.377	0.381	**0.350**

Table 1: Results of segmenting the three datasets using the Bayesian segmenter, the Minimum Cut segmenter and *APS*.

parameters for the similarity metric (best variation of tf.idf, the window size and the decay factor for smoothing) were set using the script provided in the Minimum Cut segmenter's distribution.

Evaluation metric. We have measured the performance of the segmenters with the WindowDiff metric (Pevzner and Hearst, 2002). It is computed by sliding a window through reference and through segmentation output and, at each window position, comparing the number of reference breaks to the number of breaks inserted by the segmenter (hypothetical breaks). It is a penalty measure which reports the number of windows where the reference and hypothetical breaks do not match, normalized by the total number of windows. In Equation 21, ref and hyp denote the number of reference and hypothetical segment breaks within a window.

$$winDiff = \frac{1}{N-k} \sum_{i=1}^{N-k} (|ref - hyp| \neq 0) \quad (21)$$

6 Experimental Results and Discussion

Table 1 compares the performance of the three segmenters using WindowDiff values. On the lecture and fiction datasets, the *APS* segmenter outperforms the others by a small margin, around 8% over the better of the two. It is second-best on the clinical textbook dataset. According to a one-tailed paired t-test with 95% confidence cut-off, the improvement is statistically significant only on the fiction dataset. All datasets are challenging and the baselines are very competitive, so drawing definitive conclusions is difficult. Still, we can be fairly confident that *APS* performs at least as well as the other two segmenters. It also has certain advantages.

One important difference between *APS* and the other segmenters is that *APS* does not require the number of segments as an input parameter. This is very helpful, because such information is generally unavailable in any realistic deployment setting. The parameters are fine-tuned to maximize WindowDiff values, so this results in high-precision, low-recall segment assignments; that is because WindowDiff favours missing boundaries over near-hits.

APS also outputs segment centres, thus providing some information about a segment's topic. We have not evaluated how descriptive the segment centres are; this is left for future work.

APS performs slightly better than the other segmenters but not by much. We hypothesize that one of the reasons is that *APS* relies on the presence of descriptive segment centres which are not necessarily present for large, coarse-grained segments such as chapters in novels. It is possible for *APS* to have an advantage performing fine-grained segmentation.

7 Conclusions and Future Work

In this paper we have presented *APS* – a new algorithm for linear text segmentation. *APS* takes into account the global structure of the document and outputs segment boundaries and segment centres. It scales linearly in the number of input sentences, performs competitively with the state-of-the-art and is easy to implement. We also provide a Java implementation of the *APS* segmenter.

We consider two main directions for future work: using more informative similarity metrics and making the process of segmentation hierarchical. We chose to use cosine similarity primarily to allow fair comparison and to judge the algorithm itself, in isolation from the information it uses. Cosine similarity is a very simple metric which cannot provide an adequate picture of topic fluctuations in documents. It is likely that dictionary-based or corpus-based similarity measures would yield a major improvement in performance.

Reliance on descriptive segment centres may handicap *APS*'s performance when looking for coarse-grained segments. One possible remedy is to look for shorter segments first and then merge them. One can also modify the algorithm to perform hierarchical segmentation: consider net similarity with low-level segment centres as well as with high-level ones. We plan to explore both possibilities.

Acknowledgements

We thank Inmar Givoni for explaining the details of binary Affinity Propagation and for commenting on our early ideas in this project. Many thanks to Yongyi Mao for a helpful discussion on the use Affinity Propagation for text segmentation.

References

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Blei, David and Pedro Moreno. 2001. Topic Segmentation with an Aspect Hidden Markov Model. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 343–348. ACM Press.

Blei, David M., Andrew Ng, and Michael Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Choi, Freddy Y. Y. 2000. Advances in Domain Independent Linear Text Segmentation. In *Proceedings of NAACL*, pages 26–33.

Eisenstein, Jacob and Regina Barzilay. 2008. Bayesian Unsupervised Topic Segmentation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 334–343, Honolulu, Hawaii, October.

Frey, Brendan J. and Delbert Dueck. 2007. Clustering by Passing Messages Between Data Points. *Science*, 315:972–976.

Givoni, Inmar E. and Brendan J. Frey. 2009. A Binary Variable Model for Affinity Propagation. *Neural Computation*, 21:1589–1600.

Haghighi, Aria and Lucy Vanderwende. 2009. Exploring Content Models for Multi-Document Summarization. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 362–370, Boulder, Colorado, June.

Hearst, Marti A. 1997. TextTiling: segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23:33–64, March.

Kschischang, Frank R., Brendan J. Frey, and Hans-A Loeliger. 2001. Factor graphs and the sum-product algorithm. In *IEEE Transactions on Information Theory, Vol 47, No 2*, pages 498–519, February.

Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-01*, pages 282–289.

Malioutov, Igor and Regina Barzilay. 2006. Minimum Cut Model for Spoken Lecture Segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 25–32, Sydney, Australia, July.

Misra, Hemant, François Yvon, Joemon M. Jose, and Olivier Cappé. 2009. Text segmentation via topic modeling: an analytical study. In *18th ACM Conference on Information and Knowledge Management*, pages 1553–1556.

Oh, Hyo-Jung, Sung Hyon Myaeng, and Myung-Gil Jang. 2007. Semantic passage segmentation based on sentence topics for question answering. *Information Sciences, an International Journal*, 177:3696–3717, September.

Pearl, Judea. 1982. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the American Association of Artificial Intelligence National Conference on AI*, pages 133–136, Pittsburgh, PA.

Pevzner, Lev and Marti A. Hearst. 2002. A Critique and Improvement of an Evaluation Metric for Text Segmentation. *Computational Linguistics*, 28(1):19–36.

Stoyanov, Veselin and Claire Cardie. 2008. Topic identification for fine-grained opinion analysis. In *COLING '08 Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, pages 817–824.

Youmans, Gilbert. 1991. A new tool for discourse analysis: The vocabulary-management profile. *Language*, 67(4):763–789.