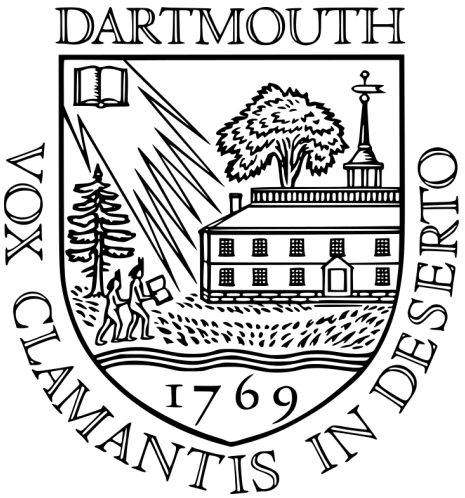
Dartmouth College
Hanover, NH

HONORS THESIS



Paula X. Chen

Neural Spike Sorting Algorithms for Overlapping Spikes

Department of Mathematics

Professor Alexander H. Barnett, *advisor*
Submitted in partial fulfillment of a Bachelor of Arts in Mathematics

May 2017

Acknowledgments

I extend my deepest gratitude to my advisor, Professor Alexander H. Barnett, for his endless expertise, guidance, and patience. His flexibility and constant communication demonstrate his dedication to teaching and that collaboration is not bounded by state (or at times, international) lines. Needless to say, without him, this thesis would not have been completed. Additionally, I thank my family for their lifelong support and for inspiring me to be passionate about math, even when it wasn't my strong suit; my friends for enduring a year of me geeking out about formatting, MATLAB, and math-related stories that probably weren't as amusing as I found them to be; and the Kaminsky Family Fund Award for allowing me to travel to the Simons Foundation in New York and get a taste of life working in the math industry.

Title: Neural Spike Sorting Algorithms for Overlapping Spikes

Author: Paula X. Chen

Department: Department of Mathematics, Dartmouth College

Thesis Advisor: Alexander H. Barnett, Associate Professor of Mathematics

Abstract: One of the key, unsolved problems in spike sorting is that of overlapping spikes. This project explores different variants of greedy algorithms for spike sorting to address this problem. In particular, this project defines simple greedy, forward-backward greedy, and greedy with pairs algorithms for sorting overlapping spikes. Using toy models, we ultimately show that greedy with pairs is the most promising of the introduced algorithms and represents a reasonable compromise between sorting accuracy and computational efficiency. We also test our algorithms on real signals and discuss the issues in assessing sorting accuracy when no ground truth is available for comparison.

Keywords: spike sorting, overlapping spikes, greedy algorithms

List of Symbols and Abbreviations

Symbol/ Abbreviation	Description
k	number of known spike types
n_s	number of time shifts
N	number of time samples per clip
T	firing time, center of the spike
t	any sample time within a clip
η	noise level, $1/SNR$
SNR	signal-to-noise ratio
$\mathbf{H}$	N-component noise vector, $\sim \mathcal{N}(0, \eta^2 I_N)$
$\mathbf{f}_i$	N-component vector, known (synthetic) spike shape i
$\mathbf{g}_i$	N-component vector, real spike shape i
γ_i	probability of $\mathbf{f}_i$ appearing in a clip/ expected firing rate of neuron i
$\mathbf{y}$	N-component signal vector
λ_i	penalty for detecting spike shape i

Contents

Chapter 1

Introduction

In order to understand how the brain works, we begin by studying the firing of individual neurons. The most common method for measuring neural firing is direct electrical recording, which uses electrodes to record the voltage patterns of nearby neurons [1]. Each neuron yields a characteristic electrical signal, known as a spike. Thus, spike sorting, the grouping of spikes by shape, can be used to match measured signals to their generating neurons. More precisely, spike sorting extracts the firing times and corresponding neuron labels of noisy electrophysiological recordings. In practice, spike sorting has been used to study retinal function as well as learning and memory in the hippocampus.

One of the major problems in spike sorting is that of overlapping spikes. When nearby neurons fire at similar times, their corresponding spike signals overlap. If two nearby spikes consistently fire synchronously, we would suspect that such neurons are functionally related. The primary challenges of overlapping spikes are that one, it can be difficult to determine whether a signal is the result of overlapping spikes or some unique single spike; and two, once a single-spike explanation is ruled out, there is no clear method for determining which spikes are present [6].

Additionally, the problem very quickly becomes computationally-intensive. Given k spike types, there are 2^k possible combinations. But the firing times of the overlapping spikes are not necessarily known. Thus, in testing n_s time shifts, the problem difficulty increases to maximally $O(2^{kn_s})$. It is clear that in addressing overlapping spikes with time shifts, we need a method that is not only accurate, but efficient, as well.

The outline for our project is as follows. Before we actually sort spikes, we first address the question of how to determine whether or not a clip contains a spike. Once we answer that question, we then develop algorithms for sorting spikes. First, we develop a basic greedy algorithm for spike sorting and compare its performance to a brute-force method, which represents the globally optimal fit. We then show that while the greedy algorithm is markedly more efficient than brute-force fitting, it is not as accurate as brute-force fitting, especially when time shifts are introduced. After discussing a few modifications and variants, we ultimately propose that the greedy algorithm with pairs

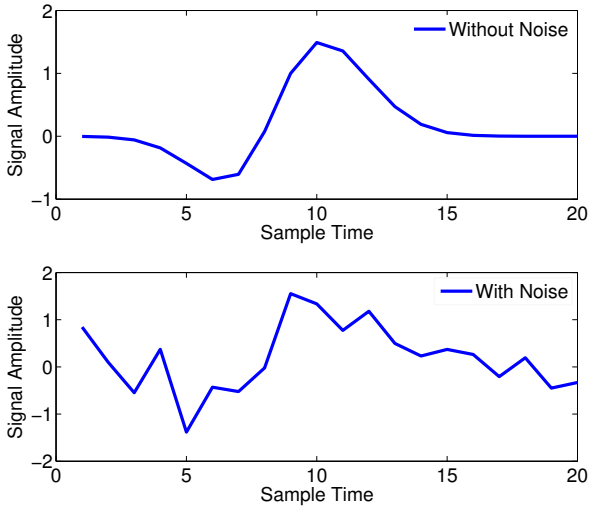


Figure 1.1: Sample clip containing overlapping spikes with and without normally-distributed iid noise ($\eta = 0.4$ or $\eta = 0$, respectively). This sample depicts the overlap of spikes $\mathbf{f}_1$ ($T = 9$) and $\mathbf{f}_4$ ($T = 5$).

is a viable solution for sorting overlapping spikes with time shifts. Finally, we test our algorithms on real spikes and discuss how to assess sorting accuracy in the absence of a ground truth.

1.1 Model Assumptions and Simplifications

In practice, signals are recorded using multichannel (i.e. multielectrode) arrays. For simplicity, we consider only one channel of data at a time. We also reduce the signal to clips, or small time windows of length N. Clips are assumed to be chosen (by algorithms not discussed here) such that suspected spikes are more or less centered within the clip. We define the center of the spike to be the firing time T of the spike. Note that we use T specifically to denote the firing time, and t to denote any sample time within a clip.

In practice, the number of spike types k is an unknown value. Here, we assume that we have k known spike type templates. Unless otherwise stated, we run our algorithms on synthetic spikes (Figure 1.2). These spike shapes are generated as either Gaussians or Gaussian derivatives. In general, assuming a Gaussian distribution is a reasonable and commonly used approach for spike sorting algorithms [6]; notably, this assumption also allows for simpler probabilistic analysis and models for the data. In practice, these spike type templates are extracted as the mean voltage waveforms.

Additionally, we also assume iid, Gaussian noise. Noise vectors $\mathbf{H}$ are generated such that $\mathbf{H} \sim \mathcal{N}(0, \eta^2 I_N)$ and are additive to the true spike shapes. Realistically, we would not expect noise to be iid and uncorrelated since background noise generally consists of the signals of spikes further away from the electrodes [6]. We make this assumption in order to allow for Bayesian among other probabilistic models, noting that this assumption is not uncommon among others using similar approaches [3, 4]. In practice,

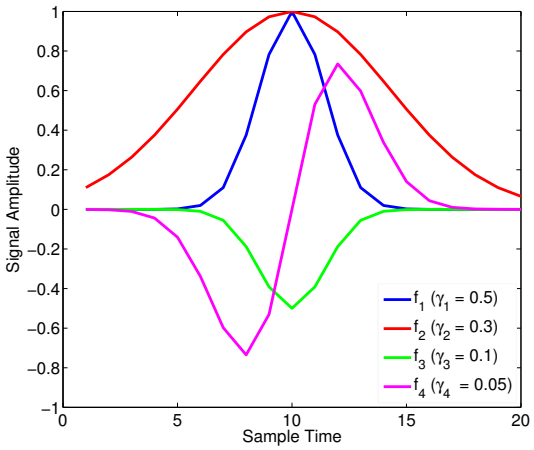


Figure 1.2: Synthetic spike types used.

Figure 1.3: Sample error fraction bars.

noise levels can be extrapolated using band pass (i.e. amplitude) filtering or from clips determined to contain no spikes. Since we use spike amplitudes that are maximally 1 and minimally -1, we interpret the noise level η as corresponding to $1/SNR$, where SNR is the the signal-to-noise level. Note that this interpretation is also only possible when noise is assumed to be Gaussian.

Neurons vary in firing rates, usually ranging from from 1 to 100 Hz. We use expected firing rates that range from $\gamma = 0.5$ (i.e. a frequently firing neuron) to $\gamma = 0.05$ (i.e. an infrequently firing neuron) in order to model these variations. This γ parameter could also be easily extrapolated experimentally.

Neural firing includes a refractory period; after a neuron has fired, there is a short period of time during which the neuron cannot fire again. Refractory periods usually last about 2 ms. Typically, samples are recorded at a rate of 10-30 kHz. Thus, a refractory period of 2 ms is equivalent to approximately 20-60 time samples. Since we use $N = 20$, we can assume that no spike type can appear more than once per clip.

1.2 Error Analysis

As a measure of algorithm performance, we use error fractions for actual and detected spikes (Figure 1.3). For both actual and detected spikes, we interpret the correct rates to be the fraction of spikes that are both present and detected correctly. The fraction of actual spikes that is correct is also referred to as the sensitivity; the fraction of detected spikes that is correct is also referred to as the precision. For both actual and detected spikes, the wrong rates represent spikes that are present but are incorrectly identified. The miss rate (type II error) represents the fraction of actual spikes that are not detected. The false positive rate (type I error) represents the fraction of detected spikes that are not actually present.

We represent these error fractions via bar graphs (see Figure 1.3). For all experiments using our synthesized spike types, the error bars of these bar graphs represent the standard deviation of 5 trials of 200 runs each. We calculate the standard deviation s as

follows:

$$s = \sqrt{\frac{\sum_{i=1}^{5}(x_i - \bar{x})^2}{5}},$$

where the x_i's represent the observed values and $\bar{x}$ is the mean value. Note that we divide by $\sqrt{5}$, instead of $\sqrt{4}$, since our estimation of the sample standard deviation involves all 1000 samples used.

Chapter 2

Basic Spike Detection

In this chapter, we consider the detection of single spikes. Namely, we need to be able to distinguish between $\mathbf{0}$ and any known spike $\mathbf{f}$ given iid, Gaussian noise. Thus, (in this chapter only) we use the following error matrix:

Detection \ Truth	No Spike	Spike
No Spike	**True Negative**	**Missed (Type II)**
Spike	**False Positive (Type I)**	**Correct**

Table 2.1: Error Matrix for Single Spike Detection

Consider a signal vector $\mathbf{y}$. In this binary case, our generative model for $\mathbf{y}$ is given by

$$\mathbf{y} = \begin{cases} \mathbf{0} + \mathbf{H} & \text{for 0 spikes} \\ \mathbf{f} + \mathbf{H} & \text{for 1 spike,} \end{cases}$$

where $\mathbf{H} \sim \mathcal{N}(0, \eta^2 I_N)$ is a random noise vector. Since the elements of $\mathbf{H}$ are iid and normally distributed, we have that

$$p(\mathbf{H}) = \prod_{j=1}^{N} \frac{1}{\sqrt{2\pi\eta^2}} e^{-\frac{H_j^2}{2\eta^2}} = \frac{1}{\sqrt{2\pi\eta^2}} e^{-\frac{\|\mathbf{H}\|^2}{2\eta^2}}.$$

Thus, with the addition of noise, signals containing no spike form a Gaussian blob centered at $\mathbf{0}$, and signals containing $\mathbf{f}$ form a Gaussian blob centered at $\mathbf{f}$ (Figure 2.1). We then determine whether or not $\mathbf{y}$ contains a spike using linear classifiers. We do this by setting a threshold Ω and defining a detection vector (i.e. linear classifier) ω, such

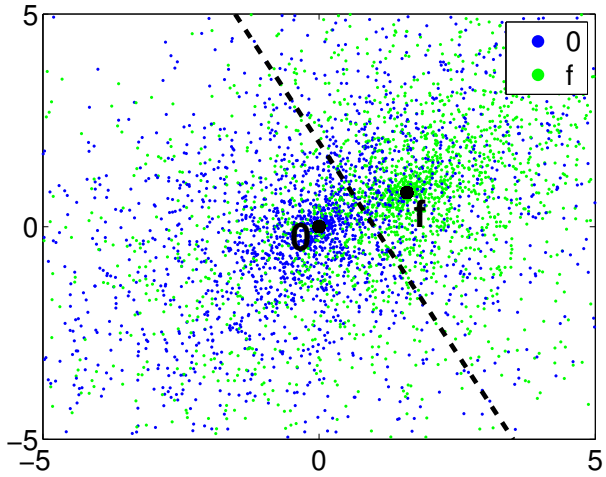


Figure 2.1: Gaussian blobs of signals containing either $\mathbf{0}$ or $\mathbf{f}$ plus iid, Gaussian noise. The dotted line represents the decision hyperplane.

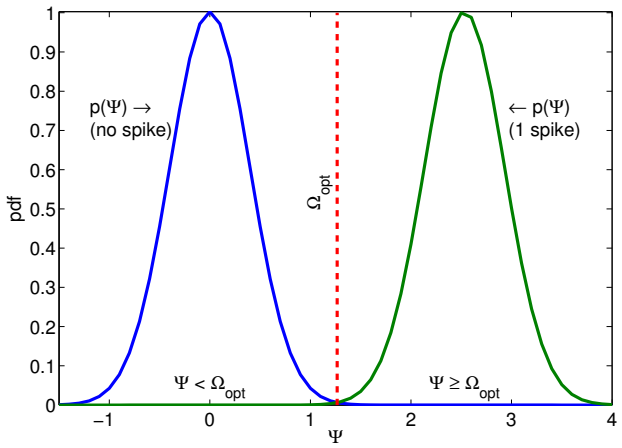


Figure 2.2: Diagram of spike detection via linear classifiers.

that if $\omega^T \mathbf{y} \geq \Omega$, then we conclude that $\mathbf{f}$ is present; if $\omega^T \mathbf{y} < \Omega$, then we conclude that no spike is present. We refer to $\Psi = \omega^T \mathbf{y}$ as the detection parameter.

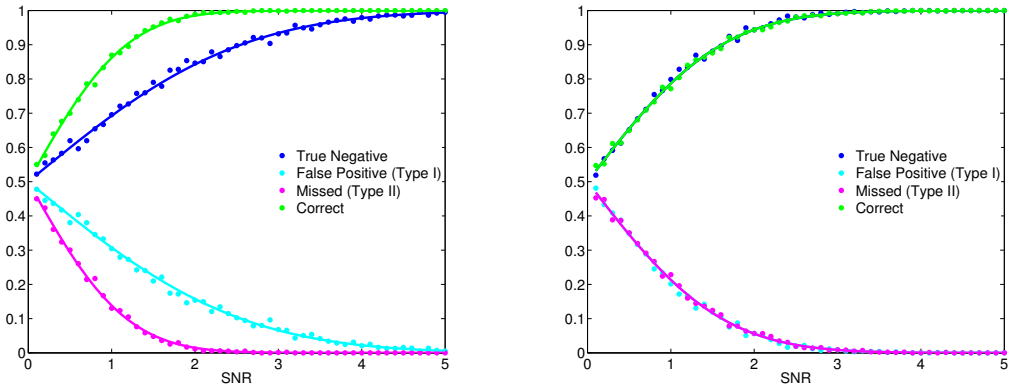
Proposition 1. $\omega = c\mathbf{f}$ *is the optimal detection vector.*

Proof. Note that the detection parameter can be written as

$$\Psi = \omega^T \mathbf{y} = \begin{cases} \omega^T \mathbf{H} & \text{for 0 spikes} \\ \omega^T (\mathbf{f} + \mathbf{H}) & \text{for 1 spike.} \end{cases}$$

Thus, the expected value of Ψ is

$$E(\Psi) = \begin{cases} E(\omega^T \mathbf{H}) = 0 & \text{for 0 spikes} \\ E(\omega^T \mathbf{f} + \omega^T \mathbf{H}) = E(\omega^T \mathbf{f}) + E(\omega^T \mathbf{H}) = \omega^T \mathbf{f} & \text{for 1 spike.} \end{cases}$$



(a) $\Omega = 0.8$ (b) $\Omega = \Omega_{opt} \approx 1.2660$

Figure 2.3: Error fractions for detecting spikes using linear classifiers. Error fractions are calculated as the column-normalized values of Table 2.1. The solid lines represent the theoretical curves.

The variance of Ψ is

$$V(\Psi) = \begin{cases} V(\omega^T \mathbf{H}) = \omega_1^2\eta^2 + \cdots + \omega_N^2\eta^2 = \|\omega\|^2\eta^2 & \text{for 0 spikes} \\ V(\omega^T \mathbf{f}) + V(\omega^T \mathbf{H}) = 0 + \|\omega\|^2\eta^2 = \|\omega\|^2\eta^2 & \text{for 1 spike.} \end{cases}$$

Let $\sigma = \|\omega\|\eta$. Then, the pdf for Ψ is

$$p(\Psi) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\Psi^2}{2\sigma^2}} & \text{for 0 spikes} \\ \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\Psi-\omega^T\mathbf{f})^2}{2\sigma^2}} & \text{for 1 spike.} \end{cases}$$

Note that the pdf for the 0-spike case overlaps with that for the 1-spike case at $\Psi_{opt} = \frac{1}{2}\omega^T\mathbf{f}$. The optimal ω should minimize this overlap, i.e. maximize $\Psi_{opt} = \frac{1}{2}\omega^T\mathbf{f}$ (Figure 2.2).

Using Lagrange multipliers and the constraint $\|\omega\|^2 = 1$, we have the following system of equations

$$\nabla\Psi_{opt} = \lambda\nabla\|\omega\|^2$$
$$\|\omega\|^2 = 1.$$

Solving this system of equations, we get that $\omega = \frac{1}{4\lambda}\mathbf{f}$, where $\lambda = \pm\frac{4}{\|\mathbf{f}\|}$. By defining $c = \frac{\|\mathbf{f}\|}{16}$, we get the desired result. □

Note that the above proof also shows that the optimal threshold is

$$\Omega_{opt} = \Psi_{opt} = \frac{1}{2}\omega^T\mathbf{f} = \frac{1}{2}\|\mathbf{f}\|.$$

This application of linear classifiers demonstrates that spike detection is essentially just a maximum likelihood problem.

Figure 2.3 shows the error fractions for detecting spikes using the linear classifier $\omega = \mathbf{f}$ and various thresholds. The theoretical curves are calculated as follows:

$$\text{True Negative} = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{\Omega}{\sigma\sqrt{2}}\right), \qquad \text{False Positive} = \frac{1}{2} - \frac{1}{2}\text{erf}\left(\frac{\Omega}{\sigma\sqrt{2}}\right)$$

$$\text{Missed} = \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{\Omega - \Omega_{opt}}{\sigma\sqrt{2}}\right), \qquad \text{Correct} = \frac{1}{2} - \frac{1}{2}\text{erf}\left(\frac{\Omega - \Omega_{opt}}{\sigma\sqrt{2}}\right).$$

As expected, the experimental error fractions match the theoretical values. Note that as $SNR = 1/\eta$ approaches 0, all the error fractions approach $1/2$. This result corresponds to the fact that when noise levels are very high, spike detection is essentially equivalent to the flip of a coin. We also note that as $\eta = 1/SNR$ approaches 0, the correct and true negative rates approach 1, while the false positive and miss rates approach 0. This result corresponds to the fact that as noise levels approach 0, spike detection performs increasingly more accurately.

Chapter 3

The Greedy Algorithm for Spike Sorting

The greedy algorithm is a well-established method for solving computationally expensive (i.e. NP-hard or exponentially-hard) problems. The greedy algorithm approximates the globally optimal solution by making the locally optimal choice at each step. The greedy algorithm is unidirectional, meaning that it cannot undo any of its past choices. Thus, if the locally optimal choices lead the wrong path, the greedy algorithm will fail to find the globally optimal solution.

In this chapter, we establish a simple forward greedy algorithm for spike sorting. For comparison, we also create a brute-force algorithm, which takes an exhaustive approach, to represent globally optimal fitting. In the following chapters, we introduce variations of greedy aimed at improving the algorithm accuracy. Note that in this chapter, we define and test our algorithms on spikes *without* time shifts.

3.1 Defining the Simple Forward Greedy Algorithm

We define the simple forward greedy algorithm as beginning with no spikes detected and then detecting optimal spikes, one-by-one. Note that we specifically use the term "forward" to distinguish this algorithm from a later variant (namely, the backward greedy algorithm). Throughout the remainder of this paper, we will use the terms "simple greedy", "simple forward greedy", and "forward greedy" interchangeably to refer to the algorithm we define here.

Consider a signal vector $\mathbf{y}$. Let $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_k$ represent k known spike shapes, and let F be an index set for the set of detected spikes. Then, we define the optimal spike to be the spike shape $\mathbf{f}_I$ that minimizes the least squared residual. This spike is given by the index

$$I = \underset{i \in \{1,2,...,k\}-F}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{f}_i\|_2^2, \tag{3.1}$$

provided that

$$\|\mathbf{y} - \mathbf{f}_I\|^2 \leq \|\mathbf{y}\|^2 \tag{3.2}$$

Thus, if the above condition is satisfied, we accept (detect) spike shape $\mathbf{f}_I$ as present in the signal vector, we update the signal vector ($\mathbf{y} = \mathbf{y} - \mathbf{f}_I$), and the algorithm continues. Conversely, if $\mathbf{f}_I$ fails to satisfy this condition, then we conclude that there is no optimal spike, and the algorithm terminates. Pseudocode for simple greedy is outlined in Algorithm 1 below. The computational effort for simple greedy is $O(k)$.

3.2 Brute-Force Fitting

Throughout this paper, we define brute-force fitting algorithms as testing every possible combination of spikes. We consider the results of brute-force fitting to represent the globally optimal solution. Because brute-force fitting takes an exhaustive approach, it is notably inefficient. Without time shifts, the brute-force algorithm always requires 2^k steps. Thus, we consider an ideal algorithm as matching the accuracy of brute-force fitting but significantly more efficient.

3.3 Decision Boundaries in the Two-Spike Case

Consider the $k = 2$ case. Let $\mathbf{y}$ be the signal vector, and $\mathbf{f}_1, \mathbf{f}_2$ be the two known spike shapes. Then, we can predict the performance of the simple forward greedy algorithm by deriving the decision boundaries for the four possible spike combinations: $\mathbf{0}, \mathbf{f}_1, \mathbf{f}_2$, and $\mathbf{f}_{1+2}$. These decision boundaries exist in $\mathbb{R}^N$ and represent the regions of points closest to each combination.

Lemma 2. *If $\|\boldsymbol{f}_1 - \boldsymbol{f}_2\| \leq \|\boldsymbol{f}_1 + \boldsymbol{f}_2\|$, then the simple forward greedy algorithm yields the globally optimal solution.*

Algorithm 1: Simple Forward Greedy

Inputs : $\mathbf{y}$ = input signal vector, $\mathbf{f}_1, ..., \mathbf{f}_k$ = known spike types
Output : $\mathbf{F}$ = indices of detected spikes

```
s = 0;
while s = 0 do
    I = argmin_{i∈{1,...,k}−F} ||y − f_i||^2   # detecting spike type I minimizes LSE;
    δ_f = ||y||^2 − ||y − f_I||^2              # improvement;
    if δ_f ≥ 0 then
        F ← F ∪ {I};
        y ← y − f_I;
    else
        s = 1;
    end
end
```

Proof. Simple forward greedy fitting begins with no spikes detected. To prove the desired result, we consider two cases.

Case 1: $\|\mathbf{y} - \mathbf{f}_1\|^2 < \|\mathbf{y} - \mathbf{f}_2\|^2$

In this case, we clearly have that

$$1 = \underset{i \in \{1,2\}}{\operatorname{argmin}} ||\mathbf{y} - \mathbf{f}_i||_2^2 .$$

Therefore, the first step of greedy fitting detects spike type $\mathbf{f}_1$ (Figure 3.1a). Next, the algorithm decides whether or not to add $\mathbf{f}_2$, which creates a decision boundary between $\mathbf{f}_1$ and $\mathbf{f}_1 + \mathbf{f}_2$ (Figure 3.1b).

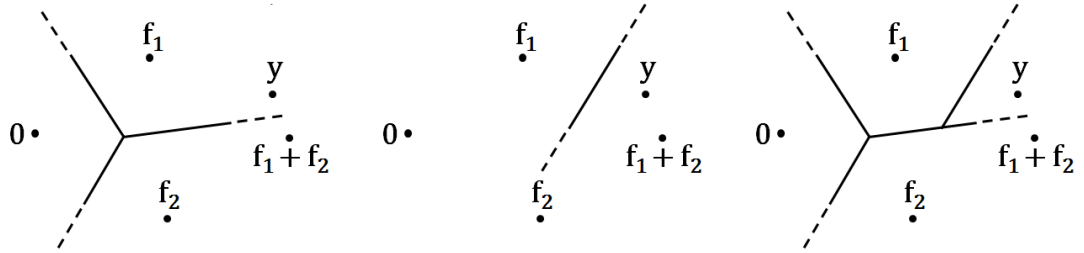
Case 2: $\|\mathbf{y} - \mathbf{f}_1\|^2 > \|\mathbf{y} - \mathbf{f}_2\|^2$

In this case, we clearly have that

$$2 = \underset{i \in \{1,2\}}{\operatorname{argmin}} ||\mathbf{y} - \mathbf{f}_i||_2^2 .$$

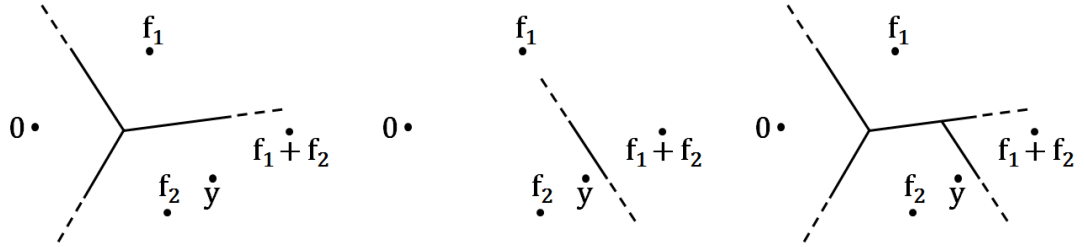
Therefore, the first step of fitting detects spike type $\mathbf{f}_2$ (Figure 3.2a). Next, the algorithm decides whether or not to add $\mathbf{f}_1$, which creates a decision boundary between $\mathbf{f}_2$ and $\mathbf{f}_1 + \mathbf{f}_2$ (Figure 3.2b).

Combining the decision boundaries we found in Cases 1 and 2, we see that we recover those for brute-force fitting, i.e. those for globally optimal fitting (Figure 3.3). □



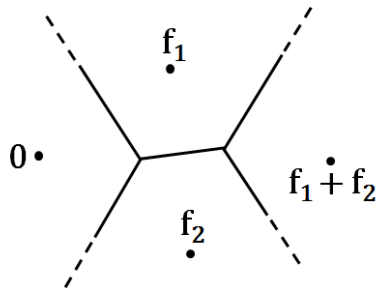
(a) Detect $\mathbf{f}_1$ (b) Detect $\mathbf{f}_1 + \mathbf{f}_2$ (c) Final Boundaries

Figure 3.1: Decision Boundaries for Simple Greedy in Case 1: $\|\mathbf{y} - \mathbf{f}_1\|^2 < \|\mathbf{y} - \mathbf{f}_2\|^2$.

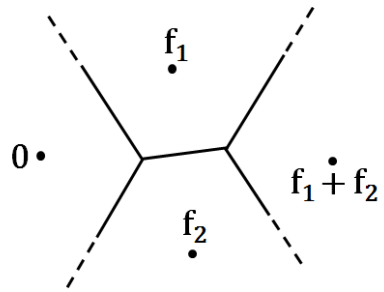


(a) Detect $\mathbf{f}_2$ (b) Detect $\mathbf{f}_2$ (c) Final Boundaries

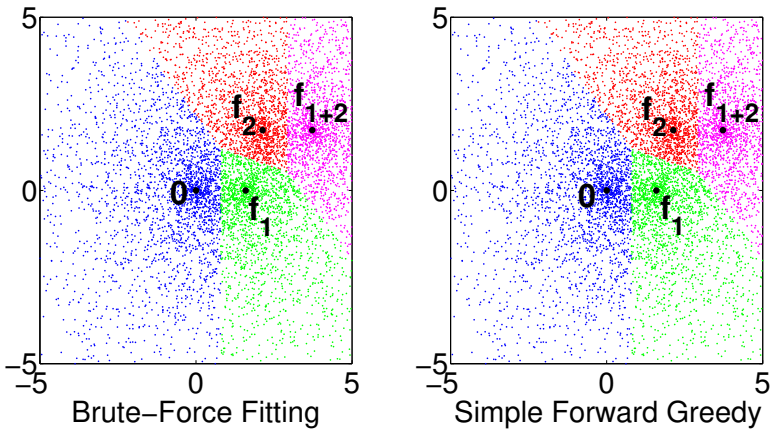
Figure 3.2: Decision Boundaries for Simple Greedy in Case 2: $\|\mathbf{y} - \mathbf{f}_1\|^2 > \|\mathbf{y} - \mathbf{f}_2\|^2$.



(a) Brute-Force Fitting (Theoretical)

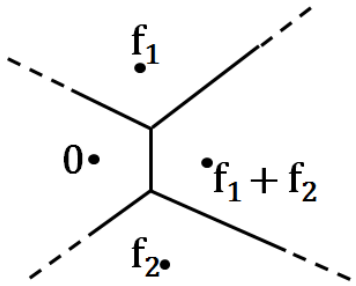


(b) Simple Forward Greedy (Theoretical)

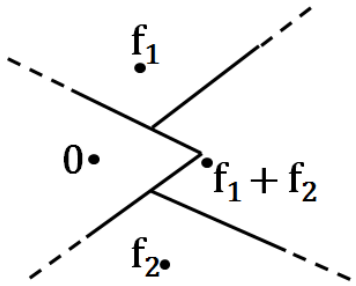


(c) Experimental Decision Boundaries

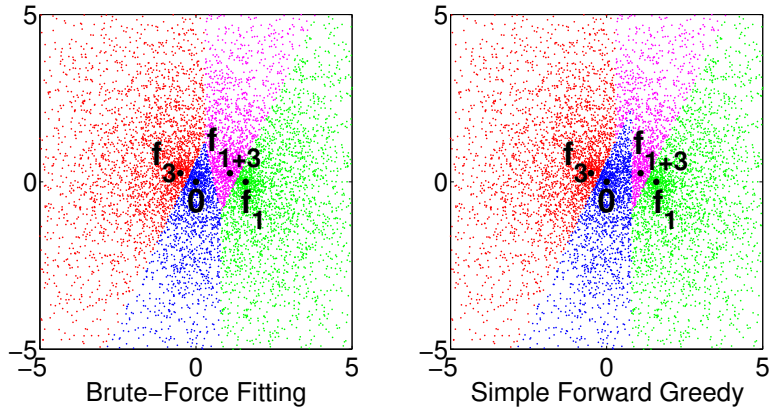
Figure 3.3: Decision boundaries for simple greedy if $\|\mathbf{f}_1 - \mathbf{f}_2\| \leq \|\mathbf{f}_1 + \mathbf{f}_2\|$ (condition for Lemma 2). Note that the decision boundaries are the same for both algorithms.



(a) Brute-Force Fitting (Theoretical)



(b) Simple Forward Greedy (Theoretical)



(c) Experimental Decision Boundaries

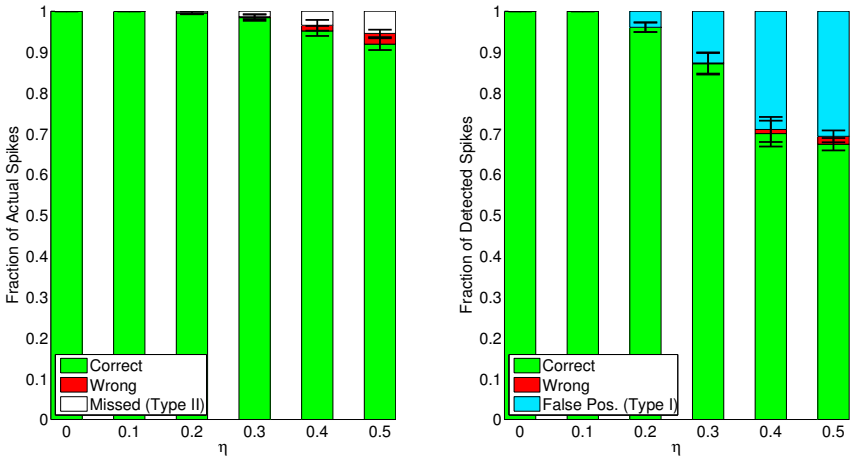
Figure 3.4: Decision boundaries for simple greedy if $\|\mathbf{f}_1 - \mathbf{f}_2\| > \|\mathbf{f}_1 + \mathbf{f}_2\|$ (condition for Lemma 3). Note that the decision boundaries differ between algorithms.

As a direct consequence of Lemma 2, if $\|\mathbf{f}_1 - \mathbf{f}_2\| \leq \|\mathbf{f}_1 + \mathbf{f}_2\|$, then the simple forward greedy algorithm and brute-force fitting will identify the same set of spike shapes.

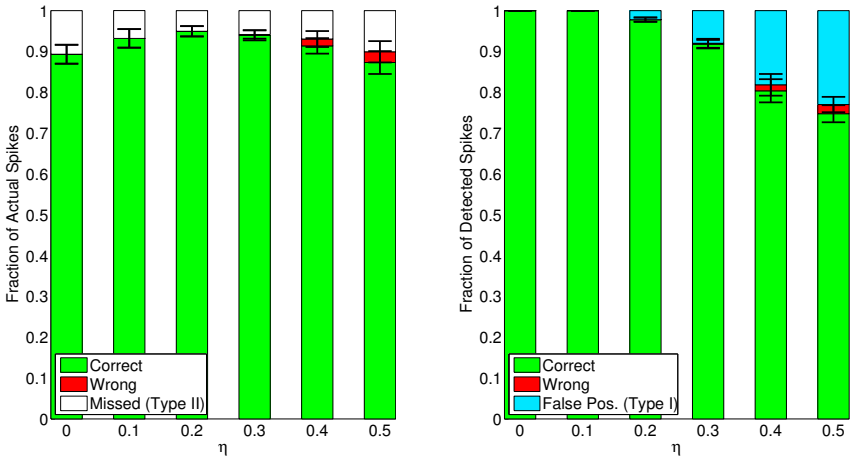
Lemma 3. *If $\|\boldsymbol{f}_1 - \boldsymbol{f}_2\| > \|\boldsymbol{f}_1 + \boldsymbol{f}_2\|$, then there are input signals for which the simple forward greedy algorithm does not yield the globally optimal solution.*

Proof. We derive the decision boundaries for the simple forward greedy algorithm using the same method as in the proof for Lemma 2. However, the key difference is that the condition $\|\mathbf{f}_1 - \mathbf{f}_2\| > \|\mathbf{f}_1 + \mathbf{f}_2\|$ results in an enlarged decision region for the origin at the first step of fitting (Figure 3.4). □

As a direct consequence of Lemma 3, if $\|\mathbf{f}_1 - \mathbf{f}_2\| > \|\mathbf{f}_1 + \mathbf{f}_2\|$, then the simple forward greedy algorithm will yield a higher false negative (miss) rate than brute-force fitting.



(a) Brute-Force Fitting



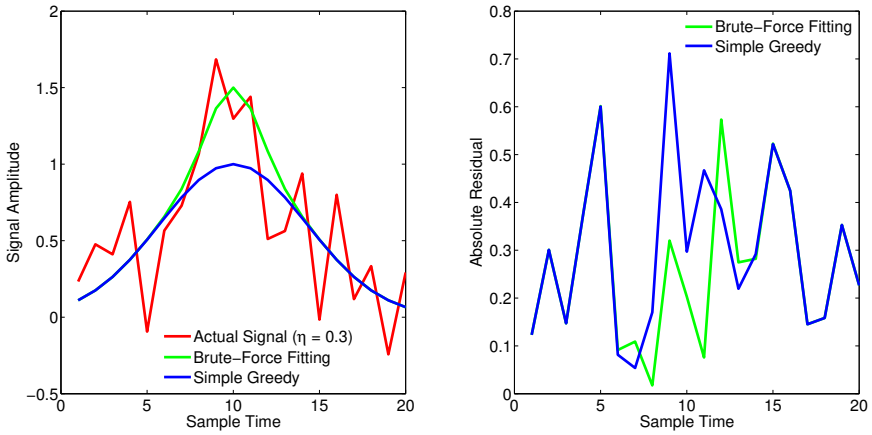
(b) Simple Forward Greedy

Figure 3.5: Error fractions for sorting spikes (no time shifts).

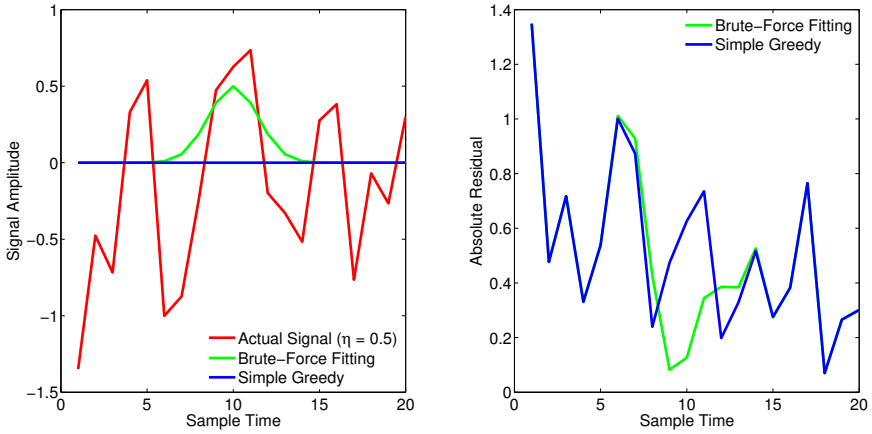
3.4 Results (No Time Shifts)

In this experiment, we used various combinations of the four synthesized spike shapes, all centered at $T = 10$, as they are shown in Figure 1.2. Figure 3.5 shows the results for 5×200 runs per noise level tested.

In Figure 3.5, we see that brute-force fitting has a higher false positive rate and a lower miss rate than simple greedy across all noise levels. Empirical investigation into these results (i.e. looking at a limited number of clips for which either algorithm fails) suggests that this discrepancy is due to brute-force detecting the combination $\mathbf{f}_{1+3}$ more often than simple greedy when $\mathbf{f}_{1+3}$ yields the minimal least squares residual; because simple greedy can only detect spikes one-by-one, there are cases in which it is unable



(a) Sample clip of simple greedy missing spikes $\mathbf{f}_{1+3}$. The actual signal consists of $\mathbf{f}_{1+2+3}$ with noise ($\eta = 0.3$), which brute-force fitting correctly detects. *Left:* Actual and detected spike signals. *Right:* Absolute residual between the detected and actual signals.



(b) Sample clip of brute-force overfitting noise to $\mathbf{f}_{1+3}$. The actual signal contains no spikes ($\eta = 0.5$), which simple greedy correctly identifies. *Left:* Actual and detected spike signals. *Right:* Absolute residual between the detected and actual signals.

Figure 3.6: Sample clips showing the difference in detection by Brute-Force Fitting vs. the Simple Greedy Algorithm.

to detect the pair $\mathbf{f}_{1+3}$. We propose that this results as a consequence of Lemma 3, the conditions for which $\mathbf{f}_1$ and $\mathbf{f}_3$ satisfy. For example, based on our empirical investigation, simple greedy seems more prone to incorrectly detecting $\mathbf{0}$ (instead of $\mathbf{f}_{1+3}$) at higher noise levels than brute-force fitting. The false positive and miss rate patterns then follow directly from these proposed trends. This suggests that in order to reduce simple greedy's missed rates, it may be helpful to have a greedy variant that also tests pairs of spikes.

Chapter 4

The Forward-Backward Greedy Algorithm

As we discussed briefly at the beginning of Chapter 3, the greedy algorithm can sometimes commit to the wrong path and thus fail to find the globally optimal solution. Consequently, it may be useful to have a variant of greedy that allows for some degree of bidirectionality.

For this purpose, Zhang [8] introduces the forward-backward greedy algorithm, which uses forward greedy steps to detect spikes and then adaptively applies backward greedy steps to remove potentially incorrectly detected spikes. In particular, the forward-backward greedy algorithm is designed to address the scenario shown in Figure 4.1 [8]. In this scenario, a signal vector $\mathbf{y}$, which is actually a linear combination of $\mathbf{f}_5$ and $\mathbf{f}_6$, is incorrectly fit to $\mathbf{f}_7$. Note this scenario resembles that in Figure 3.6a, in which simple greedy misses $\mathbf{f}_{1+3}$. According to Zhang [8], in this scenario, the forward-backward algorithm should act as follows: (1) forward greedy steps first detect $\mathbf{f}_7$ and then detect $\mathbf{f}_5$ and $\mathbf{f}_6$; (2) $\mathbf{f}_7$ is removed by a backward greedy step.

In this chapter, we show that the forward greedy algorithm as we have designed it (see Algorithm 1) fails at step (1) since it is unable to detect both $\mathbf{f}_7$ and $\mathbf{f}_{5+6}$. Based on outside literature [8], we then propose that continuous spike amplitude weighting may be required in order to allow the forward-backward algorithm to work as proposed.

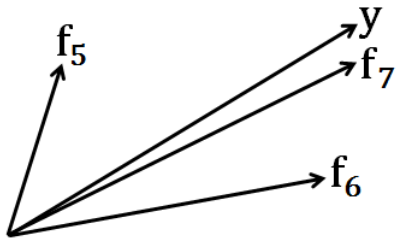


Figure 4.1: Model of the problem that the forward-backward algorithm is designed to solve.

4.1 The Backward Greedy Algorithm

As it serves as one of the basis algorithms for the forward-backward greedy algorithm, it is essential that we define the backward greedy algorithm. The backward greedy algorithm begins with all spikes detected and then optimally removes spikes, one-by-one (Algorithm 2).

Let $\mathbf{F}$ be an index set for the detected spike shapes. Then, the optimal spike to remove is $\mathbf{f}_I$, where the index I is given by

$$I = \underset{i \in \mathbf{F}}{\operatorname{argmin}} \left|\left| \mathbf{y} - \sum_{j \in \mathbf{F} - \{i\}} \mathbf{f}_j \right|\right|_2^2, \tag{4.1}$$

provided that

$$\left|\left| \mathbf{y} - \sum_{j \in \mathbf{F} - \{I\}} \mathbf{f}_j \right|\right|^2 \leq \|\mathbf{y}\|^2. \tag{4.2}$$

There are two notable consequences of the definition of the backward greedy algorithm. First, the backward greedy algorithm only makes sense for sorting spikes without time shifts; once time shifts are introduced, the concept of beginning with "all spikes detected" has no clear, logical definition.

Secondly, on its own, the backward greedy algorithm is an impractical approach to spike sorting. Just as for forward greedy, the computational effort for the backward greedy

Algorithm 2: Backward Greedy

Inputs : $\mathbf{y}$ = input signal vector, $\mathbf{f}_1, ..., \mathbf{f}_k$ = known spike types
Output : $\mathbf{F}$ = indices of detected spikes

$\mathbf{F} = \{1, \ldots, k\}$ # *begin will all spike types found*;
$\mathbf{Y}_F = \sum_{j \in \mathbf{F}} \mathbf{f}_j$ # *detected spike signal*;
$s = 0$;
while $s = 0$ **do**
  $J = \underset{j \in \mathbf{F}}{\operatorname{argmin}} \|\mathbf{Y}_F - \mathbf{f}_j - \mathbf{y}\|^2$ # *removing type J minimizes LSE*;
  $\delta_b = \|y\|^2 - \|\mathbf{Y}_F - \mathbf{f}_J - \mathbf{y}\|^2$ # *improvement*;

  if $\delta_b \geq 0$ **then**
    $F \leftarrow F - \{J\}$;
    $\mathbf{Y}_F \leftarrow \mathbf{Y}_F - \mathbf{f}_J$;
  else
    $s = 1$;
  end
end

algorithm is between $O(k)$ and $O(k^2)$. However, in practice, backward greedy is more likely to require $O(k^2)$ effort. Realistically, we expect that overlapping spikes contain at most 2-3 spikes. Thus, if k is large, then more iterations will be required to remove extraneous spikes (i.e. to implement backward greedy) than to add the necessary spikes (i.e. to implement forward greedy).

4.1.1 Decision Boundaries in the Two-Spike Case

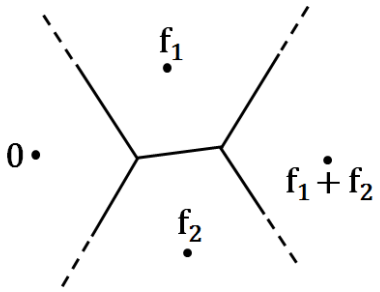
Using the same method as in §3.3, we derive decision boundaries to prove the following two lemmas.

Lemma 4. *If $\|\boldsymbol{f}_1 - \boldsymbol{f}_2\| \leq \|\boldsymbol{f}_1 + \boldsymbol{f}_2\|$, then the backward greedy algorithm yields the globally optimal solution.*

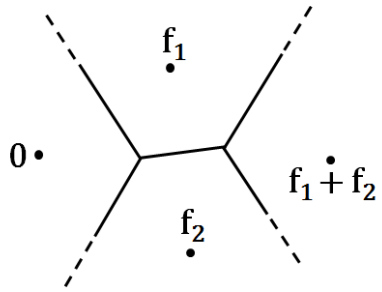
As a direct consequence of Lemmas 2 and 4, if $\|\mathbf{f}_1 - \mathbf{f}_2\| \leq \|\mathbf{f}_1 + \mathbf{f}_2\|$, then the backward greedy algorithm, simple forward greedy algorithm, and brute-force fitting will identify the same set of spike shapes.

Lemma 5. *If $\|\boldsymbol{f}_1 - \boldsymbol{f}_2\| > \|\boldsymbol{f}_1 + \boldsymbol{f}_2\|$, then there are input signals for which the backward greedy algorithm does not yield the globally optimal solution.*

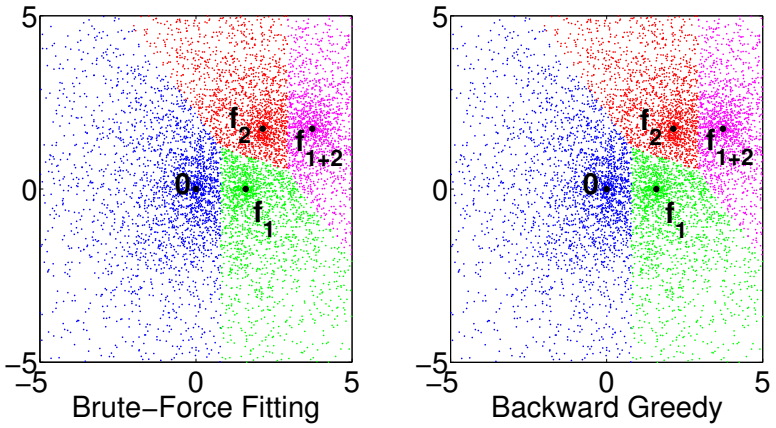
As a direct consequence of Lemma 5, if $\|\mathbf{f}_1 - \mathbf{f}_2\| > \|\mathbf{f}_1 + \mathbf{f}_2\|$, then the backward greedy algorithm will yield a higher false positive rate than brute-force fitting.



(a) Brute-Force Fitting (Theoretical)

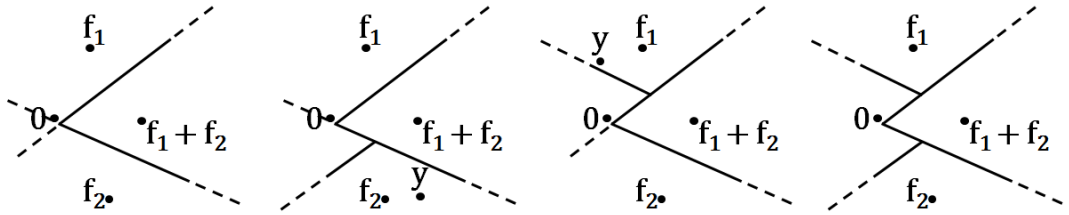


(b) Backward Greedy (Theoretical)



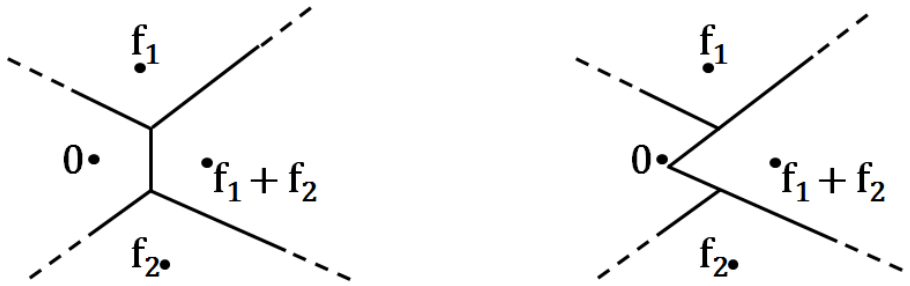
(c) Experimental Decision Boundaries

Figure 4.2: Decision boundaries for backward greedy if $\|\mathbf{f}_1 - \mathbf{f}_2\| \leq \|\mathbf{f}_1 + \mathbf{f}_2\|$ (condition for Lemma 4). Note that the decision boundaries are the same for both algorithms.

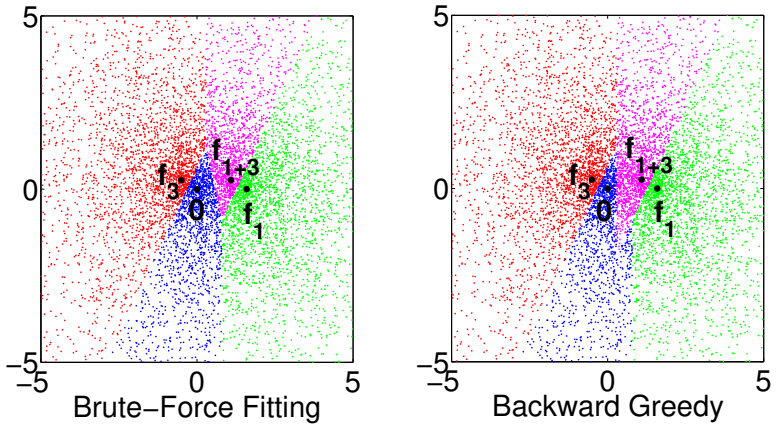


(a) Step 1 Boundaries (b) Detect $\mathbf{f}_2$ (c) Detect $\mathbf{f}_1$ (d) Overlay (b) and (c)

Figure 4.3: Deriving the decision boundaries for backward greedy if $\|\mathbf{f}_1 - \mathbf{f}_2\| > \|\mathbf{f}_1 + \mathbf{f}_2\|$ (condition for Lemma 5).



(a) Brute-Force Fitting (Theoretical) (b) Backward Greedy (Theoretical)



(c) Experimental Decision Boundaries

Figure 4.4: Decision boundaries for backward greedy if $\|\mathbf{f}_1 - \mathbf{f}_2\| > \|\mathbf{f}_1 + \mathbf{f}_2\|$ (condition for Lemma 5). Note that the decision boundaries differ between algorithms.

4.2 Defining the Forward-Backward Greedy Algorithm

The forward-backward greedy algorithm is implemented by using forward greedy steps (see Algorithm 1) to detect spikes one-by-one and then adaptively applying backward greedy steps (see Algorithm 2) to remove extraneously detected spikes. The optimal spike in the forward steps are defined identically as in the simple forward greedy model. Namely, the optimal spike to add is $\mathbf{f}_I$, where the index I is given by

$$I = \underset{i \in \{1,2,\ldots,k\} - F}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{f}_i\|_2^2, \tag{4.3}$$

provided that

$$\delta_f^{(I)} = \|\mathbf{y}\|^2 - \|\mathbf{y} - \mathbf{f}_I\|^2 \geq 0, \tag{4.4}$$

where $\delta_f^{(I)}$ is the improvement in adding $\mathbf{f}_I$ and F is the index set for the detected spike shapes. Then, the optimal spike to remove in the backward steps is $\mathbf{f}_J$, where the index J is given by

$$J = \underset{i \in \mathbf{F}}{\text{argmin}} \left|\left| \mathbf{y} - \sum_{j \in \mathbf{F} - \{i\}} \mathbf{f}_j \right|\right|_2^2, \tag{4.5}$$

Algorithm 3: Forward-Backward Greedy

Inputs : $\mathbf{y}$ = input signal vector, $\mathbf{f}_1, ..., \mathbf{f}_k$ = known spike types
Output : F = indices of detected spikes

```
ν = 0.5        # max LSE increase in backward step;
n = 1          # number of found spikes + 1;
s = 0;
while s = 0 do
    # forward step (detect spikes)
    I = argmin_{i∈{1,...,k}} ||y − f_i||^2      # detecting type I minimizes LSE;
    δ_f(n) = ||y||^2 − ||y − f_I||^2            # improvement from forward step;

    if δ_f(n) ≥ 0 then
        # accept spike
        F ← F ∪ I;
        y ← y − f_I;
        n ← n + 1;
    else
        s = 1;
    end

    # backward step (remove extraneously detected spikes)
    s_b = 0;
    while s_b = 0 do
        J = argmin_{j∈F} ||y + f_j||^2          # removing type J has min LSE;
        δ_b = ||y + f_J||^2 − ||y||^2           # error increase from removing type J;

        if δ_b ≤ ν · δ_f(n − 1) then
            # remove spike
            F ← F − {J};
            y ← y + f_J;
            n ← n − 1;
        else
            s_b = 1;
        end
    end
end
```

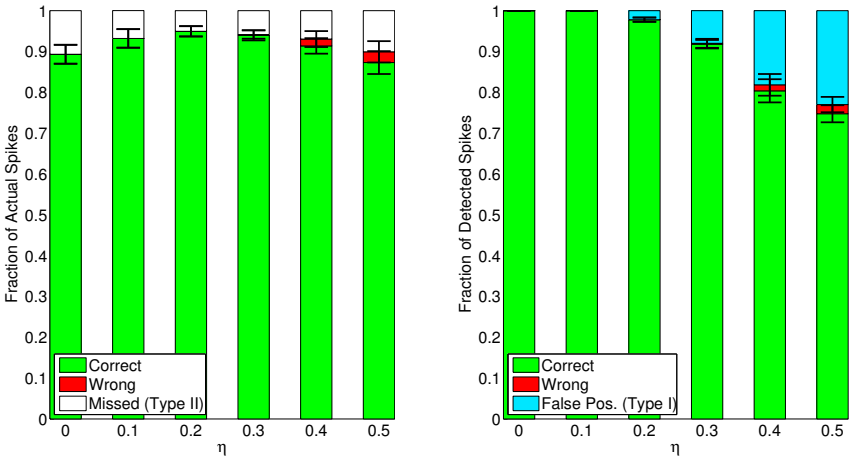
provided that

$$\delta_b^{(J)} = \left|\left| \mathbf{y} - \sum_{j \in \mathbf{F} - \{J\}} \mathbf{f}_j \right|\right|^2 - \|\mathbf{y}\|^2 \leq \delta_f^{(I)} \cdot \nu, \tag{4.6}$$

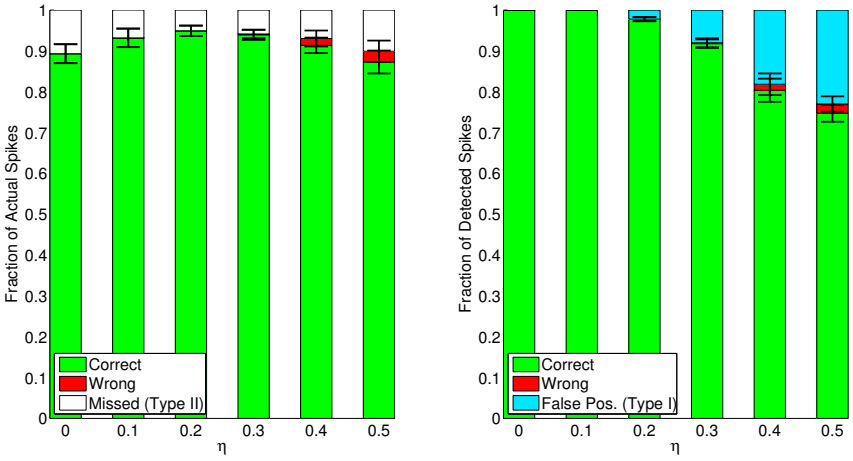
where $\delta_b^{(J)}$ is the increase in error from removing $\mathbf{f}_J$ and $\nu \geq 0$ is a parameter constraining the size of this increase. The efficiency of the forward-backward greedy algorithm is proven by Zhang [8].

4.3 Results

In Figure 4.5, we see that there is no difference in sorting by the simple greedy algorithm versus the forward-backward greedy algorithm. This is because the backward steps of the forward-backward greedy algorithm are never implemented. In the exam-



(a) Simple Forward Greedy



(b) Forward-Backward Greedy

Figure 4.5: Error fractions for spike sorting by forward greedy versus forward-backward greedy (no time shifts).

ple discussed at the beginning of this chapter (Figure 4.1), we showed that the desired result of the forward greedy steps is to first detect $\mathbf{f}_7$ and then detect $\mathbf{f}_{5+6}$. The main issue with this model is that this requires forward greedy to detect two spikes ($\mathbf{f}_{5+6}$) in one step, whereas our forward greedy algorithm can only detect one spike per step.

Based on the outside literature [8], we propose that introducing a parameter β will solve this issue, where β is a k-element vector of amplitude weights. In terms of our current models, the elements of β only take on the values 0 or 1. Our proposed solution is to allow the elements of β to take on values continuously on $[0, 1]$. Let $\mathscr{F}$ be a $k \times N$ matrix, such that the ith row is the spike type $\mathbf{f}_i$. Then, our generative model for a signal $\mathbf{y}$ is now given by

$$\mathbf{y} = \beta\mathscr{F} + \mathbf{H},$$

where $\mathbf{H}$ is a noise vector. By allowing the values of β to vary continuously, we propose that forward greedy will be able to overfit the data as desired. It is, however, a little unclear what the practical interpretation of these continuous β values would be.

Chapter 5

Time-Shifted Spikes

Up until now, we have been using combinations of spikes centered at the same sample time ($T = 10$). In reality, however, templates for known spike shapes may not necessarily be perfectly centered. Furthermore, the time shifts for overlapping spikes are generally unknown. Therefore, we introduce time shifts by allowing the firing times of the spikes in the actual signal to be real-valued and lying within the interval $[5, 15]$ (assuming a clip length of $N = 20$). Note that this interval does not span the entire clip due to the nature of how clips are chosen; it would be unrealistic for a clip to contain spikes that fell too far off either end.

In our sorting algorithms, we test discrete time shifts on the interval $[3, 17]$ with a grid spacing of $\Delta t = 2$. Using this grid spacing, we consider detected shifts $T_{det} = T_{act} \pm 1$ as correctly fitted. Although using a finer grid spacing would yield more accurate fitting, in practice, it is more important to identify the actual spike type than the exact time shift; thus, we reason that in favor of algorithm efficiency, using $\Delta t = 2$ is a reasonable assumption.

In terms of implementation, our sorting algorithms treat each time shift as a separate spike shape. Thus, given k spike types and n_s time shifts, our algorithms effectively operate on kn_s individual spike shapes. In this chapter, we see that time shifts not only increase the computational difficulty of the problem, but also introduce additional accuracy challenges.

5.1 Brute-Force Fitting with Time Shifts

Since we are treating each time shift as a separate spike shape, it may be tempting to say that brute-force fitting now faces a 2^{kn_s}-hard problem. However, because the refractory period prevents the same spike type from appearing more than once in a clip, we can disregard the combinations that include multiple time shifts of the same spike type. Therefore, the effort for brute-force scales as $O(n_s^k)$. Once again, we take the results of brute-force fitting to represent the globally optimal solution.

5.2 The Forward Greedy Algorithm with Pairs

Even without time shifts, our results hint that it may be advantageous not to restrict greedy fitting to testing spikes one-by-one (§3.4). Thus, especially given that introducing time shifts effectively increases the number of spike shape templates, it becomes beneficial to take a more exhaustive approach.

We define the forward greedy algorithm with pairs as a greedy algorithm that tests every single spike and every pair of spikes. It then follows that the effort for greedy with pairs is minimally $O(k^2 n_s^2)$ steps and maximally $O(k^3 n_s^2)$. With the exception of this modification, greedy with pairs is implemented identically as simple greedy, as demonstrated by the pseudocode outlined in Algorithm 4.

5.3 Results for Time-Shifted Spikes

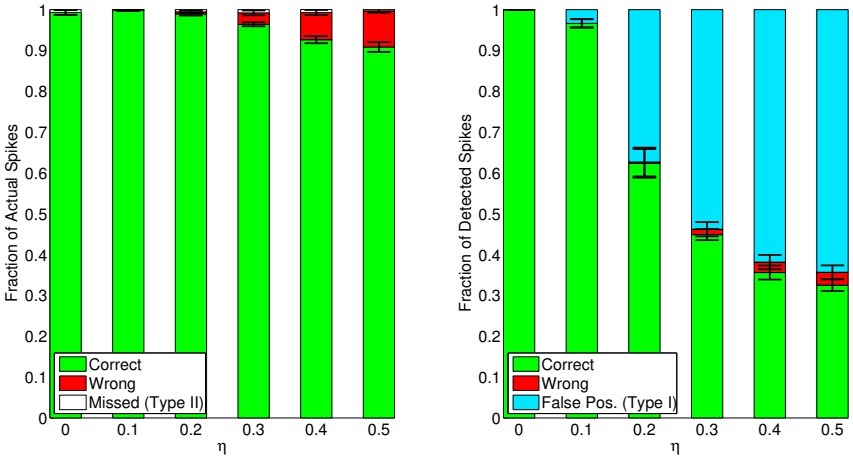
This experiment uses $k = 4$ and $n_s = 8$. As before, we use 5×200 runs per noise level (Figure 5.1). With time shifts, we see that at low noise levels, simple greedy yields the highest false positive rates, but at high noise levels, brute-force fitting yields the highest false positive rates.

Across all noise levels, simple greedy has the most wrong and the most missed. Meanwhile, brute-force fitting and greedy with pairs have similar missed and wrong rates across all η. This reflects the fact that simple greedy is the least exhaustive of the algorithms and that the addition of testing pairs allows greedy with pairs to achieve (by these measures) a more globally optimal fit.

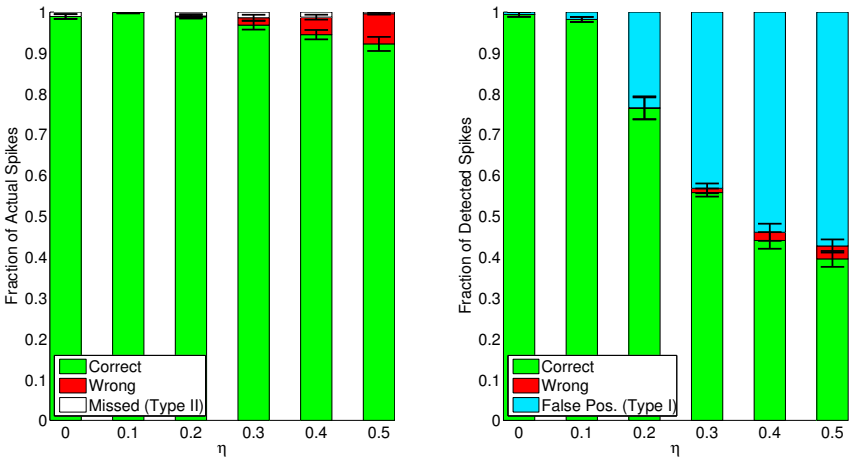
Algorithm 4: Greedy with Pairs

Inputs : $\mathbf{y}$ = input signal vector, $\mathbf{f}_1, ..., \mathbf{f}_k$ = known spike types
Output : $\mathbf{F}$ = indices of detected spikes

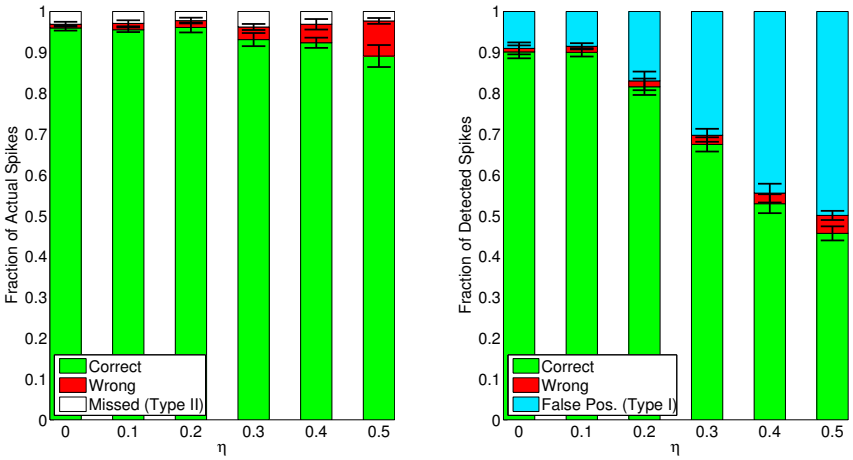
$A = \{1, \ldots, k, \underset{\substack{i,j \in \{1,\ldots,k\} \\ i \neq j}}{\{i, j\}}\}$ # *list of combinations to test*;
$s = 0$;
while $s = 0$ **do**
  $I = \underset{i \in A}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{f}_i\|^2$ # *index/indices of optimal spike/pair*;
  $\delta_f = \|\mathbf{y}\|^2 - \|\mathbf{y} - \mathbf{f}_I\|^2$ # *improvement*;
  if $\delta_f \geq 0$ **then**
    Remove any element of A containing any element of I;
    $\mathbf{F} \leftarrow \mathbf{F} \cup \{I\}$;
    $\mathbf{y} \leftarrow \mathbf{y} - \mathbf{f}_I$;
  else
    $s = 1$;
  end
end



(a) Brute-Force Fitting



(b) Foward Greedy with Pairs



(c) Simple Foward Greedy

Figure 5.1: Error fractions for sorting time-shifted spikes (no penalty).

The most notable difference between the results with time shifts (Figure 5.1) and those

Algorithm	Runtime (s)	Computational Effort
Brute-Force	315.4333	$O(n_s^k)$
Greedy with Pairs	25.2876	min = $O(k^2 n_s^2)$, max = $O(k^3 n_s^2)$
Simple Greedy	1.3049	min = $O(kn_s)$, max = $O(k^2 n_s)$

Table 5.1: Runtime and Computational Effort Scaling per Algorithm. The experimental runtimes represent those for 6000 runs ($k = 4$, $n_s = 8$).

without time shifts (Figure 3.5) is that with time shifts, the false positive rates are markedly higher. At $\eta \approx 0.2 - 0.3$, the false positive rates increase drastically for all three algorithms. This means that when the noise level is approximately 1/4 the signal amplitude, the algorithms become highly prone to fitting noise with spikes. In fact, with $\eta = 0.5$, we see that the false positive rates are as high as 0.6 with time shifts, versus only around 0.2 without time shifts. Based on empirical investigations into the incorrectly sorted clips, the results suggest that overfitting of the low firing spikes (i.e. $\mathbf{f}_4$, $\gamma_4 = 0.05$) significantly contributes to these inflated false positive rates. Clearly, we want a way to counter these effects.

5.4 Penalty for Detecting Multiple Spikes

As we observed in Figure 5.1, the introduction of time shifts causes a significant increase in false positive rates. This increase results from having n_s different possible ways to fit each spike type. Thus, the inclusion of n_s time shifts effectively lowers the significance level required to match a signal to any given spike. In order to counteract this effect, we introduce a penalty for detecting spikes.

Let F be the index set for the detected spike types, resulting from any sorting algorithm. Then, our generative model for $\mathbf{y}$ is given by

$$\mathbf{y} = \sum_{i \in F} f_i + \mathbf{H},$$

where $\mathbf{H} \sim \mathcal{N}(0, \eta^2 I_N)$ is a random noise vector. The likelihood that this model is correct is given by

$$L(\mathbf{y}|\mathbf{H}) = c \exp\left(-\frac{\|\mathbf{H}\|_2^2}{2\eta^2}\right),$$

where c is a normalization factor. In Chapter 2, we used the maximum likelihood to determine whether or not we detect a spike. In the following penalty derivation, we will show that using the maximal posterior probability yields a more optimal detection.

Remark. The penalty will include a factor of n_s as a Bonferroni correction. In our algorithms, we assume that each spike type can possibly take on n_s different time shifts.

Thus, we have a multiple comparisons problem. The Bonferroni correction compensates for these multiple hypotheses by decreasing their significance level by a factor of $\frac{1}{n_s}$ each.

Lemma 6. *The penalty for detecting spike shape i is given by*

$$\lambda_i = 2\eta^2 \log \frac{n_s(1-\gamma_i)}{\gamma_i}.$$

Derivation. Consider the detection of spike type $\mathbf{f}_i$. We detect $\mathbf{f}_i$ if $p(\mathbf{f}_i|\mathbf{y})$ is the maximal posterior probability such that

$$p(\mathbf{f}_i|\mathbf{y}) \geq p(\mathbf{0}|\mathbf{y}), \tag{5.1}$$

where $p(\mathbf{0}|\mathbf{y})$ is the posterior probability that signal vector $\mathbf{y}$ contains no spike. Recall that Bayes' Theorem [2] states that

$$p(\mathbf{f}_i|\mathbf{y}) = \frac{L(\mathbf{y}|\mathbf{f}_i)p(\mathbf{f}_i)}{p(\mathbf{y})}.$$

Since the expected firing rate of spike type $\mathbf{f}_i$ is γ_i, the prior probability $p(\mathbf{f}_i)$ that $\mathbf{f}_i$ is present is equivalent to

$$p(\mathbf{f}_i) = \frac{\gamma_i}{n_s} \prod_{\substack{j\in\{1,2,\ldots,k\}\\ j\neq i}} (1-\gamma_j). \tag{5.2}$$

Thus, we can rewrite Inequality 5.4 as

$$k \exp\left(-\frac{\|\mathbf{y}-\mathbf{f}_i\|^2}{2\eta^2}\right)\left(\frac{\gamma_i}{n_s}\right) \prod_{\substack{j\in\{1,2,\ldots,k\}\\ j\neq i}} (1-\gamma_j) \geq k \exp\left(-\frac{\|\mathbf{y}\|^2}{2\eta^2}\right) \prod_{j\in\{1,2,\ldots,k\}} (1-\gamma_j),$$

where $k = c/p(\mathbf{y})$. Cancelling like terms, we get that

$$\exp\left(-\frac{\|\mathbf{y}-\mathbf{f}_i\|^2}{2\eta^2}\right)\left(\frac{\gamma_i}{n_s}\right) \geq \exp\left(-\frac{\|\mathbf{y}\|^2}{2\eta^2}\right)(1-\gamma_i).$$

Taking the $-\log$ of both sides and then rearranging terms yields

$$\|\mathbf{y}\|^2 \geq \|\mathbf{y}-\mathbf{f}_i\|^2 + 2\eta^2 \log\left(\frac{n_s(1-\gamma_i)}{\gamma_i}\right).$$

Note that this is equivalent to Condition 3.2, but with the additional term

$$\lambda_i = 2\eta^2 \log\left(\frac{n_s(1-\gamma_i)}{\gamma_i}\right).$$

□

As a result of this derivation, the original conditions for the optimal spike $\mathbf{f}_I$ (Equations 3.1, 3.2) are replaced by

$$I = \underset{i \in \{1,2,\ldots,k\}-F}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{f}_i\|_2^2 + \lambda_i, \tag{5.3}$$

and

$$\|\mathbf{y} - \mathbf{f}_I\|\|^2 + \lambda_I \leq \|\mathbf{y}\|^2. \tag{5.4}$$

Note that if $\eta = 0$ or if $n_s(1 - \gamma_i) = \gamma_i$, then this is the same as having no penalty ($\lambda_i = 0$).

Figure 5.2 shows that the addition of the penalty does indeed reduce the false positive rate, as intended. However, the addition of the penalty also causes a fairly significant increase in the miss rates. This implies that there is a compromise to be made between these two error types. In practice, it will be up to the user to determine which of these error types is less desirable at higher noise levels.

5.5 Results with Penalty

Figure 5.3 shows that the addition of the penalty significantly reduces the false positive rates, so that they are comparable to those without time shifts. As noted at the end of the previous section, the addition of the penalty also causes an increase in the miss rates.

Generally speaking, simple greedy does not yield particularly accurate results. At low noise levels, it still has high false positive rates relative to the other algorithms. Additionally, the miss rates for simple greedy are also higher than those for brute-force at all noise levels. This confirms that we need an alternative to simple greedy, especially when sorting time-shifted spikes.

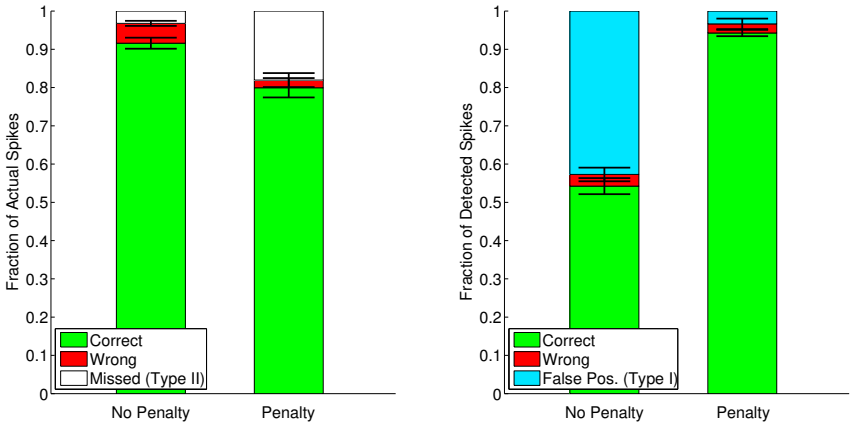
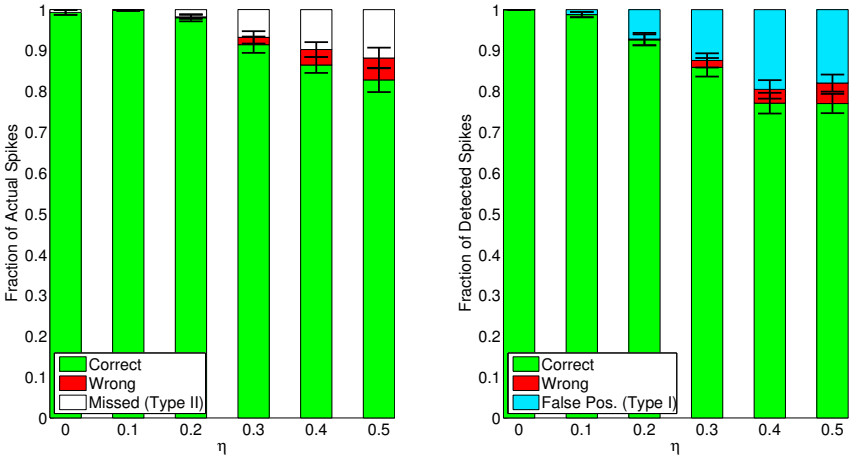
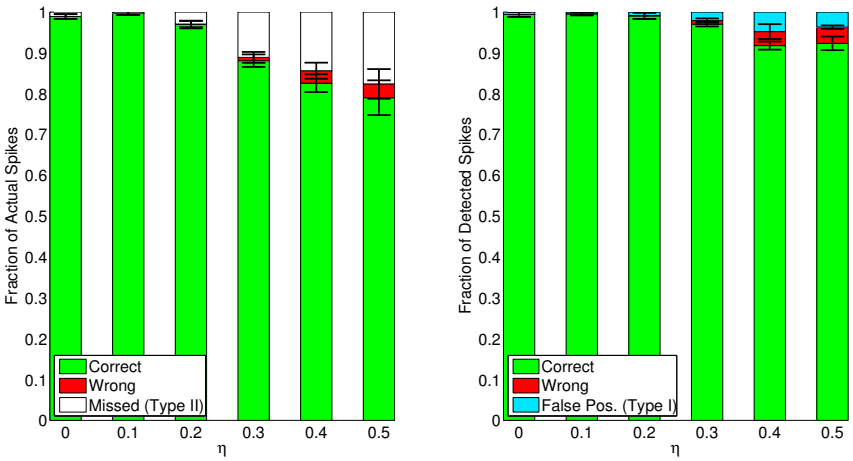


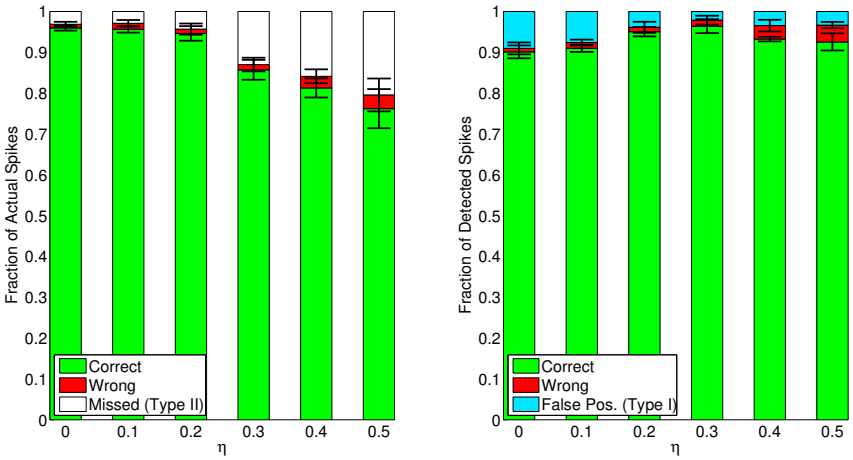
Figure 5.2: Comparing error fractions with and without penalty. These results represent those for simple greedy with and without penalty, using $\eta = 0.4$.



(a) Brute-Force Fitting



(b) Foward Greedy with Pairs



(c) Simple Foward Greedy

Figure 5.3: Error fractions for sorting time-shifted spikes (with penalty).

Fortunately, across all error fractions, greedy with pairs outperforms simple greedy.

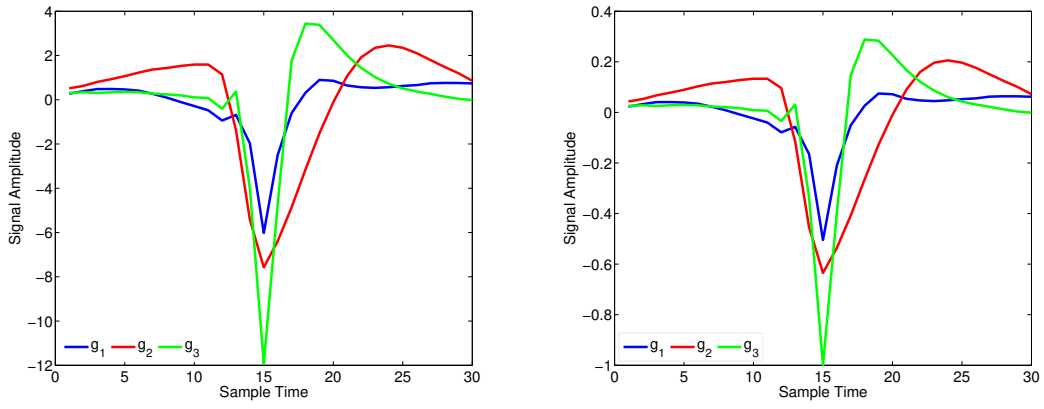
This shows that greedy with pairs is a more promising solution for sorting overlapping spikes with time shifts than simple greedy. Compared to brute-force, greedy with pairs performs similarly at lower noise levels ($\eta \leq 0.2$), but has higher miss rates and lower false positive rates at higher noise levels ($\eta > 0.2$). The former observation confirms that greedy with pairs approximates the globally optimal solution pretty well; the latter observation simply reiterates that there is a compromise to be made between the miss and false positive rates.

Chapter 6

Real Data

In the previous chapter, we established that greedy with pairs shows promise in sorting synthetic spikes. In order to fully justify its practicality, we need to assess its performance using real data. In this chapter, we analyze one channel of data from Harris, et al [3]. First, we assess the sortability of the real spike types by sorting synthetic clips containing combinations of time shifts of the real spike shapes. Next, we run our algorithms on real clips and discuss ways to assess algorithm accuracy.

To prepare the data for experimentation, we use band pass filtering to visualize the spikes. From the filtered signal, we extract clips ($N = 30$) and extrapolate three template spike types. Next, we normalize the data and templates by dividing by the maximum amplitude of the templates (Figure 6.1). This normalization ensures that the noise level η can still be interpreted as $1/SNR$. Since we obviously don't have generating functions for the templates, we can only test integral time shifts. Specifically, our algorithms test time shifts on the interval $[9, 21]$ with a grid spacing of $\Delta t = 2$. Additionally, our algorithms assume that $\gamma_i = 0.5$ for $i = 1, 2, 3$.



(a) Raw Templates

(b) Normalized by Amplitude

Figure 6.1: Templates for real spike types $\mathbf{g}_1$, $\mathbf{g}_2$, and $\mathbf{g}_3$.

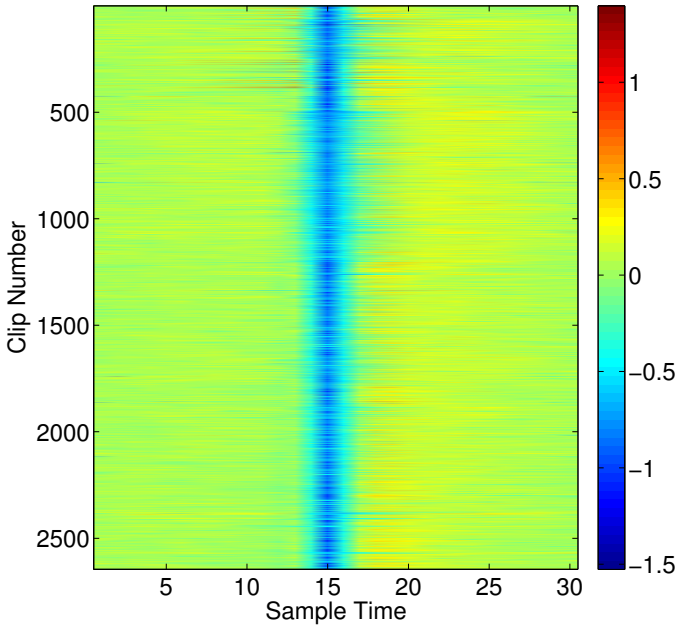


Figure 6.2: Color Map of the Real Clips

6.1 Sorting Synthesized Clips using Real Spike Shapes

We synthesize clips containing combinations of the real spike with time shifts on the interval $[11, 19]$ with a grid space of $\Delta t = 1$. Note that this interval was chosen to resemble those of the real clips (Figure 6.2).

Figure 6.4 shows the results for sorting these synthetic clips. Since the real clips are filtered, the contribution of noise can be assumed to be relatively low. Thus, for our synthetic clips, we focus on the results at low noise levels. At low noise levels, we

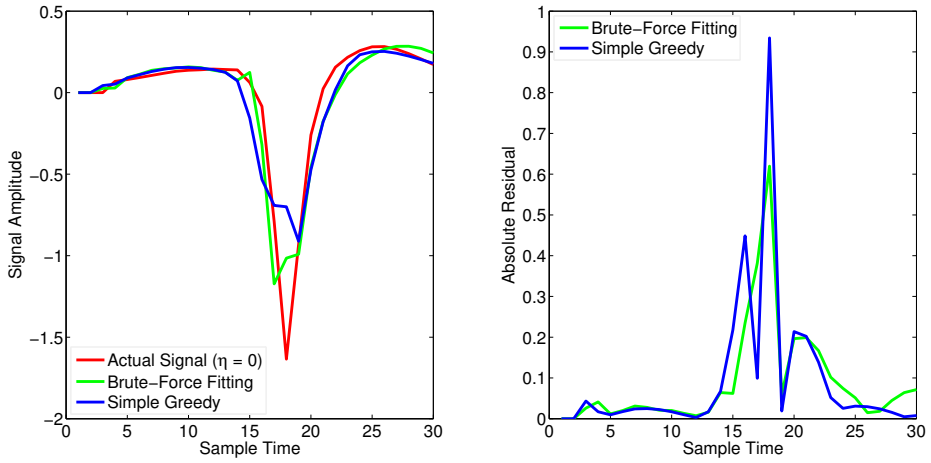
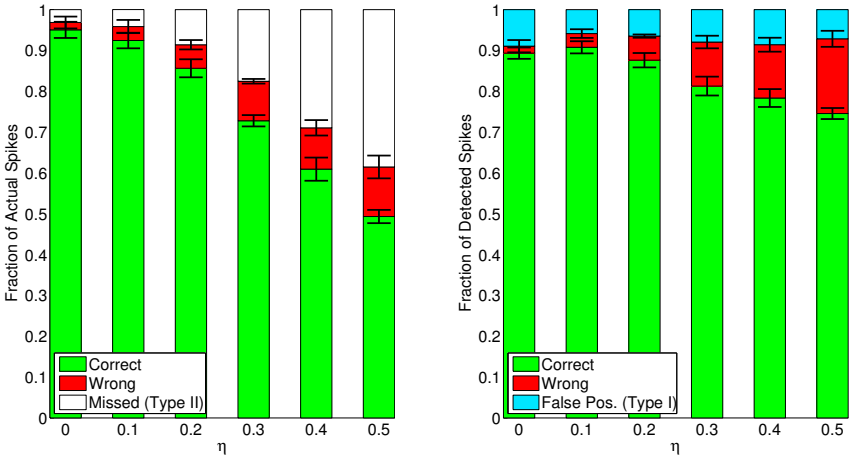
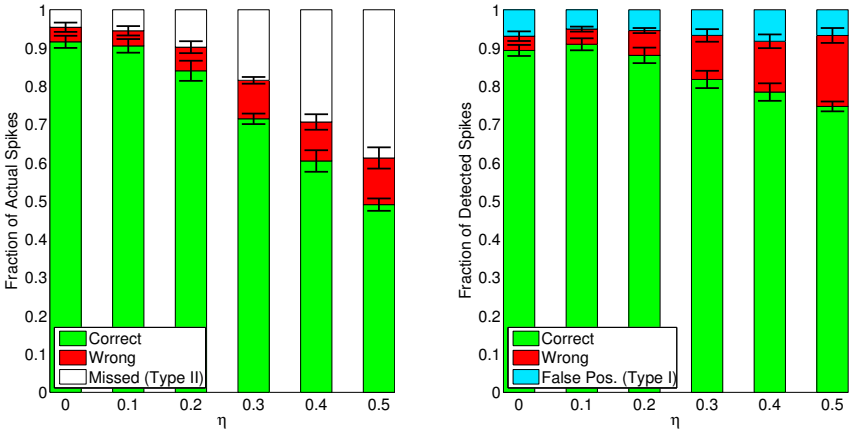


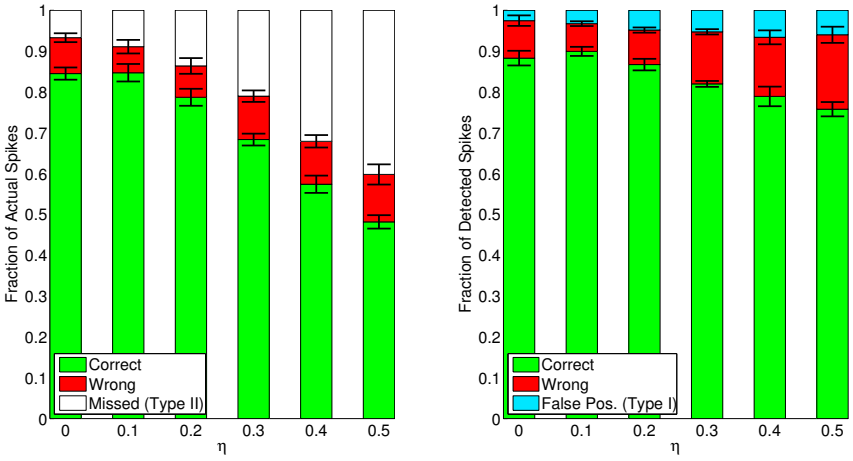
Figure 6.3: Sample clip for which simple greedy and brute-force fail. The actual signal contains $\mathbf{g}_2$ ($T = 18$) and $\mathbf{g}_3$ ($T = 18$) with $\eta = 0$. Simple greedy detects $\mathbf{g}_1$ ($T = 19$) and $\mathbf{g}_2$ ($T = 17$). Brute-force detects $\mathbf{g}_1$ ($T = 19$), $\mathbf{g}_2$ ($T = 19$), and $\mathbf{g}_3$ ($T = 17$).



(a) Brute-Force Fitting



(b) Greedy with Pairs



(c) Simple Forward Greedy

Figure 6.4: Error fractions for sorting real spike types with time shifts.

observe that brute-force fitting has the highest false positive rates and that simple greedy yields the highest wrong rates.

As shown in Figure 6.3, it is observed that brute-force occasionally overfits signals. Simple greedy, meanwhile, tends to incorrectly identify $\mathbf{g}_3$ as either $\mathbf{g}_1$ or $\mathbf{g}_2$ (with no empirical tendency toward one type over the other).

In general, at low noise levels, the error fractions are comparable to those for our synthetic spike shapes. Thus, we conclude that these spike shapes are reasonably sortable and do not present a problem too difficult to solve.

6.2 Sorting Real Clips

Here, we sort the real clips from Harris, et al [3]. Since we filtered the signal, we assume that there is little to no noise within the clips. For comparison, we run the algorithms first using $\eta = 0$ (i.e. no penalty) and then using $\eta = 0.2$ (i.e. with penalty). The latter value was chosen by looking at the signal amplitude at the ends of the clips (see Figure 6.2). Note that these clips are suspected to contain relatively few overlapping spikes, and hence, are probably not the ideal data set for our purposes.

The main challenge in sorting real clips is that there is no ground truth metric for measuring algorithm accuracy. To address this problem, we take two approaches.

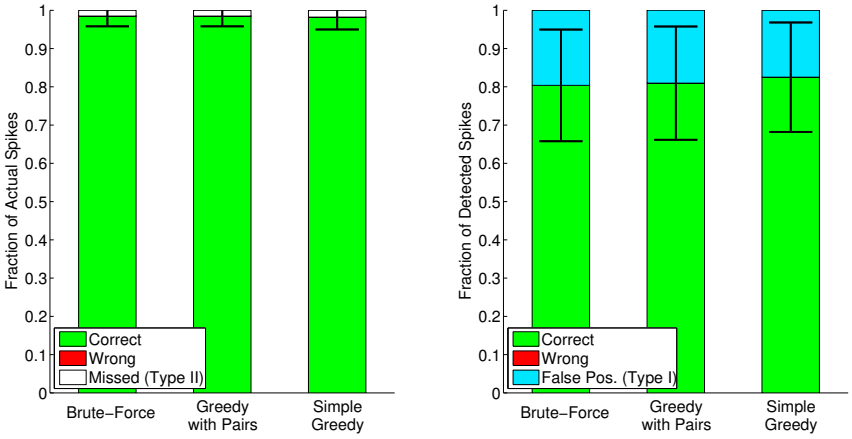
First, we look specifically at the detection of spike type $\mathbf{g}_2$. Here, we have an actual ground truth for the presence of $\mathbf{g}_2$ in each clip, which was acquired via patch clamp recordings. Patch clamp recordings are intracellular recordings in which electrodes are attached to individual neurons [7]. Thus, we can take the results of patch clamp recordings as the ground truth for a given spike type. These recordings, by nature, do not allow us to determine the presence of overlapping spikes.

Second, we compare the results of our algorithms against those of another algorithm called ISO-SPLIT [5]. It is important to note that ISO-SPLIT is unable to detect overlapping spikes.

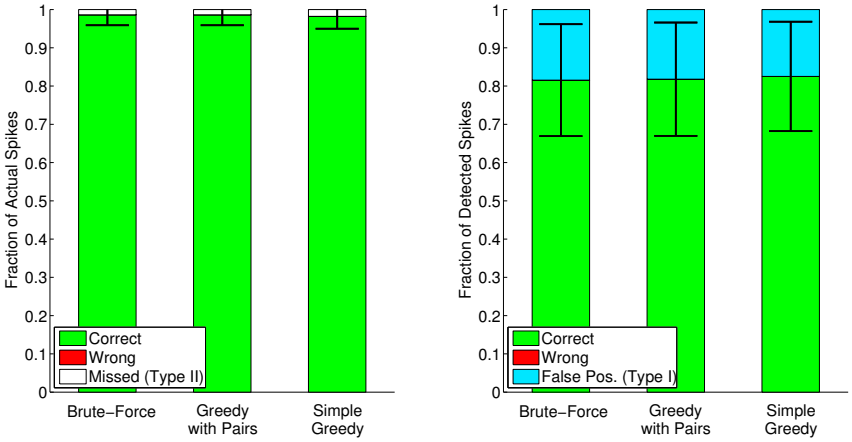
6.2.1 Results: Comparing to Patch Clamp Recordings

Figure 6.5 shows how well our algorithms detect spike type $\mathbf{g}_2$. We note that using $\eta = 0$ versus using $\eta = 0.2$ does not significantly affect the error fractions. Note that there is no wrong rates as this analysis only considers the detection of $\mathbf{g}_2$ and disregards the detection of all other spike types. Here, the false positive rates represent how often the algorithms detect $\mathbf{g}_2$ when it is not actually present.

We see that all three algorithms overfit $\mathbf{g}_2$ at approximately the same rates. In §6.1, we observed that brute-force has a tendency to overfit the data using these real spike types. Thus, it is not surprising that we observe a relatively high false positive rate; what is surprising, however, is that with penalty ($\eta = 0.2$), there is no significant effect on the false positive rate for brute-force. In §6.1, we also observed that simple greedy tends to misidentify $\mathbf{g}_3$ as $\mathbf{g}_2$ or $\mathbf{g}_1$. Thus, some of these misidentifications could contribute to an inflated detection rate for $\mathbf{g}_2$.



(a) Assuming $\eta = 0$.



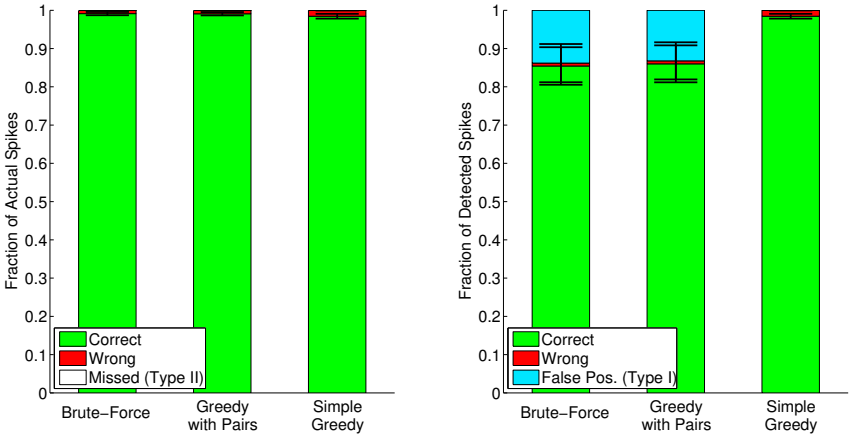
(b) Assuming $\eta = 0.2$.

Figure 6.5: Algorithm accuracy in detecting real spike type g_2. The ground truth is determined via patch clamp recordings.

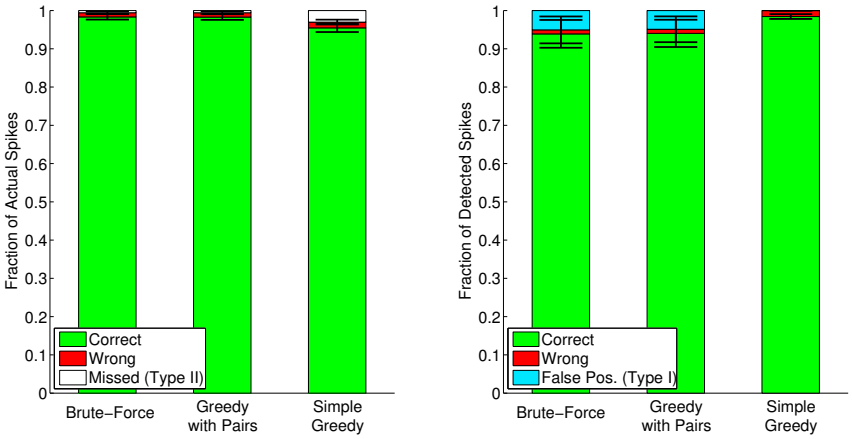
6.2.2 Results: Comparative Algorithm Performance

Since these results represent the comparative sorting performance by different algorithms, we modify our interpretations of the error fractions accordingly. In particular, the correct rates now represent spikes on which the algorithms agree. The wrong rates represent spike identities on which the algorithms disagree. We interpret the miss rates as ISO-SPLIT detecting a spike when our algorithms detect none. Finally, we interpret the false positive rates as our algorithms detecting more spikes in a clip than ISO-SPLIT. Consequently, we expect that the false positive rates represent a combination of actual false positives and overlapping spikes.

Figure 6.6 shows that our algorithms agree pretty well with ISO-SPLIT; notably, all the correct rates are ≥ 0.8 and the wrong rates are approximately 0. For the most part, the miss rates for all three algorithms are also approximately 0, meaning that our algorithms and the outside algorithm generally agree on which clips are empty or nonempty. This shows that our algorithms detect spikes reasonably well.



(a) Assuming $\eta = 0$.



(b) Assuming $\eta = 0.2$.

Figure 6.6: Comparative algorithm performance in sorting real clips. Error fractions are calculated using the results of ISO-SPLIT (an algorithm that cannot detect overlapping spikes) as the ground truth.

The addition of the penalty (Figure 6.6b) reduces the false positive rates, as we expect. In Figure 6.4, we observe that at $\eta = 0$, the false positive rates are ~ 0.1 for brute-force and greedy with pairs, and ~ 0.02 for simple greedy. At $\eta = 0.2$, the false positive rates are ~ 0.05 for all three algorithms. Detracting these values from the false positive rates in Figure 6.6 and taking into account the error bars, we consider the the remaining false positive rates to be almost negligible. Thus, as we suspected, we conclude that these clips contain few to no overlapping spikes.

Chapter 7

Further Work

Our first target for future work is to adjust the forward-backward greedy algorithm so that it works as intended. Namely, we would introduce a real-valued β parameter, where β is a vector of spike amplitude weights. Successful implementation of this β parameter would determine whether or not the forward-backward greedy algorithm is worth pursuing as a spike sorting algorithm.

In general, it would be prudent to have more robust analyses of where our algorithms fail within our toy models. Because of time constraints, we were only able to look at a few sample clips to infer which spike types a given algorithm misses or overfits. Ideally, a less empirical approach would give us a better idea of what our algorithms can handle.

Our models are highly contingent on the assumptions that noise is iid Gaussian and that spike shapes can be approximated using Gaussians. It would be interesting to implement our algorithms so that they can handle non-Gaussian data. Namely, we would like to adjust our algorithms so that they are able to handle correlated, non-Gaussian noise as well as non-Gaussian spike shapes.

Finally, the real data analyzed in this paper contained few (to no) overlapping spikes. Ideally, we would like to run our algorithms on data with a higher frequency of overlapping spikes, as this would be the true test of the practicality of our algorithms. Additionally, we would like to investigate better ways to analyze the accuracy in sorting real clips, given that ground truth metrics for real overlapping spikes do not exist.

Chapter 8

Conclusions

Without time shifts, we showed that simple greedy is an efficient and relatively accurate algorithm for spike sorting. However, by only allowing simple greedy to detect one spike at a time, it can sometimes miss a more optimal spike combination. This suggests a need for a more exhaustive algorithm. To this end, we explored the forward-backward greedy algorithm and a greedy with pairs algorithm.

We found that in our current implementation, the forward-backward greedy algorithm is identical to simple greedy. We proposed that introducing a parameter for amplitude weights may fix this result, but due to time constraints, were unable to implement this change here.

We then looked at a greedy with pairs algorithm. We observed that with the introduction of time shifts, greedy with pairs generally outperforms simple greedy across all error fraction measures. Additionally, we observed that greedy with pairs is less efficient than simple greedy, but is not unreasonably less so. Namely, greedy with pairs is still significantly more efficient than brute-force fitting. Thus, we conclude that greedy with pairs represents a reasonable compromise between the accuracy of brute-force fitting and the efficiency of simple greedy.

The inclusion of time shifts introduces a multiple comparisons problem. Thus, we observed that with higher noise levels, the false positive rates for all three algorithms increases drastically. The introduction of a penalty for spike detection effectively reduces the false positive rates to those without time shifts. However, the penalty also increases the miss rates at higher noise levels. This suggests that in practice, either these higher noise levels represent problems that are too difficult to solve or that the user must choose which error rate is less desirable and apply the penalty correspondingly.

Finally, we were not able to draw any significant conclusions from our results from sorting real clips since we used a data set included very few overlapping spikes. However, using the real data, we were able to determine that our algorithms perform reasonably well in comparison to other established algorithms. We also determined that our algorithms fairly accurately detect single spikes. However, perhaps a different data set will

yield more illuminating results as to our algorithms' accuracy in sorting overlapping spikes.

Code (in MATLAB) for all of the sorting algorithms discussed in this paper can be found at `https://github.com/pxchen95/overlappingspikes.git`.

References

[1] A. H. Barnett, J. F. Magland, and L. F. Greengard. Validation of neural spike sorting algorithms without ground-truth information. *J. Neurosci. Methods*, 264:65–77, 2016.

[2] C. M. Grinstead and J. L. Snell. *Introduction to Probability*. Amer. Math. Soc., 2nd edition, 2006.

[3] K. D. Harris, D. A. Henze, J. Csicsvari, H. Hirase, and G. Buzsaki. Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *J. Neurophysiol.*, 84:401–414, 2000.

[4] M. Lewicki. Bayesian modeling and classification of neural signals. *Neural Comp.*, 6:1005–1030, 1994.

[5] J. F. Magland and A. H. Barnett. Unimodal clustering using isotonic regression: Iso-split. pages 1–23, 2016.

[6] R. Q. Quiroga. Spike sorting. *Scholarpedia*, 2(12):3583, 2007.

[7] H. G. Rey, C. Pedreira, and R. Q. Quiroga. Past, present and future of spike sorting techniques. *Brain Res. Bull.*, 119:106–117, 2015.

[8] T. Zhang. Adaptive Forward-Backward Greedy Algorithm for Learning Sparse Representations. *IEEE Trans. Inf. Theory*, 57(7):4689–4708, 2011.