`ffgrep`:
Scalable Approximate String Matching

Dean Knox*

Preliminary draft: 10 September 2019

Abstract

Approximate substring searching is a common but computationally demanding task in bioinformatics and text analysis. We present a new approach that recasts string search as a multiple convolution problem, then exploits highly efficient fast Fourier convolution techniques. This approach, which we call `ffgrep`, computes and caches the spectra of a target corpora, drastically reducing the cost of subsequent searches. Like other approaches, this algorithm is embarrassingly parallelizable; unlike other approaches, it is capable of operating on not only raw strings, but also word embeddings. `ffgrep` is applied to an original corpus of imperfect automatic transcriptions of campaign speeches in the 2012 U.S. presidential election. We contrast our approach with `agrep`, an industry-standard meta-algorithm that selects the optimal member from a number of highly optimized approximate string matching algorithms. Searching for approximate recurrences of a manually curated set of candidate catchphrases, we show that `ffgrep` speeds computation by up to a factor of 60x in typical settings, with increasing gains as alignments grow longer or more complex. Moreover, these computational gains come at little cost in performance. Taking `agrep` search results as ground truth, over a wide range of `agrep` parameters, we show that `ffgrep` is capable of recovering highly similar results with accuracies exceeding 0.94 and F_1 of 0.84–0.9. Finally, we demonstrate how efficient substring matching enables new substantive research by identifying candidate catchphrases without human supervision. By rapidly computing and organizing 90 billion pairwise string comparisons, our proposed method automatically learns that the phrases "kick children off of Head Start or eliminate health insurance for the poor" and "kick students are [*sic*] financial aid or get rid of funding for Planned Parenthood or eliminate health care for millions on Medicaid"—along with 32 other campaign appeals—all map onto a single recurring theme, President Barack Obama's critique of a proposed Medicare reform.

*Assistant Professor, Princeton University, Fisher Hall, Princeton, NJ 08544; http://www.dcknox.com/

1 The Problem

We begin by formulating the problem. Consider a corpus of variable-length "target" documents in which the i-th target, T_i, is given by the J_i-word sequence $(T_{i,1}, \ldots, T_{i,J_i})$, of word indices in a vocabulary $\mathcal{V}$. Within this corpus, we wish to identify all near-recurrences of a K-length "pattern," $P = (P_1, \ldots, P_K)$, where P_k again indexes words in $\mathcal{V}$. Specifically, the goal is to identify (i, j) pairs that indicate a subregion of similarity to P in document i beginning at offset j.

Various substring alignment algorithms differ in their exact operationalization of "similarity;" we clarify ours in Section 2. All agree that two (sub)strings containing an identical sequence of words are similar. But because words in $\mathcal{V}$ are themselves composed of letter sequences, not disparate, two strings may also be similar while containing no identical words. Conceptually, differences can manifest through insertions of letters, deletions, substitutions, or transpositions of letters or spaces (which split or merge words as a result) [Dam64]. However, the semantic distance between two strings, which is typically unmeasured, varies nonlinearly with not only the count of these mutations [Lev66] but also their location.

2 Yet Another String Distance

A wide range of string distances have been proposed for quantifying general and domain-specific similarity, including the classical Levenshtein edit distance, simplified variants [Ham50, NW70], and numerous modifications, generalizations, and alternative approaches [AAL+97, KS95, Tic84, Ukk92]. For a review of this extensive literature, we refer the reader to [Nav01].

Here, we propose yet another.

We encode each string as a word-letter matrix in which the k-th row contains frequencies for each of the L letters—e.g., $L = 4$ in genomics, $L = 26$ in English. The result is a lossy representation of the original string that discards information about letter ordering within words. This representation of the pattern is denoted $\underset{K \times L}{\boldsymbol{P}}$, and target i is $\underset{J_i \times L}{\boldsymbol{T}_i}$. An example is given in Table 1. It is worth noting that word-embedding matrices may be substituted for word-letter matrices with no further modification of the algorithm proposed below.

The similarity between two K-word sequences, $\boldsymbol{P}$ and $\boldsymbol{Q}$, is then operationalized as

$$\mathcal{S}(\boldsymbol{P}, \boldsymbol{Q}) = \frac{\sum_{k=1}^{K} \sum_{\ell=1}^{L} \tilde{p}_{k,\ell}\ \tilde{q}_{k,\ell}}{||\tilde{\boldsymbol{P}}||_F\ ||\tilde{\boldsymbol{Q}}||_F} \tag{1}$$

where $\tilde{\boldsymbol{A}} = [a_{k\ell} - \bar{a}_\ell]$ indicates the column-demeaned transformation of $\boldsymbol{A}$, $\tilde{a}_{k,\ell}$ is the (k, ℓ)-th element of $\tilde{\boldsymbol{A}}$, and $||\boldsymbol{A}||_F = \sqrt{\sum_k \sum_\ell a_{k,\ell}^2}$ is the Frobenius norm. The chief advantages of this metric are that it is extremely fast to compute with Algorithm 1 (as we show in Section 5.1) and that it does not perform very badly (Section 5.2).

In intuitive terms, $||\tilde{\boldsymbol{P}}||_F^2$ is proportional to the pattern's total variance, or the sum of letter-specific variances, and the numerator is proportional to $\sum_{\ell=1}^{L} \mathrm{Cov}\,(P_\ell, Q_\ell)$, where P_ℓ is the sequence of counts for letter ℓ. Thus, when $L = 1$, Equation 1 yields the correlation coefficient. For lack of imagination, we refer to $1 - \mathcal{S}(\boldsymbol{P}, \boldsymbol{Q})$ as the string correlation distance. $\mathcal{S}(\cdot, \cdot)$ is symmetric, bounded in $[-1, 1]$, and has the property $\mathcal{S}(\boldsymbol{P}, \boldsymbol{P}) = 1$.

Table 1: **Word-letter matrix.** Excerpted words from a President Barack Obama's campaign speech during the 2012 presidential election are represented using their letter counts. Word-letter matrix representations are used for approximate string alignment in `ffgrep`.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
we've					2																	1	1			
doubled		1		2	1							1			1						1					
the					1			1												1						
amount	1												1	1	1					1	1					
of						1									1											
renewable	1	1			3							1		1				1					1			
energy					2		1							1				1								1
that	1							1												2						
we					1																		1			
generate	1				3		1							1				1		1						

3 The Algorithm

Approximate string search involves examining all target documents i and candidate offsets j within each document. Figure 1 illustrates how this sequence can be obtained by sweeping a pattern over a target document. At each position, the similarity measure is computed, producing the alignment sequence $\big[\mathcal{S}(\boldsymbol{P}, \boldsymbol{T}_{i,1:K}), \ldots, \mathcal{S}(\boldsymbol{P}, \boldsymbol{T}_{i,(J_i-K+1):J_i})\big]$. A "hit," or high-quality alignment, is a position in the target document that produces a spike in this similarity sequence. In this section, we show how this apparently intensive task can be reformulated using highly efficient rolling sums and Fourier transforms. We begin by examining the elements of Equation 1.

First, observe that $||\tilde{\boldsymbol{T}}_{i,1:K}||_F$ is the grand sum of a row subset of $[\tilde{t}^2_{i,j}]$. Corresponding values must be computed at every offset in document i to produce the sequence $\Big[||\tilde{\boldsymbol{T}}_{i,1:K}||_F, \ldots, ||\tilde{\boldsymbol{T}}_{i,(J_i-K+1):J_i}||_F\Big]$, which is simply a rolling windowed sum on $[\tilde{t}^2_{i,j}]\mathbf{1}$. Computation of $||\tilde{\boldsymbol{P}}||_F$ is even more straightforward.

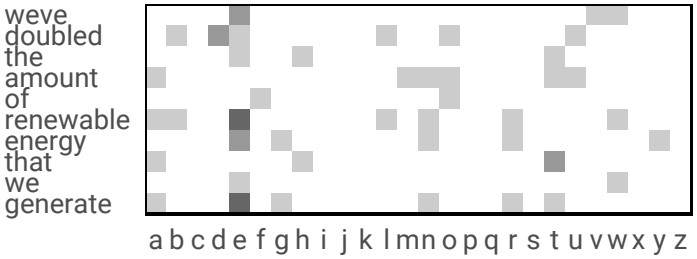
Next, we observe that the numerator, $\sum_{k=1}^{K}\sum_{\ell=1}^{L}\tilde{p}_{k,\ell}\,\tilde{t}_{i,j+k-1,\ell}$, can be rewritten as $\sum_{k=1}^{K}\sum_{\ell=1}^{L}p_{k,\ell}\,t_{i,j+k-1,\ell} - \sum_{k=1}^{K}\sum_{\ell=1}^{L}\bar{p}_\ell\,\bar{t}_{i,j,\ell}$, where $\bar{p}_\ell$ is the mean of the pattern's ℓ-th column and $\bar{t}_{i,j,\ell}$ is the mean count of letter ℓ in the K words starting at offset j in target i. The latter term can be simultaneously evaluated for all offsets as follows: Compute the rolling column means of $\boldsymbol{T}_i$, forming $\underset{J_i\times L}{\bar{\boldsymbol{T}}_i} = [\bar{t}_{i,j,\ell}]$, then take its matrix product with the vector $[\bar{p}_\ell]$.

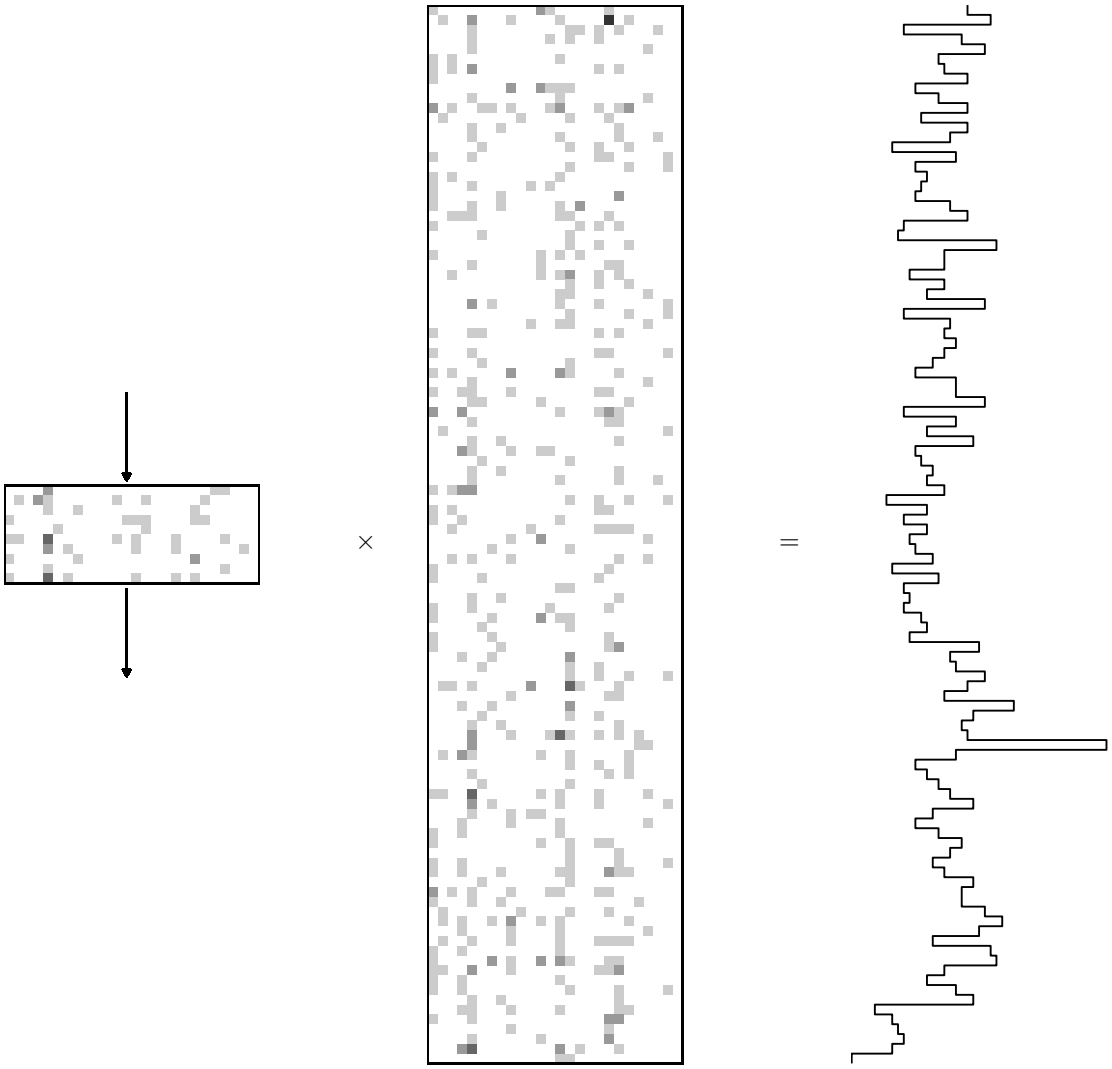
Finally, we are left with the term $\sum_{k=1}^{K}\sum_{\ell=1}^{L}p_{k,\ell}\;t_{i,j+k-1,\ell}$. Consider the contribution of a single letter, $x_{i,j,\ell} = \sum_{k=1}^{K}p_{k,\ell}\;t_{i,j+k-1,\ell}$. Evaluating this expression at every possible offset in the target, from $j=1$ to J_i, is computationally demanding. However, the resulting vector, $[x_{i,1,\ell}, \ldots, x_{i,J_i,\ell}]$, is the convolution $P_\ell * T_{i,\ell}$. It is well-known that the Fourier convolution theorem offers a drastically more efficient approach for solving such problems. Briefly, the theorem states that $P_\ell * T_{i,\ell} = \mathcal{F}^{-1}\big(\mathcal{F}(P_\ell)\odot\mathcal{F}(T_{i,\ell})\big)$, where $\mathcal{F}$ is the Fourier transform, $\mathcal{F}^{-1}$ is the inverse transform, and $\odot$ denotes the elementwise product. Thus, $\sum_{\ell=1}^{L}\mathcal{F}^{-1}\big(\mathcal{F}(P_\ell)\odot\mathcal{F}(T_{i,\ell})\big)$ completes the rolling similarity score. By linearity of the Fourier transform, this can be rewritten $\mathcal{F}^{-1}\left(\sum_{\ell=1}^{L}\mathcal{F}(P_\ell)\odot\mathcal{F}(T_{i,\ell})\right)$, reducing complexity of the inverse step by an additional factor of L. Moreover, because the goal of approximate string matching is to identify sharp peaks in the similarity sequence, a sparse Fourier transform [HIKP12] in the inverse step has the potential to reduce

computation time further. We do not explore sparsity-based optimizations here.

To identify approximate alignments, the resulting similarity sequence is thresholded. Additional details of the implementation are discussed in Algorithm 1. Among other steps, we zero-pad the pattern to a convenient length, then use the overlap-save method to cut targets into smaller batches of the same length. Target batches are also zero-padded to avoid circular convolution. After computing the Fourier transforms of the pattern and each batch, the target batch spectra are cached to accelerate subsequent searches against the same targets.

Figure 1: **Convolution of text sequences.** The top panel depicts a word-letter matrix, $\boldsymbol{P}$, for a single pattern: "we've doubled the amount of renewable energy that we generate," a quote from an Obama rally in Madison, WI. The bottom-left panel illustrates how this pattern is swept over a target document, $\boldsymbol{T}_i$, an earlier speech in West Palm Beach, FL (bottom middle). At offset j, the elementwise product with $\boldsymbol{T}_{i,(j-K+1):j_i}$ is taken and summed. This is repeated from $j = 1$ to target length J_i, and the sequence of resulting sums—the convolution—is plotted on the bottom right. Appropriate scaling yields the desired sequence of correlation similarities. The peak successfully identifies the previous usage of a similar phrase, "we've doubled our use of renewable energy like wind and..." from an earlier rally in West Palm Beach. Section 3.1 explores word blurring and gap smoothing, two modifications that improve the robustness of `ffgrep` to various string mutations.





3.1 Extensions

The proposed string distance function assigns zero loss to transpositions of letters within words. However, in the form described above, correlation distance is highly sensitive to word transposition, which often occurs in scenarios that users of fuzzy string matching hope to detect, such as paraphrasing. When $\boldsymbol{P}$ is compared to the mutated $\boldsymbol{P'}$ with words k and $k+1$ transposed, the loss is given by

$$1 - \mathcal{S}(\boldsymbol{P}, \boldsymbol{P'}) = \frac{\sum_{\ell=1}^{L} \left(p_{k,\ell} - p_{k+1,\ell}\right)^2}{||\tilde{\boldsymbol{P}}||_F^2} \tag{2}$$

which can be as large as $\frac{\sum_{\ell=1}^{L} p_{k,\ell}^2 + p_{k+1,\ell}^2}{\sum_{k'=1}^{K} \sum_{\ell=1}^{L} p_{k',\ell}^2}$ or roughly $2/K$. To address this issue, we employ *word blurring*, in which elements of the target document's word-letter matrix, $\boldsymbol{T}_i$, are replaced with a weighted average of nearby elements in the same column to produce the word-blurred matrix $\boldsymbol{T}_i'$. For example, a triangular blur kernel of radius $\beta = 2$ results in $T'_{ij\ell} = \frac{1}{4} t_{i,j-1,\ell} + \frac{1}{2} t_{ij\ell} + \frac{1}{4} t_{i,j+1,\ell}$. For simplicity, we examine only triangular word blurring in this paper, though more complex kernels are possible. Another variant, not considered here, blurs across the letter dimension as well to reduce sensitivity to common misspellings.

By the same token, the proposed distance is not overly sensitive to insertions, deletions, and substitutions of letters within words. It is also not severely affected by word substitutions. However, a key weakness is that without further modification, it assigns potentially catastrophic loss to word insertions and deletions. Consider the case when word $W = [w_\ell]$ is inserted at position k, which pushes back subsequent words to form a sequence of length $K+1$. Let $\boldsymbol{P'}$ be the first K words and $\boldsymbol{P''}$ the last K. The resulting similarity is lower-bounded by

$$\mathcal{S}(\boldsymbol{P}, \boldsymbol{P'}) \geq \frac{\sum_{k'=1}^{k-1} \sum_{\ell=1}^{L} p_{k',\ell}^2}{||\tilde{\boldsymbol{P}}||_F^2 - \sum_{\ell=1}^{L} p_{K,\ell}^2 + \sum_{\ell=1}^{L} w_\ell^2}, \tag{3}$$

or approximately $(k-1)/K$ in the worst case, and similarly, worst-case $\mathcal{S}(\boldsymbol{P}, \boldsymbol{P''})$ is approximately $(K-k)/K$. The analysis of word deletion is essentially identical. This analysis immediately suggests a remedy. Mutated phrases that differ by insertion or deletion of a single word can be detected by adding the similarity scores of two adjacent offsets, producing an aggregate score of roughly $(K-1)/K$ or larger, then thresholding the result. More generally, we propose postprocessing of the string correlation sequence by *gap smoothing*, which sums the top γ values in a rolling window of width δ, allowing `ffgrep` to detect up to $\gamma - 1$ insertion/deletion events with a cumulative gap of up to $\delta - 1$.

Importantly, this flexibility comes at extremely low computational cost relative to the convolution task. In Section 5.2, we present benchmarks demonstrating how these tuning parameters produce alignments that are highly comparable to those of other approximate string matching algorithms. Algorithm 1 describes their implementation in full.

Data: Word-letter matrices for pattern, $\boldsymbol{P}$, and targets, $(\boldsymbol{T}_1, \ldots, \boldsymbol{T}_N)$.

Parameters: Blur radius β, gap parameter γ, gap window δ, detection threshold θ.

Result: (i, j) pairs indicating subregion of target i beginning at offset j.

Procedure:

Preprocess pattern.

Zero-pad $\boldsymbol{P}$ to M rows, blur columns with triangular kernel of width $2\beta - 1$.
Compute total variance $||\tilde{\boldsymbol{P}}||_F = \sqrt{\sum_{k=1}^{K} \sum_{\ell=1}^{L} p_{k,\ell}^2}$, column means $\bar{P} = [\bar{p}_\ell]$.
Compute real-valued pattern spectrum $\underset{(1+M/2)\times L}{\mathcal{F}(\boldsymbol{P})}$.

Search for pattern in targets.

for *target* i *in* $1, \ldots, N$ **do**

Initialize target offset $j = 1$

Initialize similarity sequence $\left[\mathcal{S}(\boldsymbol{P}, \boldsymbol{T}_{i,1:K}), \ldots, \mathcal{S}(\boldsymbol{P}, \boldsymbol{T}_{i,(J_i-K+1):J_i})\right]$

Overlap-save.

while $j \leq J_i$ **do**

Take batch $\boldsymbol{U} = \boldsymbol{T}_{i,j:(j+M-K-\beta)}$, a row subset of target i

Zero-pad $\boldsymbol{U}$ to M rows, blur columns with triangular kernel

Compute and cache rolling batchwise target summaries.

Compute $\mathcal{F}(\boldsymbol{U})$

Initialize rolling total variance $\left[||\tilde{\boldsymbol{U}}_{1:K}||_F, \ldots, ||\tilde{\boldsymbol{U}}_{(M-K+1):M}||_F\right]$

Initialize rolling column means $\underset{M\times L}{\bar{\boldsymbol{U}}}$

for *batch offset* j' *in* $1, \ldots, M - K + 1$ **do**

Compute local variance $||\tilde{\boldsymbol{U}}_{j':(j'+K-1)}||_F = \sqrt{\sum_{j''=j'}^{j'+K-1} \sum_{\ell=1}^{L} u_{j'',\ell}^2}$

Compute local means $\bar{U}_{j'} = \left[\frac{1}{K} \sum_{j''=j'}^{j'+K-1} u_{j'',\ell}\right]$

end

Convolve and compute rolling similarity score.

Convolve $\left[\sum_{j''=j'}^{j'+K-1} \sum_{\ell=1}^{L} p_{j'',\ell}\ u_{j'',\ell}\right] = \mathcal{F}^{-1}\Big(\big(\mathcal{F}(\boldsymbol{P}) \odot \mathcal{F}(\boldsymbol{U})\big)\mathbf{1}\Big)$

Compute batch similarity sequence $[\mathcal{S}(\boldsymbol{P}, \boldsymbol{U}_{1:K}), \ldots, \mathcal{S}(\boldsymbol{P}, \boldsymbol{U}_{i,J_i})] =$

$$\left(\mathcal{F}^{-1}\Big(\big(\mathcal{F}(\boldsymbol{P}) \odot \mathcal{F}(\boldsymbol{U})\big)\mathbf{1}\Big) - \bar{\boldsymbol{U}}\bar{P}\right) \oslash \left(||\tilde{P}||_F \left[||\tilde{\boldsymbol{U}}_{1:K}||_F, \ldots, ||\tilde{\boldsymbol{U}}_{(M-K+1):M}||_F\right]\right)$$

Take first $M - K - \beta$ batch similarities and store in full similarity sequence.

Increment progress $j = j + M - K - \beta + 1$

end

Compute average of top γ peaks in rolling window of width δ

For each offset j exceeding detection threshold θ, output (i, j)

end

Algorithm 1: `ffgrep`. A procedure for efficient approximate substring alignment that applies Fourier convolution and overlap-save to word-letter matrices. $\mathcal{F}(\boldsymbol{P})$ is the univariate Fourier transform applied to columns of $\boldsymbol{P}$. The operations $\odot$ and $\oslash$ denote elementwise multiplication and division, respectively.

4 Data

We now introduce an original corpus of campaign speeches from the 2012 U.S. presidential election. A set of campaign videos were collected from `www.electad.com`. Amazon Mechanical Turk workers segmented videos to speech by President Barack Obama and Mitt Romney, eliminating introductory speakers and other extraneous portions.[1] Audio was extracted and processed with the Google Cloud Speech-to-Text API, producing 100 imperfect transcriptions of presidential campaign speech. Transcription errors, along with frequently repeated or paraphrased stump speeches, make the corpus an ideal setting to test approximate string alignment.

On average, presidential candidates spoke approximately 3,000 words over 20 minutes. From this 300,000-word corpus, we manually curated 50 ten-word campaign "catchphrases" that approximately recurred across multiple speeches. Catchphrases are reported in Table 2. Readers skeptical about the value of unsupervised phrase detection, introduced in Section 6, are encouraged to attempt this task.

[1] Three independent workers reported segmentation times for each video, and recordings with substantial disagreement were manually adjudicated.

Table 2: **Campaign catchphrases.** Ten-word campaign catchprases were manually identified from transcribed campaign speeches. For each phrase, the text (corrected for punctuation and capitalization) and date of first exact usage is reported. One phrase was used exactly by both candidates: the "United States of America is the greatest nation on earth."

Speaker	**First use**	**Pattern**
Obama	22 Jun. 2012	United States of America is the greatest nation on earth.
Obama	1 Nov. 2012	Al Qaeda has been decimated. Osama Bin Laden is dead.
Obama	4 Oct. 2012	... develop our natural gas thats right beneath our feet and...
Obama	4 Oct. 2012	... will win the election again. We'll finish what we started.
Obama	11 Oct. 2012	That's not who we are. That's not what we're about.
Obama	11 Oct. 2012	... we're fighting for. That's what's at stake in this election.
Obama	18 Oct. 2012	... fuel standards so by the middle of the next decade...
Obama	6 Sep. 2012	America is not about what can be done for us.
Obama	6 Sep. 2012	Help me recruit a hundred thousand math and science teachers.
Obama	6 Sep. 2012	I promised to end the war in Iraq. We did.
Obama	6 Sep. 2012	... immigrant who grew up here and went to school here...
Obama	6 Sep. 2012	In the next four years you can make that happen.
Obama	6 Sep. 2012	No family should have to set aside a college acceptance...
Obama	6 Sep. 2012	... since government can't do everything, it should do almost nothing...
Obama	6 Sep. 2012	... spend their golden years at the mercy of insurance companies...
Obama	6 Sep. 2012	... trucks will go twice as far on a gallon of...
Obama	6 Sep. 2012	... voices will fill the void. The lobbyists and special interests...
Obama	6 Sep. 2012	... will sustain the strongest military the world has ever known.
Obama	6 Sep. 2012	... will use the money we're no longer spending on war...
Obama	7 Sep. 2012	... chance to gain the skills that they need to compete.
Obama	7 Sep. 2012	I refuse to ask students to pay more for college.
Obama	7 Sep. 2012	Nobody who fights for this country should ever have to...
Obama	7 Sep. 2012	We want to keep investing in wind and solar and...
Obama	8 Sep. 2012	Bless you, and God bless the United States of America.
Obama	8 Sep. 2012	No company should have to look for workers in China.
Obama	8 Sep. 2012	We reinvented a dying auto industry thats back on top.
Obama	8 Sep. 2012	You're the reason why we ended don't ask dont tell.
Obama	9 Sep. 2012	... a choice between two fundamentally different visions of our future.
Obama	9 Sep. 2012	... giving tax breaks to companies that are shipping jobs overseas.
Obama	9 Sep. 2012	... there's not another country on earth that wouldn't trade places...
Obama	12 Sep. 2012	Everybody does their fair share. Everybody plays by the same...
Obama	12 Sep. 2012	It's a bargain that says hard work will pay off.
Obama	12 Sep. 2012	... to work rebuilding roads and bridges and runways and schools...
Obama	12 Sep. 2012	When our troops come home and take off their uniform...
Obama	13 Sep. 2012	America is less dependent on foreign oil than any time...
Obama	13 Sep. 2012	Make some phone calls for me and vote for me.
Obama	13 Sep. 2012	We've doubled the amount of renewable energy that we generate...
Obama	17 Sep. 2012	... new plants and training new workers and creating new jobs...
Obama	17 Sep. 2012	... polluting the air your children breathe, well, thats the price...
Obama	17 Sep. 2012	We still got the best workers in the world and...
Obama	27 Sep. 2012	... wind down the war in Afghanistan in a responsible way...
Romney	24 Feb. 2012	... free in this country to pursue happiness as we choose.
Romney	26 Feb. 2012	Those rights were life, liberty, and the pursuit of happiness.
Romney	12 May. 2012	Bless you, and God bless the United States of America.
Romney	1 Nov. 2012	We have to champion small business. We have to help...
Romney	12 Oct. 2012	... spending every year a trillion dollars more than we take...
Romney	17 Oct. 2012	... get to see North American energy independence at eight years...
Romney	25 Oct. 2012	... please raise your hand so we can recognize you here.
Romney	25 Oct. 2012	... take full advantage of our oil, our coal, our gas...
Romney	8 Sep. 2012	... parents and the teachers first, and the teachers union behind...
Romney	11 Sep. 2012	... military that is second to none, that is so strong...

5 Benchmarks

We now report results from two comparisons between `ffgrep` and an industry-standard meta-algorithm, `agrep`, that selects from a number of highly optimized fuzzy string alignment approaches. Notably, this set includes bitap [BYG92, WM92], which uses extremely fast bitwise operations and bit shifting to identify alignments with a Levenshtein distance below some threshold. We use the implementation of `TRE`, a widely used `C` library, as a reference point for evaluating the speed (Section 5.1) and accuracy (Section 5.2) of `ffgrep`. Our proposed procedure is implemented in a mix of `R` and `C++`, integrated with `Rcpp` [EF11]. Unlike `agrep`, it is not particularly well optimized and could likely benefit from the attention of a skilled programmer. However, our Fourier routines utilize the `FFTW` library [FJ93]. `FFTW` is an acronym for the "fastest Fourier transform in the West."

In additional unreported tests, we also evaluated `ffgrep` against Smith-Waterman, a dynamic programming algorithm for sequence alignment. We utilized an implementation in `Biostrings`, an `R` package for analysis of genetic and protein sequences. However, the runtime of this approach was impractically longer than either `agrep` or `ffgrep`, and we did not pursue the comparison further.

5.1 Runtime

To compare the runtime of these two string matching approaches, we turn to an extended appeal that President Barack Obama used in some form as part of 20–30 speeches, depending on the detection criterion.

> *We've doubled the amount of renewable energy that we generate from sources like wind and solar. Thousands of Americans have jobs today building wind turbines, long lasting batteries. Today, United States of America is less dependent on oil than at any time in nearly two decades.*
>
> (President Barack Obama, 4 Oct. 2012, Madison, WI.)

We begin by excerpting the first K words of this phrase as our pattern, then using `agrep` to identify approximate alignments in the full 100-document target corpus. We consider pattern lengths ranging $K = 10$ to 30, as well as edit distances thresholds of 0.1 to 0.4 times the number of characters in the pattern. Runtimes are averaged over ten searches for each parameter combination.

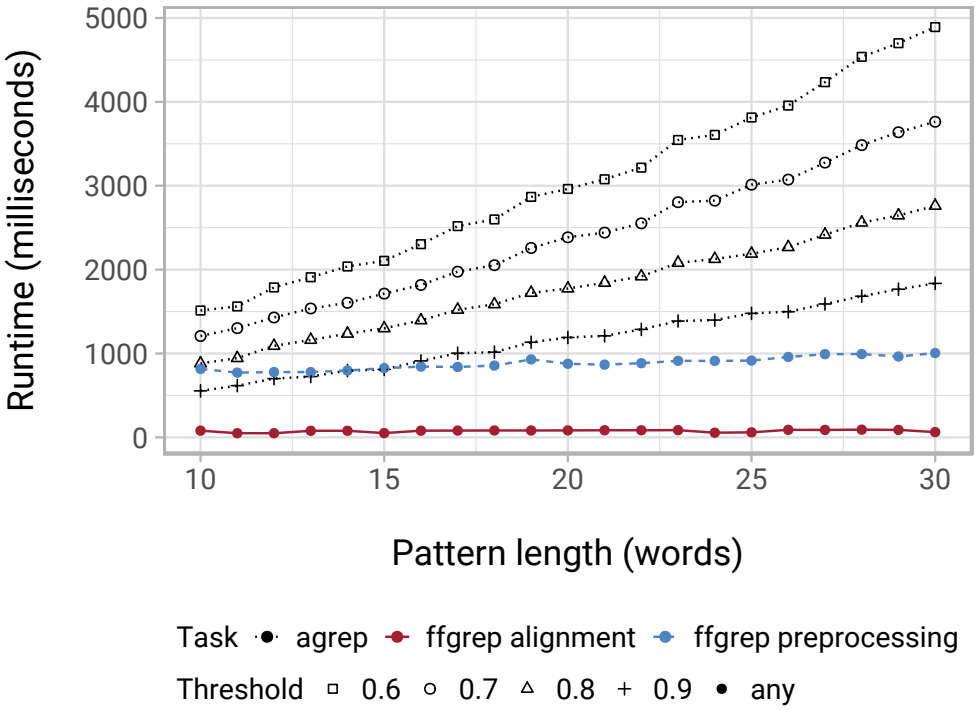
We then examine the runtime of `ffgrep` in two stages. Unlike `agrep`, our proposed procedure requires preprocessing of the target corpus to precompute its spectra and various rolling summary statistics. Thus, its first use requires substantially longer (requiring both preprocessing and alignment) than subsequent runs (alignment only). To facilitate comparison, we test preprocessing runtime by averaging ten iterations. (Because preprocessing affects only the target corpus, it is unaffected by pattern length.) We then separately report the average `ffgrep` alignment runtime for each K using default parameters, with no word blurring or gap smoothing. The time costs of these steps are miniscule by design.

The results of this comparison are reported in Figure 2. We first discuss results when seeking close matches, or those resulting in an edit distance of less than 0.1—a favorable comparison for `agrep`, as we show below. When approximate string alignment is attempted for a single pattern of ten words, `agrep` is more computationally efficient at finding close matches due to the large preprocessing time required by `ffgrep`. (Preprocessing takes longer than the actual alignment task by a factor of roughly 10–15x.) However, the two methods immediately reach parity when a second search is run. This is because the marginal per-search cost incurred by `ffgrep` is drastically lower—by factors of about 10–60x in the tasks we examine.

Next, we vary the alignment task. It is well known that the time complexity of `agrep` is increasing in the length of the search pattern or the allowed edit distance of a match. In our benchmark task, runtime roughly doubles when increasing pattern length from 10 to 20 words. Similarly, compared to a tightest edit distance threshold considered, looser matching criteria can result in nearly triple the time cost. In fact, when using an edit distance threshold of 0.2 times the pattern's character count, combined `ffgrep` preprocessing and alignment is faster than a *single* invocation of `agrep`. (This threshold is not a unrealistic one; in practice, we find that it is often appropriate for detecting light paraphrasing.)

In `ffgrep`, longer patterns impact alignment time as well, but to a lesser extent. Preprocessing time increased by roughly 25% when doubling pattern length from 10 to 20, and alignment time increased by less than 10%. Minor growth in search time stems from the fact that when longer patterns are used, greater overlap between batches is needed in the overlap-save portion of the convolution. This increases the number of batches, which in turn affects both preprocessing and alignment time roughly linearly. However, the techniques that we develop for controlling sensitivity to word insertions, deletions, and transpositions are by design virtually free. Blurring incurs a fixed cost for each target and each search pattern, but not for every combination thereof. Gap smoothing is applied to the convolution output of every pattern-target combination, but the time cost of both operations are negligible due to the high efficiency of rolling windowed operators.

Figure 2: **Runtime benchmarks.** Dotted black lines indicate average `agrep` time cost, with different shapes each denoting a detection threshold. Thresholds are expressed in terms of the ratio of allowed edit distance to pattern letter count. The runtime of `agrep` increases substantially with longer search patterns or looser detection thresholds. The dashed blue line indicates `ffgrep` preprocessing time, a one-time cost per target corpus. The runtime of a single `ffgrep` alignment depicted with a solid red line. The time complexity of `ffgrep` is unaffected by detection threshold.



5.2 Performance

We now demonstrate that `ffgrep`'s substantial computational gains do not come at a large cost in performance. To do so, we treat `agrep` results as ground truth (though in reality both algorithms almost certainly miss semantically related phrases in the target corpus) and show that `ffgrep` produces closely similar results. For clarity, we focus on `agrep` matches obtained at a edit-distance similarity threshold of $\theta_a = 0.75$ (the midpoint of the range considered, allowing a maximum number of edits equal to $\frac{1}{4}$ of the pattern length in characters).

To facilitate reporting, the approximate alignment problem is recast to a binary classification problem as follows. For each of the 50 patterns listed in Table 2, we use `agrep` to search within each of the 100

target documents. If a target document contains any subsequence with similarity score exceeding θ_a, a "hit" is recorded for the entire target document. This procedure avoids complications with slight differences in alignment offsets between the two algorithms. It results in 630 hits out of 5,000 pattern-target combinations.

For each parameter setting of `ffgrep`, we conduct the same 50 searches. Each setting again produced 5,000 binary classifications, which were compared to those from `agrep`. Optimal `ffgrep` parameters were then selected by maximizing the F_1 score. (`ffgrep` strictly makes rule-based decisions and contains no estimated parameters, only tuning parameters, so overfitting is not a concern.) Performance is shown in Table 3. Overall, we find that the two algorithms produce closely similar results, with `ffgrep` yielding F_1 of 0.89 and 97% accuracy when treating `agrep` results as ground truth. We find that both approaches are highly comparable over a wide range of search laxity (edit-distance thresholds ranging from $\theta_a = 0.60$ to 0.90), suggesting that `ffgrep` is able to function as a general-purpose replacement when speed is an important consideration.

Table 3: **Performance benchmark.** `ffgrep` parameters were tuned to reproduce search results found by `agrep` when used with a threshold of $\theta_a = 0.75$. Optimal `ffgrep` results, as measured by F_1, were obtained with $\beta = 2$ (blurring words into their immediate neighbors) and $\gamma = 0$ (no gap smoothing). The optimal model exhibits excellent performance; details are reported in the second column. Brackets contain the minimum and maximum `ffgrep` performance when attempting to reproduce different `agrep` detection thresholds, ranging from $\theta_a = 0.6$ to $\theta_a = 0.9$. These results show that `ffgrep` is a generally adequate replacement for `agrep` results over a wide range of parameter settings.

Metric	$\theta_a = 0.75$	$\theta_a \in [0.6, 0.9]$
Accuracy	0.97	[0.94, 0.98]
Precision	0.91	[0.81, 0.93]
Sensitivity	0.87	[0.78, 0.92]
Specificity	0.99	[0.98, 0.99]
F_1	0.89	[0.84, 0.90]

6 Unsupervised Phrase Clustering

We now turn to a substantive application that motivated the development of `ffgrep`. Our objective is to identify recurring themes in political speech, like candidate catchphrases and campaign themes, with no human intervention. The efficiency of the proposed approach makes it possible to compare every possible phrase in the corpus to every other phrase, reducing the task to a network clustering problem. Below, we describe details of the analysis and illustrate its results.

The corpus is cut into over 300,000 possible patterns by using a ten-word-wide window that is incrementally advanced one word at a time. After precomputing and caching the spectra of the target documents, Algorith 1 is applied for each pattern; it produces the same results as sweeping the pattern over every target and computing pointwise correlation similarity at each offset. Words are blurred into their immediate neighbors ($\beta = 2$), and we postprocess with gap smoothing that allows for a single-word insertion or deletion ($\gamma = 2$ and $\delta = 2$). For any given pattern, the hit (i, j) represents a subsequence in target i at offset j, which uniquely identifies another pattern. From this, we obtain a sparse, thresholded pattern similarity matrix containing the equivalent of 90 billion pairwise comparisons.

Table 4: **Selection of automatically extracted catchphrases.** Detected catchphrases by Mitt Romney include specific wordings of (a) an energy plan and (b) acknowledgement of veterans. These are particular instances of recurring themes that appeared in 28 and 27 campaign speeches, respectively (alternative formulations omitted for space). Automatically extracted catchphrases by President Barack Obama include discussions of (c) military strategy, 22 speechs; and (d) a Medicare reform critique, 34 speeches. Consecutive phrases are merged for clarity.

(a)	do it ive got five things ill do to get this economy going number one were going to take full advantage of our oil our gas are cold
	make sure we take advantage of our oil or gas or coal or nuclear renewables
	is to finally take advantage of our oil our gas are cold our nuclear
	is take full advantage of our oil our all our guests

(b)	our veterans and armed services members please raise your hands and be recognized
	veterans please raise your hand so we can recognize you
	strife who more than self their country loved and mercy more than life what our veterans and members of the armed services please raise your hands wow
	veterans are members of members of our armed forces please raise your hand

(c)	abroad four years ago i promised to end the war in iraq we did i said wed wind down the war in afghanistan and we are while a new tower rises above the new york skyline
	were going to wind down the war in afghanistan we are and while a twin tower a new a new tower rises above the new york skyline al qaeda is
	what happens abroad for years ago i promised to end the war in iraq and i did i said wed wind down the war in afghanistan in a responsible way and we are
	years ago i told you id end the war in iraq and we did i said id end the war in afghanistan and we are i said we focus on the people who actually attacked us on 911

(d)	going to ask 360000 ohio students to pay more for college or kick children off of head start or eliminate health insurance for the poor
	will kick students are financial aid or get rid of funding for planned parenthood or eliminate health care for millions on medicaid
	or kick kids out of head start or eliminate health insurance for millions of americans just to pay
	to pay for another millionaires tax cut refuse to ask students to pay more for college or kick children out of head start programs to eliminate health insurance for millions of americans who are poor and elderly or disabled also those with the most can pay

Figure 3: **Recovered structure within a single automatically detected campaign appeal.** The 100-document campaign speech corpus was cut into over 300,000 prospective patterns. For each one, `ffgrep` was used to efficiently search for approximate alignments in the full corpus. Components in the thresholded similarity graph represent candidate catchphrases and recurring appeals. The vertex contraction of one component is depicted below; consecutive patterns from the same document are fused and concatenated to facilitate visualization. With no human intervention, the approach automatically detects a critique of oil lobby influence and a related, but not always co-occurring, call for investment in clean energy.

The vast majority of patterns are unique isolates. However, we identify almost 41,500 ten-word segments with at least a single close match, falling into over 1,300 components. In Table 4, we highlight a selection of specific quotes from several recurring themes. Within a broader theme or cluster, exact quotes can exhibit considerable variation; Figure 3 illustrates how President Obama highlights focus on different aspects of his environmental platform at different points in the election. In general, smaller clusters are more common. For example, one cluster contains only two nodes (Romney's "... never again will we apologize for America abroad..." and "... I will never apologize for America abroad...")—a rare backhanded attack from a candidate that usually spoke in more directly confrontational terms about his *opponent's* supposed apologies.

We defer further analysis of catchphrases to future applied work. More generally, the ability to detect mesoscopic structure in text corpora—i.e., at a higher level than individual words or n-grams, but below that of bag-of-words topics—is a useful tool for studying the introduction, diffusion, and evolution of concepts or memes over time and space. However, an important caveat is that the proposed approach is better able to clustering concepts that either (1) have a low mutation rate or (2) are frequently recurring. The first is self-evident; when two manifestations of the same concept are similarly worded, correlation similarity will tend to be high. As for the second, when phrases A and B reflect the same underlying concept but use very different wording, no edge is formed, but if another phrase C uses elements of both, the three will be grouped together. This property means that comparisons across memes should be undertaken with care.

7 Conclusion

In this paper, we develop `ffgrep`, a new algorithm for approximate alignment of strings. The method is shown to be highly efficient, enabling previously infeasible tasks such as N-to-N comparisons of phrases in large corpora. Performance on an English corpus is generally good, and this feature is likely to hold in genetic strings as well. However, `ffgrep` is almost certainly unviable for languages such as Chinese even with phoneticization. Important questions remain about its applicability to word-embedding representations. While data structures for embedding sequences and word-letter sequences are virtually identical, the semantic meaning of covariance in embedding space remains unclear. We leave this interesting question to future work.

References

[AAL+97] A. Amir, Y. Aumann, G. Landau, M. Lewenstein, and N. Lewenstein. Pattern matching with swaps. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 144–153, 1997.

[BYG92] Ricardo A. Baeza-Yates and Gastón H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, 1992.

[Dam64] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, 1964.

[EF11] Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011.

[FJ93] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 2:216–231, 93.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[HIKP12] Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Simple and practical algorithm for sparse fourier transform. In *ACM-SIAM Symposium on Discrete Algorithms*, 2012.

[KS95] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for the inversion distance between two permutations. *Algorithmica*, 13:180–210, 1995.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[NW70] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 44:444–453, 1970.

[Tic84] W. Tichy. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems*, 2:309–321, 1984.

[Ukk92] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theoretical Computer Science*, 1:191–211, 1992.

[WM92] Sun Wu and Udi Manber. Fast text searching: allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.