Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Clasificación rápida de neuronas mediante "Super-paramagnetic Clustering"

Alumno: Christian Nicolás Heitman
L.U. 275/03
cnheitman@dc.uba.ar

Director: Dr. Diego Fernández Slezak
Departamento de Computación
Universidad de Buenos Aires

Co-Director: Dr. Matías Ison
Departamento de Ingeniería
Universidad de Leicester

Marzo de 2012

Tesis para optar al grado de
Licenciado en Ciencias de la Computación

Clasificación rápida de neuronas mediante "Super-paramagnetic Clustering"

Uno de los temas más estudiados de la neurociencia es desentrañar cómo se representa la información en el cerebro a través de la actividad neuronal. Hoy en día es posible medir la actividad de decenas de neuronas mediante registros extra-celulares *in vivo*, utilizando sistemas de adquisición de información con múltiples canales simultáneos.

A fin de poder llegar a conclusiones a partir de la gran cantidad de datos registrados, es necesario poder identificar y separar la actividad individual de cada neurona con la mayor precisión y rapidez posible. Los métodos para llevar a cabo esto se los conoce como *Ordenamiento de Espigas*.

En esta tesis se presenta un método de *Ordenamiento de Espigas on-line* basado en Quian Quiroga *et al.* El objetivo es realizar la clasificación de las espigas simultáneamente con el experimento, *i.e.*, al momento de ser registradas por los aparatos de medición.

La principal limitación del método de Quian Quiroga *et al.* está en el algoritmo de clasificación, *Super-paramagnetic Clustering*, pues necesita contar con todos los datos antes de poder aplicarse. Se realizaron modificaciones al algoritmo para garantizar la adecuación del mismo al procesamiento *on-line*. Fue necesario realizar una implementación eficiente del mismo para atacar los requisitos de temporales del problema.

Palabras claves: Neurociencias, Actividad neuronal, Ordenamiento de Espigas, Procesamiento *on-line*, Super-paramagnetic Clustering.

Fast neuron classification using
"Super-paramagnetic Clustering"

One of the most studied topics in neuroscience is to unravel how information is represented in the brain through neural activity. Nowadays is possible to measure the activity of tens of neurons by *in vivo* extra-cellular recordings using acquisition systems with multiple channels of information simultaneously.

In order to draw conclusions from the vast amount of data recorded is necessary to identify and separate the individual activity of each neuron as fast and accurately as possible. Methods for accomplishing this are known by the name of *Spike Sorting*.

This thesis presents an *on-line Spike Sorting* method based on Quian Quiroga *et al.* The aim is to perform the classification of the spikes simultaneously with the experiment, *i.e.*, at the same time they are registered by measuring devices.

The main limitation of the method of Quian Quiroga *et al.* is in the classification algorithm, *Super-paramagnetic Clustering*. All information have to be gathered before it can be applied. The algorithm was modified to ensure the suitability of on-line processing. It was necessary to make an efficient implementation in order to meet the temporal requirements of the problem.

Keywords: Neuroscience, Neural Activity, Spike Sorting, On-line processing, Super-paramagnetic Clustering.

Agradecimientos

Quiero agradecer mis directores, Diego y Matías, por la buena voluntad, tiempo, dedicación y ayuda. También a los jurados, Alejandro Furfaro y Alejo Salles por tomarse el tiempo de leer este trabajo. A mi vieja, por bancarme siempre. A Flor, por todo el apoyo y tantas otras cosas. A mis amigos de siempre. A los de la facu, con lo que transité este camino: Manix, Cesaro, Lea, Arti, El Pocho, Fran, Pachi, Nigel, Germán, Guille, David, Matías, Eze, Lucio, Marta, Gabo, Daniel y Wiwi.

A todos, Gracias!

A mis abuelos.

Índice general

Capítulo 1

Introducción

Uno de los temas más estudiados de la neurociencia es desentrañar cómo se representa la información en el cerebro a través de la actividad neuronal. En estos últimos años las técnicas de medición han avanzado de manera vertiginosa. Hoy en día es posible medir la actividad de decenas de neuronas mediante registros extra-celulares *in vivo*. En animales, estas técnicas tienen décadas de existencia [1, 2]. Actualmente, también se aplican en seres humanos, por ejemplo, en pacientes con epilepsia imposible de tratar farmacológicamente [3], utilizando sistemas de adquisición de información con múltiples canales simultáneos [4]. Estos avances han ayudado a la comprensión de la formación de conceptos y de estructuras de procesamiento en el pensamiento humano [3], y abren una puerta hacia las *Interfases Computadora-Humano* (BCI).

Con estos avances surgen nuevos retos. A fin de poder analizar y llegar a conclusiones a partir de los datos registrados, es necesario poder identificar y separar la actividad individual de cada neurona con la mayor precisión posible. Los métodos utilizados deben ser cada vez más confiables y eficientes. Actualmente, aplicar estos métodos sobre una gran cantidad de datos es computacionalmente muy costoso, tornandosé imposible de hacerlo durante el transcurso de un experimento.

1.1. Medición de la actividad neuronal

Las neuronas se comunican con otras a través de impulsos eléctricos de muy corta duración llamados *potenciales de acción* o *espigas*, que son el producto de estímulos externos o bien espontáneos. La figura 1.1 muestra una representación de dos neuronas comunicándose. Como se puede observar, las neuronas están compuestas por: el *soma* (cuerpo de la célula), las *dendritas* y el *axón*. Los estímulos son captados por las dendritas y, si son suficientemente fuertes, se genera un impulso eléctrico que viaja a través del axón de la neurona emisora hasta la neurona más cercana.

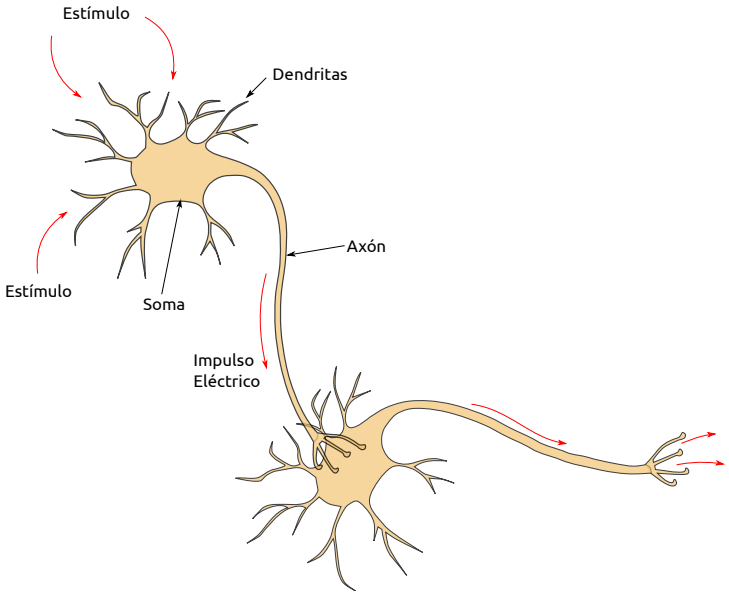


Figura 1.1: Representación esquemática de dos neuronas comunicándose.

En la figura 1.2 se muestra la forma característica de un potencial de acción. Al recibir el estímulo adecuado, se produce un incremento rápido en el voltaje eléctrico, que luego de un tiempo decae y vuelve a su nivel normal (potencial en reposo). Si el estímulo es insuficiente, el incremento es muy leve y no llega a *activar* a la neurona.

Los potenciales de acción se pueden registrar a través de microelectrodos situados en el cerebro [1]. Dependiendo de la posición de estos, se pueden registrar potenciales de acción de una o más neuronas. Tomando como ejemplo la figura 1.1, el microelectrodo debería ubicarse el área donde el axón de

la neurona emisora se encuentra con las dentritas de la neurona receptora.

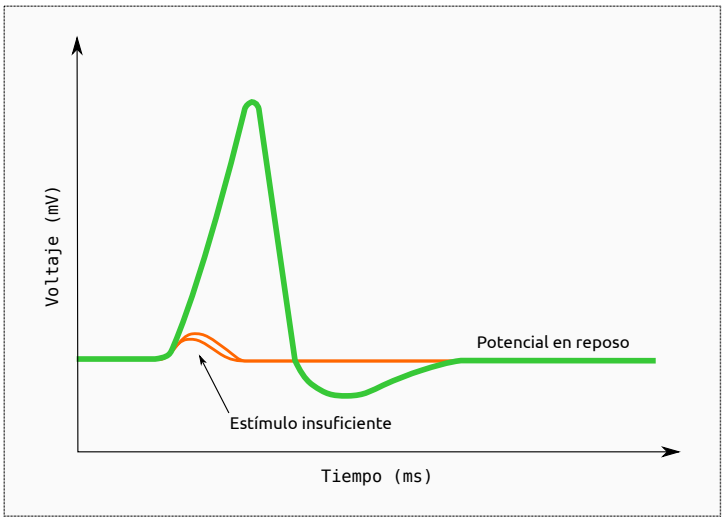


Figura 1.2: Representación esquemática del potencial de acción de una neurona.

En la figura 1.3 se muestra un esquema de cómo se realiza la medición. El microelectrodo se ubica en la región de interés para el experimento a realizar, cercano a las neuronas cuya actividad se desea medir. Como la intensidad del impulso eléctrico decae rápidamente a medida que nos alejamos de las neuronas que lo emiten, solo aquellas que están más próximas a la cabeza del electrodo, a menos de 50 µm [4], son las que se podrán distinguir del ruido de fondo. Por ejemplo, en la figura 1.3, las neuronas que se miden son las dos que se encuentran dentro del círculo más pequeño. La actividad de las neuronas que se encuentran a una distancia media, aproximadamente entre 50 y 150 µm, como ser aquellas que están dentro del círculo del medio, también es capturada. Sin embargo, en este caso no se pueden diferenciar entre sí debido al efecto del ruido. Es decir, están lo suficientemente cercanas como para que su actividad sea registrada por el microelectrodo pero no tanto como para poder diferenciarlas unas de las otras. A esto se la denomina *actividad multi-unidad*. Finalmente, las espigas de las neuronas que se encuentran más lejos aún (dentro del círculo exterior en la figura), no pueden ser detectadas y estas forman parte del ruido de fondo de la señal.

En muchos casos se dificulta capturar la actividad de una única neurona, debido a diversos motivos, por ejemplo, el ruido de fondo o a que hay otras

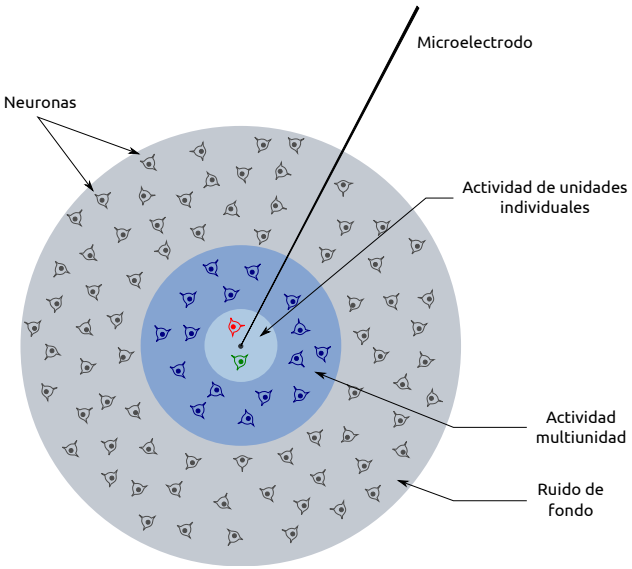


Figura 1.3: Esquema de registro extra-celular *in vivo*

neuronas cercanas a aquella cuyas espigas son similares. Se han desarrollado algoritmos para poder separar las distintas espigas que componen una señal registrada, conocidos bajo el nombre de *Ordenamiento de Espigas*.

1.2. Ordenamiento de Espigas

Los microelectrodos capturan la actividad eléctrica proveniente de una o más neuronas cercanas. Esta señal es muy débil y se confunde con el ruido generado por la actividad neuronal lejana. Para analizar propiedades de neuronas individuales a partir de datos registrados, primero es necesario separar las espigas del ruido de fondo de la señal. Por otro lado, distintas neuronas (o tipos de neuronas) muestran distintos patrones de potencial de acción por lo que resulta muy importante la clasificación y separación de las espigas por su forma. Este proceso se lo conoce como *Ordenamiento de Espigas*.

Los procesos complejos dentro del cerebro son el producto de la interacción de grandes poblaciones de neuronas, y el estudio de neuronas aisladas solo da una visión muy limitada de todo el panorama [4]. La capacidad actual de poder registrar simultáneamente grandes cantidades de neuronas

brindan un camino muy interesante para poder estudiar estos procesos. Sin embargo, a fin de obtener información útil de esta enorme cantidad de datos, es necesario contar con técnicas de ordenamiento de espigas confiables. A través de ellas se pueden estudiar cómo son los patrones de interacción de neuronas cercanas. Asimismo, es posible analizar la actividad de neuronas de bajo nivel de activación que, por lo general, al no poder ser correctamente clasificadas, se confunden entre las neuronas de alto nivel de activación. Estas neuronas, llamadas *neuronas silenciosas*, no se disparan hasta que se les presenta el estímulo "adecuado" y su correcta identificación es de difícil solución.

Por lo tanto, las técnicas de ordenamiento de espigas resultan de mucha utilidad en el proceso de analizar y comprender los principios del procesamiento neuronal.

1.3. Método de Quian Quiroga et al.

En el último tiempo han surgido métodos novedosos de ordenamiento de espigas que mejoran el rendimiento frente a los métodos clásicos, i.e. *PCA* [5] para la caracterización y *K-Means Clustering* [6] para la clasificación.

Uno de los nuevos métodos más utilizados es el propuesto por Quian Quiroga *et al.* [7]. Se basa en la utilización de la *transformada wavelet* [8] para la extracción de características de las espigas [9, 10, 11], y en *Super-paramagnetic Clustering* (SPC) [12, 13] para el agrupamiento de las mismas. Una de sus virtudes más destacables es que es *no supervisado*.

Super-paramagnetic Clustering es un algoritmo de agrupamiento de datos relativamente nuevo, que tiene su raíz en la física estadística [12, 13]. El problema del agrupamiento de datos se puede enunciar de la siguiente manera: dado N puntos en un espacio métrico de dimensión D se propone establecer la partición de estos en M grupos de manera tal que los puntos que quedan dentro de un grupo sean más *similares* entre sí que los que quedan dentro de otro. Entre las propiedades más destacables de este algoritmo, se pueden mencionar que no requiere ninguna premisa sobre la distribución de los datos, ni la ausencia de solapamiento entre los agrupamientos.

Este método mejoró significativamente la separación de señales de neuronas individuales, sin embargo, es computacionalmente muy costosa para un número elevado de neuronas. Tampoco permite el procesamiento *on-line* de los datos registrados, altamente deseable dentro del desarrollo de un experimento, ya que permite la adaptación en función de la retroalimentación de los datos.

Una de las limitaciones más importantes para llevar el método de Quian Quiroga *et al.* [7] al mundo del procesamiento *on-line* está en la etapa de *clasificación*. Más precisamente en el algoritmo SPC, puesto que éste necesita contar de antemano con todos los puntos a agrupar antes de poder empezar. Es decir, carece de la posibilidad de agregar un nuevo punto a agrupar al conjunto de puntos ya agrupados. Actualmente, incorporar un nuevo punto implica ejecutar nuevamente el algoritmo. Por lo tanto, para poder procesar los datos de un experimento es necesario que el mismo haya finalizado.

1.4. Objetivos de esta tesis

En esta tesis se presenta un método de *Ordenamiento de Espigas on-line* basado en Quian Quiroga *et al.* [7]. El objetivo es realizar la clasificación de las espigas simultáneamente con el experimento, *i.e.*, al momento de ser registradas por los aparatos de medición.

Como se mencionó, la principal limitación para llevar a cabo este objetivo radica en el algoritmo SPC [12, 13]. Éste no permite, a partir de un conjunto de puntos ya agrupados, agregar un nuevo punto y retornar un nuevo mapa de agrupamiento.

Se realizó un estudio detallado del algoritmo y se procedió a su modificación para garantizar la adecuación del mismo al procesamiento *on-line*, dotándolo con la capacidad de incorporar nuevos datos a agrupar de manera dinámica y obteniendo en cada oportunidad el mapa de agrupamiento de los datos procesados al momento. Además de realizar las modificaciones, fue necesario realizar una implementación eficiente del nuevo algoritmo de modo de alcanzar el objetivo propuesto.

A fin de poder determinar la *calidad* de los agrupamientos y la *velo-*

cidad de procesamiento del nuevo algoritmo, se realizaron pruebas sobre varios registros de actividad neuronal, tanto sintéticos como reales. Los datos sintéticos son generados a través de simulaciones que otorgan un alto grado de realismo y sus características se asemejan estrechamente a datos de registros reales [14]. Se utilizaron conjuntos de datos de distintos tamaños y con diferentes niveles de ruido, de modo de que las pruebas fueran lo más abarcadoras posible. Por último, a modo de ejemplo, se mostró el comportamiento sobre un conjunto de datos reales.

Capítulo 2

Ordenamiento de Espigas

Hay varios algoritmos de ordenamiento de espigas, de diferentes complejidades y efectividad. Uno de los más simples se basa en la *discriminación por amplitud*. En este, una espiga se separa de otra utilizando un umbral en la amplitud de la señal. De esta manera, si el pico o punto máximo de una espiga es mayor al de otra, se puede situar un umbral suficientemente alto de modo de "aislar" a aquella espiga que lo supere. En la figura 2.1 se puede observar un ejemplo de separación utilizando este método. Esta técnica es sencilla y suficientemente rápida para ser implementada de manera *on-line*. Sin embargo, cuenta con varias desventajas. La principal es que puede darse el caso de que dos espigas distintas tengan la misma amplitud pero diferente forma. En este caso es imposible separar, mediante este método, una de la otra. La otra dificultad que presenta es cómo establecer el valor del umbral. Muchas veces debe realizarse de forma manual, lo cuál torna al método impracticable en caso de contar con muchos canales de grabación.

Otro método de separación se basa en la *discriminación por ventanas*. En este caso, se fija una (o más) ventana (rectángulo) de base en el tiempo y altura en la amplitud. Todas aquellas espigas que pasen dentro de la ventana son asignadas a una misma clase o grupo. Este método también es suficientemente directo y rápido para ser implementado de manera *on-line*, pero tiene la desventaja de que las ventanas se deben establecer de forma manual e incluso pueden llegar a cambiar durante el experimento. Otro inconveniente que presenta es que, en ciertos casos, dos espigas pueden tener formas muy similares pudiendo llegar a dificultar bastante encontrar una

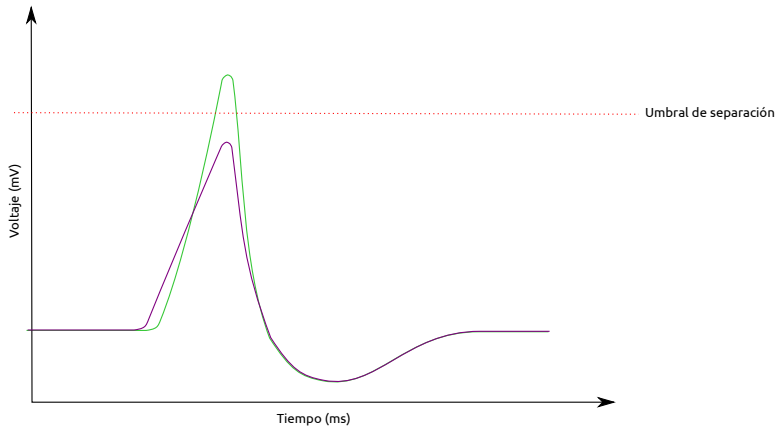


Figura 2.1: Ejemplo de discriminación por amplitud.

ventana que separe a ambas de manera correcta. Por último, es probable que se pierdan aquellas espigas de neuronas que se disparan a un ritmo muy menor que el resto. En la figura 2.2 se muestra un ejemplo de esta técnica.

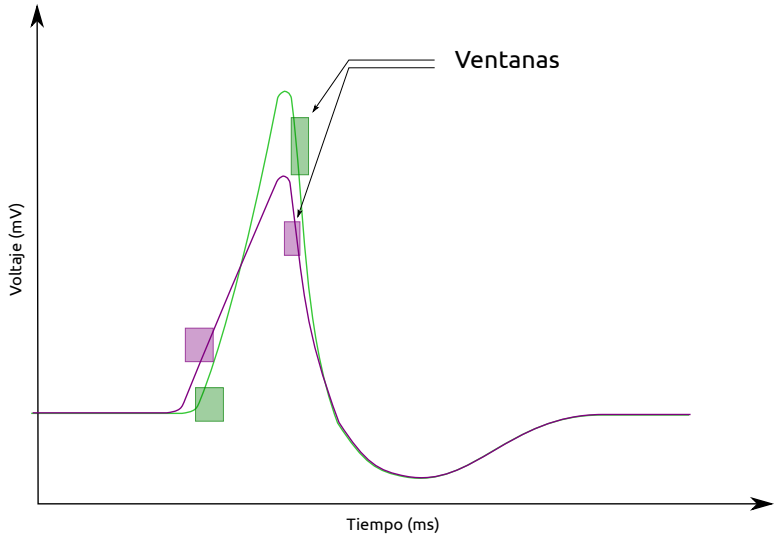


Figura 2.2: Ejemplo de discriminación por ventanas.

Otra de las técnicas se basa en elegir una espiga característica para cada grupo y, a través de estas, separar las demás espigas utilizando *Template Matching*: una espiga se asigna al grupo cuya distancia media cuadrada a la espiga característica sea menor. Este método también se puede implementar de manera *on-line*. Sin embargo, como desventajas podemos mencionar que es supervisado, con lo cuál tiene que haber una persona que diga cuáles son

las espigas características. Esto se torna impracticable cuando hay varios canales simultáneos de grabación. Otro inconveniente es que las espigas características deben ser reajustadas durante el transcurso del experimento. Por último, cuando muchas espigas se solapan, se dificulta cuáles espigas características elegir. A su vez, sufre del mismo problema que las técnicas anteriores, la imposibilidad de capturar las neuronas con bajo nivel de activación.

2.1. Estructura general de un algoritmo de ordenamiento de espigas

En general, los algoritmos de ordenamiento de espigas están compuestos por 4 etapas, a saber:

1. *Filtrado de la señal*
2. *Detección de espigas*
3. *Extracción de características de las espigas*
4. *Clasificación de las espigas*

2.1.1. Filtrado de la señal

El primer paso antes de poder empezar con la detección de las espigas es realizar un filtrado de la señal captada por los instrumentos de medición. Este paso es necesario para descartar información que es irrelevante. La elección del rango de frecuencias a filtrar está en relación a la duración de una espiga, que puede ir aproximadamente desde 0,5 ms a $2,5$ ms. Comúnmente, el rango se extiende un poco más para evitar *aliasing*. El filtro utilizado para realizar esto debe ser elegido con atención puesto que podría llegar a distorsionar la señal (por lo general, es mejor emplear un filtro no causal).

2.1.2. Detección de espigas

Por lo general, la detección de las espigas se realiza estableciendo un umbral en la amplitud de la señal. De esta manera, se consideran espigas, todos aquellos puntos que superen dicho umbral, que puede ser fijado tanto de manera manual como automática. Nuevamente, hay que tener mucho cuidado al elegir el valor del umbral, uno muy alto puede hacer que se descarten espigas mientras que uno muy bajo puede hacer que se confunda parte del ruido de la señal por espigas válidas.

Luego de realizar la detección, las espigas tienen que ser almacenadas para su posterior clasificación. Uno de los problemas a resolver es saber cuántas muestras de la señal elegir para representarlas. Para hacer esto hay que tener en cuenta la velocidad de muestreo con que fue capturada la señal original y la longitud promedio de una espiga que es, aproximadamente, 2 ms.

Una vez elegida la cantidad de muestras con las que se representan a las espigas, falta ver cómo compararlas entre sí a fin de poder clasificarlas en diferentes grupos o clases. Para poder realizar esto, es necesario alinearlas de modo que la comparación sea posible. Por lo general, todas las espigas se centran en su punto máximo, de modo de que este se encuentre siempre en la misma muestra para todas las espigas. Por ejemplo, si cada espiga se representa con 100 muestras, se ubica el máximo de cada una de ellas en la muestra 50. Sin embargo, el proceso de alineación no es directo. Puede ocurrir que el máximo de las espigas no quede registrado en la señal. Es decir, que haya una "desincronización" entre estos máximos y la velocidad de muestreo. Una de las maneras de solucionar este problema es interpolando las muestras de las espigas y alinear la versión interpolada en vez de la original. Posteriormente, se submuestrea la espiga interpolada y se guardan la cantidad necesaria de muestras.

2.1.3. Extracción de características

En esta etapa se eligen las características que mejor representan a cada espigas y que, además, mejor diferencian una clase de espiga de las demás. Otro de los objetivos de esta etapa es reducir la dimensionalidad de la re-

presentación de las espigas. Esto último en muchos casos es necesario dado que la mayoría de los algoritmos de agrupamiento de datos no soportan un número alto de dimensiones o, si lo hacen, tiene un costo computacional elevado.

Hay varios acercamientos a esta etapa, con diferentes niveles de complejidad. Entre los más simples se encuentran tomar la amplitud entre el pico máximo y mínimo de cada espiga, el "ancho" del pico, la energía de la espiga, etc. Otros más complejos involucran obtener las componentes principales de la señal y utilizar esos coeficientes para representar a cada una.

La correcta elección de las características de las espigas es primordial para obtener un buen resultado final. Si estas no definen de manera clara cada una de las distintas clases, la fase final de agrupamiento difícilmente arrojará el resultado correcto.

2.1.4. Clasificación

Finalmente se encuentra el proceso de clasificación. Esta etapa consiste de agrupar a las espigas de características similares en un mismo grupo. De esta manera se puede distinguir la actividad de las distintas neuronas que se midieron.

Llegado a este punto, el problema se transforma en uno de agrupamiento de datos. Se tiene una cantidad N de puntos en un espacio D-dimensional y el objetivo es encontrar uno o más grupos, donde los puntos que se hallan dentro de ellos sean más similares entre sí que aquellos de grupos diferentes. Esto se basa fuertemente, en que en la etapa anterior se hayan extraído lo mejor posible las características de cada una de las espigas. Por lo tanto, si dos espigas son muy similares entre sí, sus respectivas características, representadas por un vector de dimensión D, deberían ser muy parecidos. Por lo tanto, el algoritmo de agrupamiento los pondría dentro de un mismo grupo.

2.2. Método de Quian Quiroga *et al.*

El método propuesto por Quian Quiroga *et al.* [7] es un método de ordenamiento de espigas cuya principal ventaja, además de su muy buen desempeño, es que es *no supervisado*. Se basa en la *transformada wavelet* [8] para la caracterización de las espigas y en *super-paramagnetic clustering* [12, 13] para la clasificación de las mismas.

Este método se puede separa en las etapas antes comentadas: (1) filtrado, (2) detección, (3) extracción y (4) clasificación. A continuación se profundizará en cada una de ellas.

2.2.1. Filtrado y Detección

En primer lugar, se filtra la señal de modo de quedarse solo con las frecuencia dentro del rango de 300 a 3000 Hz, mediante el filtro de Butterworth de orden 4 [15].

Luego, se obtiene el valor del umbral (Thr) a utilizar para realizar la detección de una espiga. Éste se calcula de manera automática y está dado por la siguiente fórmula:

$$Thr = 4\sigma_n; \quad \sigma_n = mediana\left\{\frac{x}{0{,}6745}\right\} \tag{2.1}$$

donde x representa la señal filtrada y σ_n es una estimación de la desviación estándar del ruido de fondo.

Aplicando el umbral Thr sobre la amplitud de la señal filtrada, la detección se realiza de manera automática.

Hay otras opciones a la hora de elegir el umbral. Por ejemplo, se podría considerar la desviación estándar de la señal para obtener un umbral, sin embargo, esto hace que sea muy elevado para los casos donde hay espigas con una gran amplitud y alto nivel de activación. De esta manera, no se estarían detectando la totalidad de las espigas. En cambio, el uso de la mediana mitiga este problema [7].

Por último, para cada espiga detectada, se toman 64 muestras de la

señal que representan, aproximadamente, 2,5 ms (la longitud promedio de una espiga). Para calcular la cantidad de muestras se tiene en cuenta la velocidad de muestreo de la señal, en este caso 24 kHz.

Finalmente, todas las espigas son alineadas de modo que su máximo valor quede en la muestra número 20. Para evitar cualquier tipo de inconveniente al obtener el máximo, se interpolan 256 muestras de la señal utilizando splines cúbicos.

2.2.2. Extracción de características

Esta es una etapa decisiva dentro del algoritmo. La manera con que se caracterice a cada espiga impacta de manera directa en el resultado final. Una de la novedades de este método particular es la utilización de la transformada wavelet para obtener los rasgos principales de cada espiga.

La transformada wavelet da una representación de la señal en componentes tiempo-frecuencia, con resolución óptima en ambos dominios (tiempo y frecuencia).

La transformada se define como la convolución entre la señal $x(t)$ y una función wavelet $\psi_{a,b}(t)$, es decir:

$$W_\psi X(a,b) = \langle x(t) \mid \psi_{a,b}(t) \rangle \tag{2.2}$$

donde $\psi_{a,b}(t)$ se obtiene a partir del escalamiento y translación de una función wavelet madre $\psi(t)$ y se define de la siguiente manera:

$$\psi_{a,b}(t) = |a|^{-1/2} \psi \left(\frac{t-b}{a} \right) \tag{2.3}$$

los valores a y b, son los factores de escalamiento y translación, respectivamente. En su versión discreta, estos valores se definen de la siguiente forma: $a = 2^j$ y $b = 2^{-j}k$ con $j, k \in \mathbb{Z}$. Las escalas más pequeñas dan información de la señal sobre las frecuencias más altas mientras que las escalas más grandes lo hacen sobre las frecuencias más bajas. Asimismo, la translación da información en tiempos diferentes de la señal.

Aplicando la transformada a diferentes tiempos y escalas, y organizando el resultado jerárquicamente, se obtiene una *descomposición multi-resolución* de la señal [8]. Para este caso concreto, se utiliza la wavelet Haar con una descomposición de 4 niveles.

Por lo tanto, la aplicación de la transformada wavelet a cada una de las espigas otorga una buena caracterización de las mismas.

Al aplicar la transformada a cada una de las espigas, representadas por 64 muestras, obtenemos un vector, también de longitud 64, pero de coeficientes wavelet. Esto, en principio, nos da una descomposición de la señal, la cuál representa bastante bien los rasgos de cada espigas. Sin embargo, no termina acá el proceso de extracción de características.

De estos coeficientes se busca extraer un subconjunto de 10 valores que mejor representen las características y que, a su vez, mejor diferencien una clase de espiga de otra. De esta manera, se obtienen los mejores resultados en la etapa de clasificación. Surge entonces, la pregunta de cuál es la mejor manera de realizar la elección. La misma se basa en la siguiente idea: si hay espigas que pertenecen a dos clases diferentes entonces esto se ve reflejado en los coeficientes wavelet (puesto que estos nos brindan una buena descomposición tiempo-frecuencial), por lo tanto, la distribución de los coeficientes debe ser multi-modal. Es por esto, que se aplica un test de normalidad a los coeficientes y de este modo se eligen aquellos que más se desvían de una distribución normal.

El test utilizado en este caso es la modificación de Lilliefors del test de Kolmogorov-Smirnov [16]. Dado un conjunto de datos x, se compara la función de distribución acumulada de los datos, $F(x)$, contra la función de una distribución gaussiana con la misma media y varianza $G(x)$. La desviación de la normalidad, por lo tanto, queda representada por:

$$\max(|F(x) - G(x)|) \tag{2.4}$$

2.2.3. Clasificación

Por último, se clasifican las espigas agrupando los vectores característicos de cada una ellas obtenidos en el paso anterior mediante *Super-paramagnetic*

Clustering, que permite la separación automática de las diferentes clases de espigas sin necesitar de ningún supuesto sobre la distribución de las mismas.

Super-paramagnetic clustering es un algoritmo de agrupamiento de datos relativamente nuevo, que tiene su raíz en la física estadística [12]. El problema del agrupamiento de datos se puede enunciar de la siguiente manera: dado N puntos en un espacio métrico de dimensión D se propone establecer la partición de estos en M grupos de manera tal que los puntos que quedan dentro de un grupo sean más *similares* entre sí que los que quedan dentro de otro. Para lograr esto, el algoritmo SPC se basa en las propiedades físicas de un sistema ferromagnético. Se modela el sistema utilizando el modelo de Potts [17], en el cual a cada punto se le asigna un spin, y se introduce una interacción entre puntos vecinos cuya fuerza disminuye en función de la distancia entre ellos. Luego se realiza una simulación del sistema a través de un algoritmo de Monte Carlo para diferentes temperaturas. El sistema muestra 3 fases. A bajas temperaturas el sistema se encuentra totalmente ordenado (fase *ferromagnética*), es decir, todos los spin están alineados. En cambio, a altas temperaturas el sistema no muestra ningún orden (fase *paramagnética*). A temperaturas intermedias emergen grupos de puntos con el mismo spin (fase *super-paramagnética*). A través de este rasgo se puede determinar qué puntos pertenecen a un mismo grupo.

Entre las propiedades más destacables de este algoritmo, se pueden mencionar que no requiere ninguna premisa sobre la distribución de los datos, ni la ausencia de solapamiento entre los agrupamientos.

En la próximo capítulo se detallará en profundidad el algoritmo SPC.

2.2.4. *Wave-Clus*

A continuación se presenta una implementación del método: *Wave-Clus*. Este es un programa de Matlab que puede ser descargado desde: `http://www2.le.ac.uk/departments/engineering/research/bioengineering/neuroengineering-lab/wave-clus-docs/wave-clus.rar`

La figura 2.3 muestra la pantalla principal del programa. Una vez elegido el archivo que contiene la señal, el programa comienza a procesarla. Se pueden ver las distintas espigas que componen la señal, la cantidad que hay

en cada clase así como también aquellas que no pudieron ser procesadas. En la esquina inferior izquierda se ve la evolución del tamaño de los grupos a medida que se aumenta la temperatura del algoritmo SPC. En caso de que la clasificación que se obtuvo automáticamente no sea correcta, se puede cambiar utilizando esa información. Por último, se muestra para cada grupo, el intervalo inter-espiga (ISI, por sus siglas en inglés).

En este ejemplo concreto, se puede observar que se identificaron tres clases de espigas, una proveniente de actividad multi-unidad (en color azul) y dos de actividad de neuronas individuales (en color rojo y verde). Quedaron nueve espigas sin clasificar que se muestran bajo el nombre de *cluster 0*.

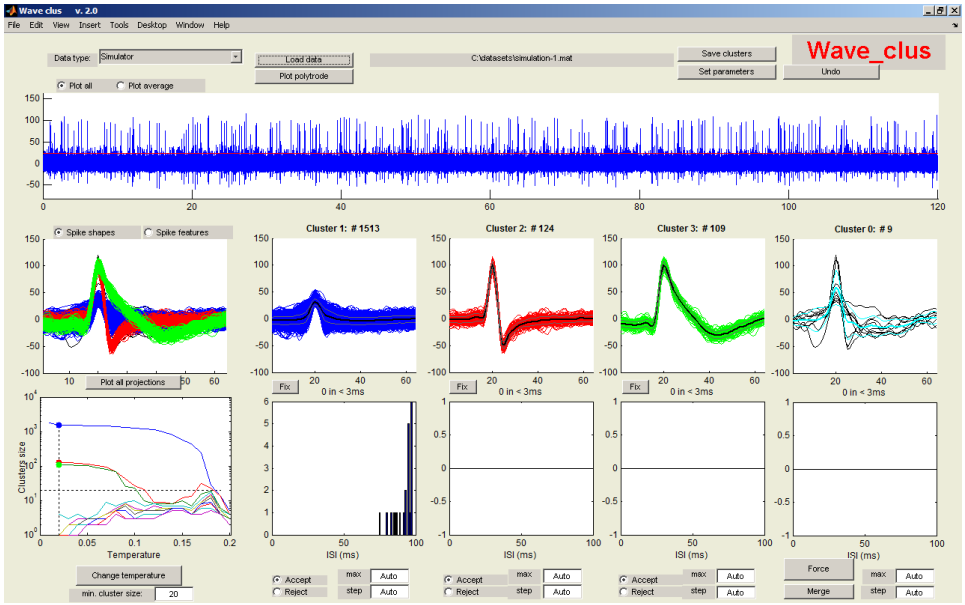


Figura 2.3: Pantalla principal de Wave-Clus.

Capítulo 3

Super-paramagnetic Clustering

El problema del agrupamiento de datos (*data clustering*) se puede enunciar de la siguiente manera: dados N puntos $\{x_i\}_{i=0}^{N}$ en un espacio métrico de dimensión D, determinar la partición de ellos en grupos (*clusters*), tales que los puntos dentro de un grupo son más *similares* entre sí que aquellos en grupos diferentes.

Los algoritmos que resuelven este problema se pueden clasificar en dos grandes categorías: *paramétricos* y *no paramétricos*. En los de la primer categoría, se asume cierta información sobre la distribución de los datos y/o la estructura de los grupos, por ejemplo, que los puntos están aglomerados entorno a un centro. Esta información se incorpora al algoritmo como parte de un *criterio global*. Luego, minimizando este criterio se obtiene la solución. En el caso de los algoritmos *no paramétricos*, esto no ocurre. En estos, se utilizan *criterios locales* para alcanzar la solución. Los grupos se construyen basándose en la estructura local de los datos, por ejemplo, identificando regiones de alta densidad en los mismos.

Super-paramagnetic Clustering es un método de agrupamiento de datos propuesto por Blatt *et al.* [12, 13] en 1996 basado en las propiedades físicas de un sistema ferromagnético no homogéneo y se encuentra dentro de la clasificación de algoritmos *no paramétricos*. Posee varias cualidades que lo hacen muy interesante, a saber: no requiere tener conocimiento previo de

la distribución de los datos o de la estructura de los grupos, la cantidad de grupos es un valor que retorna el algoritmo, tiene un muy buen comportamiento frente a datos con mucho ruido, y por último, retorna información sobre la estructura jerárquica en los datos.

El agrupamiento de los datos se obtiene resolviendo el problema físico del modelo ferromagnético de Potts. A cada uno de los puntos de entrada se le asigna un spin de Potts. Luego, se establecen interacciones entre puntos vecinos cuya fuerza disminuye en función de la distancia que hay entre ellos. Mediante la simulación del modelo, utilizando un método de Monte Carlo, se obtienen diferentes magnitudes físicas que ayudan a establecer las diferentes fases que exhibe el sistema. Finalmente, utilizando la información que otorga una de dichas fases, la *super-paramagnética*, se establecen los diferentes grupos dentro de los datos de entrada.

3.1. El modelo de Potts

El modelo de Potts [17] es una generalización del modelo de Ising. Este último es un modelo matemático de un sistema ferromagnético. Consiste de un conjunto de puntos $\{v_i\}_{i=0}^{N}$, cada uno de los cuales representa un imán atómico, dispuestos en una retícula en $\mathbb{R}^n$. A cada uno de estos puntos v_i se le asocia una variable discreta s_i, llamada *spin*, que pueden tener dos valores posibles: -1, el spin apunta hacia abajo, o $+1$, en cuyo caso apunta hacia arriba. El objetivo del modelo es encontrar las *transiciones de fase* del sistema ferromagnético.

El modelo de Potts extiende a q la cantidad de estados posibles de un spin s. Es decir, s puede tomar alguno de los siguientes valores: $1, 2, 3, \ldots q$. Cada par de spins asociados con los puntos i y j están *unidos* por una interacción cuya fuerza es $J_{ij} > 0$. El estado de cada spin dentro del sistema establece la *configuración* del sistema, denotada por $S = \{s_i\}_{i=0}^{N}$. La *energía* de una configuración está dada por el Hamiltoniano,

$$H(S) = -\sum_{<i,j>} J_{ij}\delta_{s_i,s_j} \tag{3.1}$$

donde $< i, j >$ representa los puntos vecinos i y j.

Con el fin de poder obtener información cualitativa del comportamiento del modelo, se deben calcular ciertas magnitudes termodinámicas a partir del Hamiltoniano. Por ejemplo, para calcular el promedio termodinámico de una magnitud física A a una temperatura T se debe realizar la siguiente suma[1]:

$$\langle A \rangle = \sum_s A(s)P(s) \tag{3.2}$$

donde $P(s)$ es el factor de Boltzmann

$$P(s) = \frac{1}{Z} \exp\left(-\frac{H(s)}{T}\right) \tag{3.3}$$

y donde Z es una constante de normalización,

$$Z = \sum_s \exp\left(-\frac{H(s)}{T}\right) \tag{3.4}$$

El factor de Boltzmann actúa como función de densidad de probabilidad y da el peso estadístico a cada configuración de spins S en equilibrio térmico.

Algunas de las magnitudes físicas más importantes para un sistema magnético son la *magnetización* y el conjunto de funciones δ_{s_i,s_j} ya que a través de sus promedios térmicos se pueden obtener conclusiones sobre el estado del sistema.

La magnetización de una configuración $S = \{s_i\}_{i=0}^N$ está dada por:

$$m(s) = \frac{qN_{max}(s) - N}{(q-1)N} \tag{3.5}$$

donde

$$N_{max}(s) = max(N_1(s), N_2(s), \ldots, N_q(s))$$

[1]Notar que la sumatoria es sobre todo el espacio de configuraciones posibles.

y

$$N_\mu(s) = \sum_{i=1}^{N} \delta_{s_i,\mu}$$

Esta última cuenta la cantidad de spins que tienen el estado μ dentro de la configuración dada.

A partir de la magnetización se puede calcular la *susceptibilidad*,

$$\chi = \frac{N}{T}\left(\langle m^2\rangle - \langle m\rangle^2\right) \tag{3.6}$$

que es utilizada para detectar las *transiciones de fase* que exhibe el sistema.

Por otro lado, mediante las funciones δ_{s_i,s_j} se puede obtener la *correlación spin-spin*, que indica cuál es la probabilidad de que dos spins estén alineados. Concretamente,

$$G_{ij} = \langle \delta_{s_i,s_j} \rangle \tag{3.7}$$

3.1.1. Transiciones de fases

Una *transición de fase* señala un cambio abrupto en las propiedades físicas de un sistema al modificar algunas de sus variables termodinámicas (por ejemplo, el pasaje de estado sólido a líquido de una sustancia al incrementar su temperatura). En este caso las transiciones van a estar determinadas por la variación de la *temperatura* del sistema.

Cuando todos los vecinos más cercanos están a una misma distancia con una misma fuerza de interacción, $J_{ij} = J$, el modelo de Potts se lo denomina *homogéneo*. Este caso presenta 2 fases. Cuando la temperatura es baja, la interacción entre un punto y sus vecinos es alta. Hay una alta probabilidad de que sus spins estén alineados (es decir, se encuentren en un mismo estado) que se refleja en valores altos de la correlación spin-spin. Esto se acentúa al decrementar la temperatura. A temperaturas muy bajas, la interacción entre puntos vecinos es muy alta y sus spins están muy acoplados entre sí, comportándose como si fuesen un gran spin. En este punto

el sistema está totalmente magnetizado, $\langle m \rangle \approx 1$, y se encuentra en su fase *ferromagnética*. A medida que se aumenta la temperatura, el sistema empieza a perder el "orden" que tenía. La magnetización empieza a desvanecerse y el sistema atraviesa un cambio de fase. Cuando la temperatura es muy elevada, la magnetización es nula, $\langle m \rangle = 0$, y el sistema se encuentra en su fase *paramagnética*. En esta fase, el acoplamiento entre los spins es mínimo, lo que resulta en que estos cambian de estado al azar y de manera independiente. Esto tiene como resultado que la probabilidad de que dos spins vecinos estén alineados sea muy baja. En este punto, $G_{ij} \approx 1/q$ para todo par $\{i, j\}$.

Cuando la distancia entre los puntos varía, es decir, hay regiones con mayor densidad de puntos que otras, el modelo se lo considera *no homogéneo*. En este caso, los puntos forman gránulos magnéticos con un alto acoplamiento dentro del mismo gránulo y una interacción débil entre los puntos de otros gránulos. Cada punto tiene una distancia distinta a cada uno de sus vecinos. La fuerza de interacción se debe establecer de modo tal que para pares de puntos que se encuentran a una distancia cercana sea mayor, mientras que para aquellos que están lejos entre sí sea menor. Es decir, el valor J_{ij} se establece como una función decreciente en relación a la distancia de los puntos v_i y v_j. Este modelo presenta la fase *super-paramagnética* (además de las dos ya comentadas anteriormente). La misma se encuentra a una temperatura intermedia, entre las fases *ferromagnética* y *paramagnética*. Lo que la distingue de las otras dos es que en ésta solo los spins con alto acoplamiento están alineados y casi no hay interacción entre los diferentes gránulos. Al seguir elevando la temperatura, las interacciones se vuelven cada vez más débiles, los gránulos se desintegran y los spin empiezan a cambiar de estado de manera independiente, en ese momento es cuando se alcanzar el estado *paramagnético*.

Las transiciones de fase se pueden identificar a través de cambios bruscos en la *susceptibilidad* (que depende de la temperatura). Al pasar de la fase ferromagnética a la super-paramagnética se registra un pico pronunciado en χ. Luego, al seguir incrementando la temperatura, se produce la transición a la fase *paramagnética*. Justo en el momento de la transición se observa una caída abrupta en χ. Las temperaturas a las que ocurre el incremento y el decremento de χ, sirven de cotas inferior y superior, respectivamente,

de la fase super-paramagnética. Cabe destacar que pueden ocurrir más de un pico y una caída en la susceptibilidad a medida que se incrementa la temperatura, es decir, puede haber más de una transición dentro de la fase super-paramagnética. Esto quiere decir que los gránulos se pueden romper en otros gránulos más pequeños y a su vez, estos últimos, se pueden seguir rompiendo en pedazos cada vez más pequeños.

3.2. Simulación del modelo de Potts

Calcular de manera exacta expresiones como 3.2 no es posible ya que la cantidad de configuraciones posibles se incrementa de manera exponencial con el tamaño del sistema (la cantidad total de configuraciones es igual a q^N, donde N es la cantidad de puntos). Es por esto, que se usan métodos numéricos para realizar aproximaciones de las magnitudes físicas mencionadas. Los métodos de Monte Carlo son muy utilizados para realizar a cabo este trabajo. Mediante estos se genera un subconjunto *representativo* del espacio de configuraciones $\{S_1, S_2, \ldots, S_M\}$, que es utilizado posteriormente para realizar los cálculos. De esta manera la expresión 3.2 se reduce a la siguiente sumatoria:

$$\langle A \rangle = \frac{1}{M} \sum_{i=1}^{M} A(S_i) \tag{3.8}$$

donde M es muchísimo menor que q^N para un N grande. La validez de los métodos de Monte Carlo radica en que las configuraciones S_i se generan de acuerdo a la distribución de probabilidad de Boltzmann.

3.2.1. Algoritmo de Swendsen-Wang

Uno de los algoritmos de Monte Carlo para simular el modelo de Potts es el de Swendsen-Wang [20].

El algoritmo genera una nueva configuración a partir de una configuración ya establecida, es decir, genera S_{i+1} a partir de S_i (esto representa un paso de Monte Carlo, $S_n \to S_{n+1}$). La configuración inicial S_1 se pue-

de establecer de manera aleatoria o asignando un valor fijo a cada spin. De este modo, iterativamente se van construyendo todas las configuraciones deseadas.

La generación se realiza de la siguiente manera: supongamos que ya contamos con la configuración S_n y queremos obtener S_{n+1}. El procedimiento consiste en visitar todos aquellos pares de vecinos $< i, j >$ tales que $J_{ij} > 0$. Si sus spins son iguales, es decir, $s_i = s_j$, entonces se crea un *enlace* entre esos vecinos con la siguiente probabilidad:

$$p = 1 - \exp\left(-\frac{J_{ij}}{T}\delta_{s_i,s_j}\right) \tag{3.9}$$

En caso de que los spins sean distintos, *no* se genera el enlace.

Una vez hecho esto para todo par de vecinos, se tiene que hay algunos de ellos que se encuentran *enlazados* y otros que no. Esto define uno o más grupos, llamados *grupos SW*, cada uno conformado por aquellos puntos que tienen un *camino* de enlaces entre sí. Por cada uno de estos grupos se va generar un nuevo estado de manera aleatoria entre todos los estados posibles (*i.e.* $1, 2, \ldots q$). A todos los puntos de un mismo grupo se le asigna el nuevo spin elegido para ese grupo. De esta manera, se obtiene la configuración S_{n+1}.

El pseudo-código del algoritmo se muestra en la figura 1 donde se pueden identificar claramente cada una de sus etapas: *Inicialización*, *Creación de enlaces* y *Asignación de nuevos estados.*

Algorithm 1 SwendsenWang

```
1: InicializarEnCero(B)
2:
3: for all ⟨i, j⟩ tal que J_ij > 0 do                ▷ Crear enlaces
4:     if random ≥ 1 − exp(−J_ij/T δ_{s_i,s_j}) then
5:         B_ij ← 1
6:     end if
7: end for
8:
9: C ← GenerarClustersSwendsenWang(B)
10:
11: for all clusterSW ∈ C do                         ▷ Asignar nuevos estados
12:     s_nuevo ← ElegirEstadoAlAzar(1, Q)
13:
14:     for all v_i ∈ clusterSW do
15:         s_i ← s_nuevo
16:     end for
17: end for
```

3.2.2. Algoritmo de Wolff

Otro de los posibles algoritmos de Monte Carlo para la simulación del modelo de Potts es el de Wolff [21].

Partiendo de una configuración S_i el método, al igual que Swendsen-Wang, genera una nueva configuración S_{i+1}. Esto define un paso de Monte Carlo.

Para el paso de Monte Carlo se elije un punto v_i al azar y se genera un nuevo estado (elegido al azar) entre todos los estados posibles. Se calcula la probabilidad con la cual los vecinos más cercanos de v_i cambien de estado, que es la siguiente (es la misma que el algoritmo de Swendsen-Wang):

$$p = 1 - \exp\left(-\frac{J_{ij}}{T}\delta_{s_i,s_j}\right)$$

Al igual que en el algoritmo de Swendsen-Wang, solo los vecinos que tienen el mismo spin que v_i tienen la posibilidad de cambiar su spin. Aquellos vecinos que sí cambian de estado, forman un conjunto denominado *frontera*. Los mismos no pueden volver a cambiar de estado durante el mismo paso

de Monte Carlo. Para cada vecino de cada punto del conjunto frontera (que no haya sido visitado durante la misma iteración del algoritmo) se calcula la probabilidad de que estos cambien también de estado (utilizando la fórmula anterior). La frontera se actualiza con los nuevos puntos que cambian de estado. Este proceso se repite hasta que no haya más cambios en el conjunto frontera.

Es decir, en este método los cambios de spin se propagan a partir del punto elegido al comienzo del paso de Monte Carlo. En este caso, a diferencia del algoritmo de Swendsen-Wang solo un grupo de puntos cambia de spin. Este proceso de selección y propagación del cambio de estado se realiza un cantidad de veces determinada de modo de alcanzar representatividad estadística.

El pseudo-código del algoritmo se muestra en la figura 2 donde se ven las distintas etapas: *Inicialización*, *Cálculo de la frontera* y *Asignación del nuevo estado*.

Algorithm 2 WOLFF

$v_k \leftarrow SeleccionarVerticeAlAzar(V)$

$s_{viejo} \leftarrow s_k$

$s_{nuevo} \leftarrow ElegirEstadoSpinAlAzar(1, Q)$

$Marcar(v_k, No\ Visitado)$

$F \leftarrow \{v_k\}$

for all $v_i \in F$ / v_i no haya sido $Visitado$ **do** ▷ Calcular frontera
    for all $v_j \in Vecinos(v_i)$ **do**
        if $random \geq 1 - \exp\left(\frac{-J_{ij}}{T}\delta_{s_{viejo},s_j}\right)$ **then**
            $F \leftarrow F \cup \{v_j\}$
        end if
    end for

    $Marcar(v_i, Visitado)$
end for

for all $v_i \in F$ **do** ▷ Asignar nuevo estado
    $s_i \leftarrow s_{nuevo}$
end for

3.3. Usando el modelo de Potts no homogéneo para agrupar datos

Hasta el momento se introdujo el modelo de Potts, se mostraron las distintas transiciones de fase que exhibe al modificar la temperatura y se presentaron 2 posibles algoritmos para realizar la simulación del modelo de modo de poder aproximar las magnitudes físicas de interés. Queda, entonces, mostrar cómo se utiliza el modelo para resolver el problema del agrupamiento de datos.

Sean $\{v_i\}_{i=1}^{N}$ el conjunto de puntos pertenecientes a $\mathbb{R}^n$ que se desea agrupar. El procedimiento general del algoritmo SPC consta de los siguientes pasos:

1. Construir el modelo físico análogo
    a) Asociar una variable spin s_i para cada punto v_i
    b) Identificar los vecinos de cada punto v_i de acuerdo a algún criterio de *vecindad*
    c) Calcular la interacción J_{ij} entre puntos vecinos v_i y v_j
2. Identificar la fase super-paramagnética
    a) Calcular el promedio térmico de la magnetización para diferentes temperaturas
    b) Utilizar la susceptibilidad para identificar la fase super-paramagnética
3. Armar los diferentes grupos
    a) Medir la correlación spin-spin para todo par de vecinos
    b) Construir los grupos

3.3.1. Variables spins

Es necesario determinar la cantidad de estados posibles que puede llegar a tener una variable spin. Blatt *et al.* [13] encontró que q no tiene un impacto significativo en el resultado final del algoritmo. Sin embargo, concluyó que a valores más grandes de q, mayor debe ser la cantidad de pasos de la simulación de Monte Carlo. De esta manera se puede garantizar una buena precisión estadística sobre las magnitudes calculadas. También encontró que el valor de q determinar cuan abrupta son las transiciones de fases y a qué temperatura ocurren las mismas. Blatt *et al.* [13] determinó que $q = 20$ es un valor apropiado para la mayoría de los casos.

3.3.2. Identificación de vecinos

Se debe establecer una definición precisa de vecinos. Una de las motivaciones más importantes para realizar esto es la eficiencia del algoritmo. Si cada punto fuese vecino de todos los demás, existirían un total de N^2 vecinos, donde N es la cantidad de puntos dentro del sistema. Esto puede llegar

a ser un problema, desde el punto de vista de la eficiencia computacional, en caso de valores grandes de N. Es por eso que se toma solo una cantidad limitada de ellos.

El criterio utilizado para determinar si dos puntos, v_i y v_j, son vecinos es el siguiente:

v_i y v_j son vecinos ***si y sólo si****, v_i es uno de los K vecinos más cercanos de v_j y v_j es uno de los K vecinos más cercanos de v_i*

Para evitar que algún punto quede "aislado" del resto, se imponen los ejes del *árbol generador mínimo* del grafo subyacente que conecta a todos los puntos entre sí.

3.3.3. Interacción entre los puntos

La función de interacción entre puntos vecinos debe armarse de modo tal que el modelo tenga las propiedades físicas de un sistema *no homogéneo*. El objetivo es que haya una interacción más fuerte entre vecinos de una región con mayor densidad de datos e interacciones débiles entre vecinos que se encuentran en una región de baja densidad. De esta manera se imita a los gránulos magnéticos que hay en un sistema *no homogéneo*, tal como fue descrito en la sección 3.1.1.

La interacción propuesta entre los puntos vecinos está dada por la siguiente función:

$$J_{ij} = \begin{cases} \frac{1}{\widehat{K}} \exp\left(-\frac{d_{ij}^2}{2a^2}\right) & \text{si } v_i \text{ y } v_j \text{ son vecinos} \\ 0 & \text{si no lo son} \end{cases} \tag{3.10}$$

$\widehat{K}$ representa la cantidad promedio de vecinos y a, la *escala local de longitud*. Esta última es el promedio de todas las distancias d_{ij} entre pares de vecinos v_i y v_j. Cabe notar que el valor de a está definido por las regiones de mayor densidad y es menor que la distancia promedio entre puntos de regiones de baja densidad. También hay que notar que la fuerza de la interacción decrece exponencialmente en función de la distancias entre los puntos vecinos.

3.3.4. Identificación de la fase super-paramagnética

A fin de poder determinar la temperatura a la cuál el sistema se encuentra en su fase super-paramagnética es necesario simular el modelo para diferentes temperaturas, calcular la *magnetización* de modo de obtener la *susceptibilidad* y encontrar los picos y caídas de esta última. La simulación se aplica dentro del rango de temperaturas $T_{inicial} \leq T \leq T_{final}$ donde el incremento está dado por T_{inc}. Además de calcular la susceptibilidad es necesario calcular la correlación spin-spin. Un buen estimador de la función de correlación está dado por el algoritmo de Swendsen-Wang [22] y es el siguiente:

$$c_{ij} = \begin{cases} 1 & \text{si } v_i \text{ y } v_j \text{ pertenecen al mismo cluster } SW \\ 0 & \text{sino} \end{cases} \tag{3.11}$$

Luego, la expresión 3.7 se reduce a:

$$G_{ij} = \frac{(q-1)C_{ij}+1}{q} \tag{3.12}$$

donde $C_{ij} = \langle c_{ij} \rangle$ representa la probabilidad de que los puntos v_i y v_j pertenezcan al mismo grupo SW.

3.3.5. Construcción de los grupos

Los grupos se construyen de la siguiente manera:

1. Conectar los puntos v_i y v_j en caso de que $G_{ij} > \theta$
2. Conectar los puntos v_i con su vecino v_j de máxima correlación G_{ij}
3. Los distintos grupos se forman a partir de aquellos puntos que estén conectados entre sí.

El primer paso genera el *núcleo* de cada grupo. Éste está formado por todos los puntos que tienen una correlación mayor a θ. Para ello, se crea un

enlace entre aquellos puntos que cumplen con la condición establecida. En segundo lugar se incorporan a dichos núcleos aquellos puntos que quedaron en la periferia. Es decir, no estaban lo suficientemente correlacionados con los puntos que forman el núcleo pero dada su *proximidad*, lo más probable es que formen parte de ellos. Finalmente, los grupos se forman a partir de aquellos pares de puntos que tienen un *camino* de *enlaces* entre sí.

3.4. Ejemplo

Una de las ventajas más importantes de este método es que no requiere que los datos de entrada posean ningún tipo de distribución. Esto se puede observar en la figura 3.1.

Los datos de entrada forman 3 círculos concéntricos de diferente cantidad de puntos pero cada uno con una densidad parecida. La cantidad de puntos del círculo exterior es 2400, el del centro es 1600 y el interior es 800. Cómo se puede observar en la figura, el método tiene un alto grado de efectividad. Los tres grupos de datos fueron reconocidos, además, la cantidad de puntos no clasificados es realmente baja, apenas mayor al 2 %.

En [13] se puede ver una comparación minuciosa del algoritmo SPC con otros métodos.

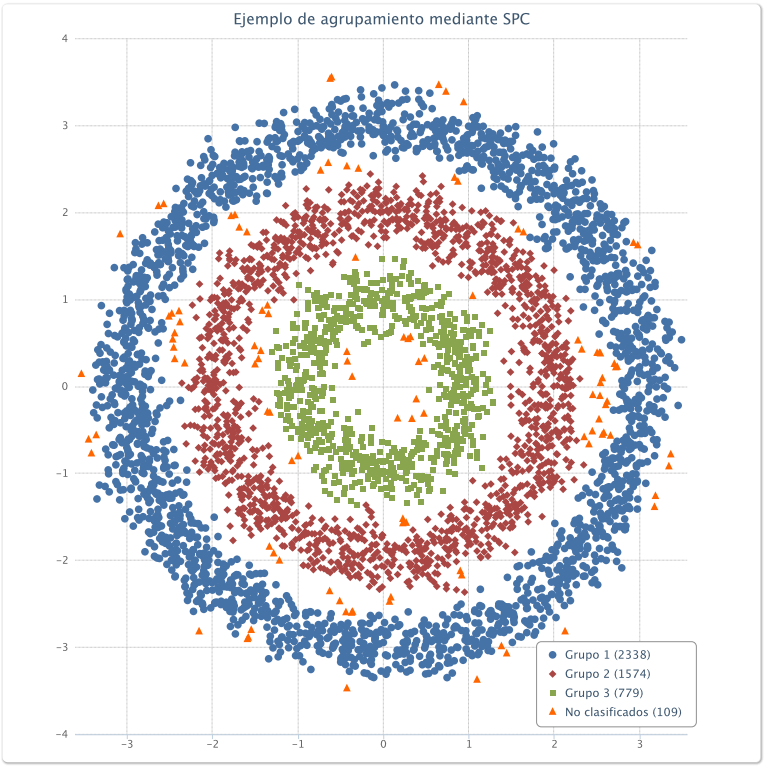


Figura 3.1: Ejemplo de agrupamiento mediante el algoritmo SPC.

Capítulo 4

Desarrollo e implementación

Para llevar a cabo el objetivo propuesto hubo que abordar 2 problemas bien definidos: a) el algoritmo SPC y b) la adaptación del resto de las etapas para así poder brindar un método de ordenamiento de espigas *on-line*.

Como se mencionó en el primer capítulo, el algoritmo SPC es uno de los impedimentos más importantes que tiene el método de Quian Quiroga *et al.* para poder convertirlo en un procesamiento *on-line*. El motivo radica en la incapacidad del algoritmo SPC de poder incorporar nuevos datos a un conjunto ya agrupado.

En este capítulo, primero se plantearán las modificaciones necesarias al algoritmo SPC para que pueda incorporar de manera dinámica nuevos datos. Posteriormente, se explicarán los cambios que deben hacerse al resto de las etapas del método para poder transformarlo, efectivamente, en un método *on-line*.

La ventaja más importante de poder contar con un algoritmo de ordenamiento de espigas *on-line* es la posibilidad de poder procesar la señal registrada simultáneamente con la realización del experimento, brindando la posibilidad de poder cambiarlo o adaptarlo en función de los resultados que se van obteniendo.

4.1. Algoritmo SPC *on-line*

El algoritmo SPC tiene como entrada un conjunto de puntos $\{x_i\}_{i=1}^{N}$ y como salida un conjunto de etiquetas $\{e_i\}_{i=1}^{N}$, donde la etiqueta e_i indica a cuál grupo pertenece el punto x_i.

El objetivo es poder agregar los puntos x_i uno por uno, y en cada oportunidad obtener el etiquetamiento de todos los puntos ingresados al momento. Es decir, luego de ingresar el punto x_i se quiere obtener el etiquetamiento $\{e_j^i\}_{j=1}^{i}$, luego de ingresar x_{i+1} el etiquetamiento $\{e_j^{i+1}\}_{j=1}^{i+1}$ y así sucesivamente [1].

La solución que aquí se propone consiste en combinar la inserción de puntos a agrupar junto con la generación de nuevas configuraciones de spins. Luego de ingresar el punto x_i, se obtiene la configuración S_i. Finalmente, se utilizan todas las configuraciones obtenidas hasta el momento para generar el nuevo etiquetamiento.

Para llevarlo a cabo, al ingresar un nuevo punto x_i, es necesario realizar los siguientes pasos:

1. Agregar x_i al conjunto de puntos y asociarlo con el vértice v_i,
2. Calcular los puntos más cercanos a x_i y, a partir de ellos, calcular los vecinos más cercanos de v_i,
3. Calcular la interacción entre v_i y cada uno de sus vecinos,
4. Calcular la probabilidad de enlace entre vértices vecinos (a partir de la interacción) para el rango de temperaturas establecido,
5. Establecer un valor de spin *apropiado* para v_i,
6. Establecer valores *apropiados* para la correlación entre v_i y cada uno de sus vecinos,
7. Realizar la simulación del sistema para generar una nueva configuración de spins,
8. Construir los grupos y retornar el etiquetamiento $\{e_j^i\}_{j=1}^{i}$

[1] Notar que en cada caso, el etiquetamiento depende del punto ingresado.

Surge la siguiente pregunta: ¿Cómo obtener valores *apropiados* para el spin y para las correlaciones? Como se verá más adelante, el caso de las correlaciones es el más difícil de responder y el que mayor impacto tiene en el resultado final del algoritmo pues la construcción de los grupos se basa fuertemente en esos valores.

La idea que subyace al procedimiento propuesto se basa en lo siguiente: supongamos que el conjunto de puntos es $\{x_1, x_2, \ldots, x_i\}$ y le aplicamos SPC para una cantidad i de iteraciones de Monte Carlo. En la iteración $i-1$ el vértice v_i tiene un valor bien definido para el spin como para las correlaciones entre él y sus vecinos. Si se pudieran aproximar estos valores, el resultado del algoritmo (al aplicar una vez más un paso de simulación) y del procedimiento anterior serían muy similares (para que esto sea así, los vecinos de v_i tienen que ser los mismo en ambas versiones).

Antes de continuar hay que dejar claro que el objetivo final del algoritmo SPC es dar un agrupamiento válido y que la fidelidad con que se simula el sistema físico se encuentra en un lugar secundario (de todas maneras, una mala simulación del mismo, claramente, llevará a un mal agrupamiento de los datos).

4.1.1. Cálculo de vecinos más cercanos

Al ingresar un nuevo punto x_i, primero se lo asocia con su respectivo vértice, v_i. Luego, se obtienen los K puntos más cercanos a x_i. Con esta información se calculan los vecinos de v_i.

El procedimiento es simple: se calcula la distancia de x_i a cada uno de los $i-1$ puntos ya ingresados, quedándose con los K más cercanos. Luego, se aplica el *criterio de vecindad* a cada uno de ellos, obteniendo así los vecinos más cercanos a v_i. Como cada vértice tiene a lo sumo K vecinos, puede darse el caso de que haya vértices que eran vecinos y que luego de insertar x_i ya no lo sean más (es decir, al insertar un nuevo vértice hay una posible actualización de los vecinos de los vértices ya ingresados).

Este método genera la misma matriz de vecinos que lo haría algoritmo SPC original.

4.1.2. Cálculo del spin

No hay una única manera de establecer un valor para el spin del nuevo vértice, v_i, puesto que, en principio, no se sabe exactamente que valor debería tener. Por este motivo, se propusieron varias estrategias y se eligió aquella que brindó mejores resultados. Entre ellas se encuentran:

- Estrategia #1: Establecer un estado al azar o inicializarlo cero.
- Estrategia #2: Establecer el mismo spin que tiene el vecino más cercano de v_i.
- Estrategia #3: Establecer el estado en que se encuentran la mayoría de los vecinos de v_i.

En primera instancia se desarrollaron estrategias de mayor complejidad, que trataban de analizar cuál era el spin predominante en un entorno del nuevo vértice (entre sus vecinos y los vecinos de los vecinos). Sin embargo, mientras más elaborada era la estrategia peor el resultado final. La estrategia que produjo mejores resultados fue la primera.

Cabe hacer en este punto una aclaración sobre el modo de trabajo, especialmente, para la elección de las mejores estrategias (la aquí presentada como también las que se presentarán más adelante). En primer lugar, las estrategias planteadas se evaluaron sobre *toy models*, como el presentado en la sección 3.4 del capítulo anterior y otros mucho más simples. De esta manera, se pudo observar el comportamiento de cada estrategia. Esto sirvió para realizar cambios a las mismas o, directamente, descartarlas. Este fue un proceso iterativo puesto que el algoritmo está compuesto por varias partes y cada una de estas repercute sobre la otra. La primera etapa del proceso terminó al obtener una versión que funcionaba aceptablemente sobre los *toy models*. Posteriormente, se probó la versión sobre un conjunto de datos de espigas para corroborar el funcionamiento y terminar de ajustar detalles. Finalmente, se realizaron las pruebas sobre el juego de datos entero cuyos resultados se verán en el próximo capítulo.

4.1.3. Cálculo de correlación

A diferencia del spin, llegar a buen puerto con el cálculo de la correlación fue una tarea difícil. Inicialmente, se abordó la solución por otro camino. Se intentó completar la correlación entre el nuevo vértice v_i y sus vecinos, a partir de la información que brindaban estos últimos. Por ejemplo, para aproximar la correlación entre v_i y v_j, se calculaba un valor en base a la correlación que había entre v_j y sus vecinos. Esto no condujo a ningún resultado satisfactorio, solo a agrupamientos extremos: se formaba un único grupo integrado por todos los puntos o bien tantos grupos como puntos ingresados. Se continuó por este camino por un tiempo, intentando analizar la matriz de correlación de la versión original del algoritmo de SPC de modo de ver si existía alguna relación entre la correlación de dos vértices cualesquiera y la distancia que había entre ellos, cuál era el valor promedio de las correlaciones entre un vértice sus vecinos, etc. Sin embargo, no se pudo establecer ninguna relación que permitiera aproximar adecuadamente algún valor. Esto se debe a que el modo en que se establece la correlación entre los vértices es muy específico, a través de la simulación, la cual tiene una componente probabilística. Fue necesario buscar otro camino. La solución provino de cómo se utiliza la correlación para formar los grupos y no en cómo establecer valores aproximados.

Teniendo en cuenta lo mencionado, se plantearon las estrategias más factibles para establecer un valor de correlación. Sin embargo, en todas las estrategias se asigna o bien un valor nulo o bien un valor *alto*. Las estrategias propuestas fueron las siguientes:

- Estrategia #1: Establecer la correlación en cero para todos los vecinos.
- Estrategia #2: Para el vecino más cercano, se establece un valor de correlación *alto*. Para el resto, se establece en cero.
- Estrategia #3: Sea v_j el vecino más cercano a v_i. Se calcula la distancia promedio de los vecinos de v_j (sin incluir a v_i) a éste. Si la distancia entre v_i y v_j es menor que dicho promedio, se establece un valor de correlación *alto* entre v_i y v_j. Para el resto de los vecinos de v_i se establece la correlación en cero.

- Estrategia #4: Si la distancia entre el nuevo vértice y uno de sus vecinos es menor que la distancia promedio entre dos vecinos cualesquiera (el valor a que se utiliza para calcular la interacción entre dos vértices), se establece un valor *alto*, caso contrario se establece en cero.

Luego de probar y evaluar cuidadosamente cada una, se eligió aquella que producía mejores resultados, a saber, *Estrategia* #4.

La distancia promedio entre dos vértices *vecinos* cualesquiera (el valor a) es menor que la distancia promedio entre dos puntos. Es utilizada en el cálculo de la interacción y tiene un impacto directo en cómo se forman los grupos. Cuando hay zonas con mayor densidad de puntos, se ve reflejado en este valor. Esto genera que puntos vecinos de zonas de baja densidad tengan una interacción más baja. Por lo tanto, estableciendo un valor de correlación alto para dos vecinos que disten menor que dicho valor aumenta la probabilidad de terminen en el mismo grupo.

El valor *alto* que se menciona, se estableció como un parámetro configurable del algoritmo. Ayuda a que, en un principio, si dos vértices son vecinos muy cercanos terminen en el mismo grupo. Luego, el algoritmo de simulación incrementará adecuamente este valor, haciendo que se mantenga el vínculo entre los dos vértices o bien se pierda. Sin embargo, elimina las fluctuaciones del vértice recién agregado entre alguno de los grupos existentes y el conjunto de vértices no agrupados (formado por aquellos puntos que forman un solo grupo o grupos muy pequeños). En el capítulo de resultados se mostrarán ejemplos concretos de soluciones que son inestables. También se estableció como parámetro, un factor de escalamiento para la distancia. Esto es, al valor a se lo multiplica por un escalar, que en caso de ser menor que 1 impone una mayor restricción para establecer la correlación y en caso de ser mayor, flexibiliza la condición. Si los puntos de entrada son muy similares entre sí, un factor pequeño mejora la separación e identificación de los distintos grupos.

4.1.4. Simulación del modelo de Potts

Para la simulación se pueden emplear los dos algoritmos descritos en el capítulo anterior: Swendsen-Wang y Wolff. Ambos llegan a resultados

muy similares pudiéndose emplear uno u otro indistintamente. La mayor diferencia entre ellos es que Swendsen-Wang genera una nueva configuración cambiando el estado de los spins de todos los vértices en cada pasada. En cambio, Wolff realiza el cambio para un subconjunto, asignando un mismo estado. Específicamente, elige un vértice al azar, cambia su estado y, luego, propaga ese cambio a través de sus vecinos, los vecinos de sus vecinos, etc., siempre y cuando se cumplan ciertas condiciones.

Ambos fueron evaluados de modo de ver cuál se comportaba mejor con la inserción dinámica de puntos. En la sección de 4.1.7 se discutirán los pros y contras de cada uno.

4.1.5. Construcción de grupos

Esta etapa es una de las que mayor incidencia tiene en el éxito del objetivo que se desea alcanzar. Consiste en la construcción de los agrupamientos y tiene una relación estrecha con la correlación. Como se mencionó en el capítulo anterior, para construir los grupos se utiliza un umbral sobre la correlación entre vértices vecinos. En primera instancia se fija el valor del umbral, posteriormente, se recorren todo par de vecinos y si su correlación supera el umbral se establece un *vínculo* o *enlace* entre ellos. Finalmente, todos los vértices que están *enlazados* forman parte de un mismo grupo.

En este punto cabe hacer una aclaración respecto al máximo valor que puede alcanzar la correlación entre dos vértices. En el algoritmo SPC, antes de comenzar con la simulación, la correlación para todo par de vértices vecinos es cero. Al aplicar un paso de la simulación la correlación aumenta, a lo sumo, en 1. Por lo tanto, si se ejecutan M pasos de simulación el mayor valor posible de correlación entre dos vértices vecinos es M.

Como se mencionó anteriormente, en un principio, se abordó la solución de manera diferente. Se intentó establecer valores aproximados de correlación entre el nuevo vértice y sus vecinos. Esto no dio buenos resultados. Posteriormente, se cambió de enfoque y se estableció la correlación inicial en cero. De esta manera, la correlación aumenta solo a través de la simulación. Sin embargo, había un problema a resolver. La correlación máxima que puede alcanzar un vértice que es agregado al comienzo es muy distinta de

la de uno que es agregado más tarde. Por ejemplo, si v_1 tiene como vecinos a v_{10} y v_{100}, la correlación máxima con el primero es 90 mientras que con el segundo es 1. Esto se debe a que después de insertar un nuevo vértice se ejecuta un paso de simulación. La aplicación del umbral tiene que tener en cuenta este hecho y, por lo tanto, queda determinada de la siguiente manera (para un par de vecinos, v_i y v_j):

1. Obtener la máxima correlación:
   $maxCorr =$ cantidad total de puntos ingresados $- max(i, j)$
2. Aplicar umbral:
   Establecer enlace si: $corr(v_i, v_j) >= umbral \times maxCorr$.

4.1.6. Criterio de selección de la temperatura

En el algoritmo SPC, se simula el sistema para diferentes temperaturas, se determina en cuál aparecen los grupos y, finalmente, se obtiene el etiquetamiento de los puntos. Al transformar el método a modo *on-line* hay que determinar una temperatura luego de insertar cada punto.

La elección de la temperatura se realiza de manera diferente a la planteada por Blatt *et al.* [13] (que utiliza los picos en la susceptibilidad). El criterio propuesto se basa en el tamaño de los agrupamientos. Precisamente, se elige aquella que exhibe la mayor cantidad de grupos con mayor cantidad de puntos cada uno. Es decir, se busca maximizar la cantidad de grupos y la cantidad de puntos por grupo. Como se verá en el próximo capítulo, este criterio da muy buenos resultados.

Esto fue necesario por dos razones. La primera, está relacionada al algoritmo SPC. En este, al aumentar las temperatura los grupos emergen respetando la distribución jerárquica de los mismos. Por ejemplo, supongamos que se cuenta con un conjunto de puntos, en el cual se pueden distinguir dos grupos bien definidos. Al aplicar SPC, estos dos grupos van a ser reconocidos como tal a una temperatura determinada. Supongamos además, que uno de esos grupos tiene una zona más densa que otra, es decir, dentro del mismo grupo se pueden identificar dos partes que lo componen. Al incrementar la temperatura, el algoritmo SPC va a reconocer a esas dos zonas como grupos. Un conjunto de entrada de este tipo genera más de un pico

en la susceptibilidad, por lo tanto, se dificulta establecer un buen criterio de elección de temperatura basado en ella. La otra razón, es que el cálculo de la susceptibilidad en el nuevo algoritmo se ve afectado, en menor o mayor medida, por los cambios introducidos, dificultando más aún establecer un criterio que involucre a esta magnitud física.

4.1.7. Aspectos de performance

Como se verá en el próximo capítulo, el algoritmo propuesto tiene un muy buen desempeño en cuanto a la identificación de los grupos. Sin embargo, en pos de que el método permita procesar una gran cantidad de espigas en un tiempo adecuado, es necesario realizar algunos cambios.

En primera instancia, no es necesario obtener el agrupamiento después de insertar cada nueva espiga. Se modificó el algoritmo para que solo se retorne el agrupamiento luego de procesar una cantidad determinada (esto quedará más claro al presentar el método de ordenamiento de espigas *online*). Esto impacta directamente en el tiempo insumido por el algoritmo y no altera la calidad de los agrupamientos.

Otro proceso que consume mucho tiempo es la actualización de la matriz de interacción y el cálculo de las matrices de probabilidad de enlace (una por cada temperatura, utilizada en la simulación). Al insertar un nuevo vértice, se calculan los vecinos del mismo. Esto provoca cambios en la matriz de vecinos (algunos vértices dejan de ser vecinos luego de insertar un nuevo punto) y, particularmente, en la cantidad promedio de vértices vecinos y distancia promedio entre vecinos (parámetros $\hat{K}$ y a, de la fórmula de interacción). Estos cambios se deben reflejar en la matriz de interacciones (hay que recalcularla por completo) y, consecuentemente, en las de probabilidad de enlace. Este es un proceso muy costo, no solo por la cantidad de valores a calcular sino porque todos estos involucran a potencias de e. Este proceso no es necesario hacerlo después de cada inserción. Es recomendable la actualización para los primeros puntos ingresados de modo de obtener buenas aproximaciones de $\hat{K}$ y a. Luego de los primeros puntos, esta actualización se puede hacer cada tanto, sin mayores perjuicios a la calidad del resultado final.

La simulación es otro de los procedimientos que mayor cantidad de tiempo demora. Ambos algoritmos presentados, tienen un orden de complejidad de $\mathcal{O}(NK)$, donde N es la cantidad de vértices y K es la cantidad de vecinos. Indefectiblemente, a medida que aumenta la cantidad de vértices también lo hará el tiempo que toma en completarse un paso del algoritmo de simulación. Se analizaron varias alternativas para minimizar este tiempo, entre las cuales están:

- Alternativa #1: Aplicar la simulación cada X cantidad de vértices agregados.
- Alternativa #2: Aplicar la simulación solo a los últimos puntos agregados.

En el primer caso, se experimento con los dos algoritmos, distanciando cada vez más un paso de simulación de otro. Swendsen-Wang probó ser el más robusto en este aspecto, logrando un distanciamiento máximo de 25 puntos antes de notar una degradación en la calidad del resultado final. En cambio, Wolff solo admite solo unos 5 puntos como máximo. El motivo de esto tiene que ver a que en SW se cambia el estado de los spin de todos los vértices en un mismo paso (generando una configuración bastante distinta de la anterior) mientras que Wolff solo lo hace para un subconjunto.

También se experimentó aplicando más de una vez el algoritmo de simulación cada X cantidad de puntos tratando de que X sea bastante más grande que 25. Esto produjo buenos resultados al principio pero desmejoraba a medida que se iban sumando cada vez más puntos.

Para el segundo caso, se buscó clasificar lo mejor posible los primeros puntos y, luego, aplicar la simulación solo a los últimos puntos agregados. Si bien esto se comporta satisfactoriamente por una cantidad de puntos, no se mantiene en el tiempo. A medida que se van agregada cada vez más puntos, estos últimos quedan sin agruparse.

Una observación que se realizó es que el algoritmo de Wolff da mejores resultados en caso de necesitar obtener el agrupamiento punto por punto.

Finalmente, se optó por la alternativa #1 ya que brinda el mejor balance entre calidad de solución y performance.

4.1.8. Problemas que surgen al agregar puntos dinámicamente

La constante actualización de la matriz de vecinos genera algunos inconvenientes. En casos donde los puntos están concentrados en una misma región, hay una actualización constante de los vecinos de cada vértice (es decir, vértices que eran vecinos dejan de serlo y se agregan nuevos). Esto hace que se pierda la correlación entre los mismos (puesto que dejan de ser vecinos). El cambio tan frecuente lleva a que los grupos sean muy inestables. Puntos que pertenecían a un mismo grupo dejan de formar parte de él de un momento a otro. Una de las posibles soluciones a este problema es elevar el número máximo de vecinos por vértice, sin embargo, esto impacta directamente en el tiempo de procesamiento del algoritmo.

4.2. Ordenamiento de espigas *on-line*

En la figura 3 se puede ver el pseudo-código del algoritmo de ordenamiento de espigas en modo *on-line*. Se lee el último segundo de señal registrada y se aplican las 4 etapas del procesamiento. Para que el algoritmo sea en tiempo real, cada iteración del ciclo que se muestra debería tardar a lo sumo 1 segundo en completarse, tiempo que hay que esperar hasta obtener el próximo segundo de señal.

Algorithm 3 Ordenamiento de Espigas On-Line

```
1: while señal ← LeerUltimoSegundoDeSeñal do
2:     señalFiltrada ← Filtrar(señal)
3:
4:     espigas ← Detectar(señalFiltrada)
5:
6:     características ← Extraer(espigas)
7:
8:     clasesDeEspigas ← Clasificar(características)
9: end while
```

En la versión que aquí se presenta las 4 etapas se basaron en el método de Quian Quiroga *et al.* [7]. Ya se mostró la transformación del algoritmo SPC a modo *on-line* que conforma la etapa de *clasificación*, resta ver las

adaptaciones de las demás.

La primer etapa, *Filtrado*, se mantiene igual dado que el único cambio es la longitud de la señal filtrada.

La etapa de *Detección*, también se mantiene igual, sin embargo, hay que notar que ahora el umbral de detección se realiza sobre una porción muy chica se la señal (antes se realizaba sobre toda la señal). Esto, en principio, no presenta ningún problema. Más aún, se cree que va a mejorar la detección ya que el umbral se torna *local*. De este modo, si el ruido en la señal varía en el tiempo el umbral se adapta a estas variaciones.

La etapa de *Extracción de Características* debe ser modificada. El motivo es que en el método de Quian Quiroga *et al.* [7] se aplica el test de normalidad sobre los coeficientes wavelet de *todas* las espigas para obtener los 10 coeficientes que representan a cada espiga. En este caso, al no contar con todas las espigas, ya que se van detectando a medida que se va leyendo la señal, no se puede realizar la extracción de características tal cual fue concebida. Las posibles alternativas son las siguientes:

- Utilizar los 64 coeficientes que representan a la espiga,
- No aplicar el test de normalidad y utilizar los 64 coeficientes de la wavelet,
- Aplicar el test *solo* a los coeficientes de las últimas espigas,
- Aplicar el test a los coeficientes de *todas* las espigas detectadas hasta el momento.

Cada una de estas opciones tiene sus desventajas. En el primer caso, si la señal de entrada tiene mucho ruido, se puede ver afectada la clasificación. Recordar que con el test de normalidad se obtienen los 10 coeficientes que mejor separan una espiga de la otra.

En el segundo caso, es posible que los coeficientes que se eligen para una espiga varíen a través del tiempo. Por ejemplo, si en el primer y décimo segundo hay una espiga de una misma clase, es probable que se hayan elegido coeficientes diferentes en ambos casos. Esto repercute al clasificarlos puesto que los vectores que representan a una misma clase de espiga son diferentes

entre sí. Algo parecido pasa con la tercera opción, aunque este efecto se va atenuando a medida que pasa el tiempo puesto que involucra la información de todas las espigas detectadas al momento.

4.3. Implementación

La implementación del algoritmo SPC *on-line* se realizó en el lenguaje C++ por cuestiones de performance. Se tuvieron todos los cuidados para brindar la implementación más eficiente posible. Además se implementó una interfaz MEX [2] para que pueda ser utilizados desde Matlab y así poder aprovechar los beneficios de este último.

En la figura 4.1 se puede ver cómo se utiliza el programa desde Matlab. Primero, se establecen los parámetros de entrada. Luego, se van agregando una por una las espigas. En cada ocasión se obtienen tres resultados:

- labeling: es una matriz de $t \times n$, donde t es la cantidad de temperaturas para las cuales se realiza la simulación del modelo de Potts y n es la cantidad de espigas insertadas al momento.
- numberOfClusters: es una matriz de $t \times 1$. Retorna la cantidad de grupos que se formaron para cada temperatura.
- clustersSize: es una matriz de $t \times nc$. Reporta el tamaño de los primeros nc grupos más grandes que se formaron para cada temperatura.

La implementación permite agregar más de una espiga por vez, en caso de que no sean necesario conocer a cada instante el agrupamiento de las espigas. Esto disminuye el tiempo de procesamiento, pues la construcción de los grupos se realiza menos veces. En la figura 4.2 se muestra su uso para *paquetes* de 30 espigas cada uno.

El método de ordenamiento de espigas *on-line* se implementó como una rutina de Matlab. Se utilizaron algunas de las rutinas que componen *Wave-Clus*, en ciertos casos, sin modificaciones de por medio, por ejemplo, la rutina

[2] Para mayor información, visitar: `http://www.mathworks.com/support/tech-notes/1600/1605.html`

```
% Configuracion de parametros
spcoParams = {
  'DataDimension',            10, ...
  'NumberOfNeighbours',       20, ...
  'TempBegin',                0.0, ...
  'TempEnd',                  0.1, ...
  'TempInc',                  0.01, ...
  'PottsSpins',               20, ...
  'ThresholdScale',           0.5, ...
  ...
}

mxspco(spcoParams)

% Procesamiento de espigas (de a una a la vez)
for s = 1 : numberOfSpikes
    spike = spikesAll(s, :)

    [labeling, numberOfClusters, clustersSize] = ...
        mxspco(spike)
end

% Liberacion de recursos.
mxspco([]);
```

Figura 4.1: Ejemplo de uso de SPC *on-line* desde Matlab.

```
...

% Procesamiento de espigas (de a paquetes de 30)
for s = 1 : 30 : numberOfSpikes
    spikes = spikesAll(s:s+30-1, :)

    [labeling, numberOfClusters, clustersSize] = ...
        mxspco(spikes)
end

...
```

Figura 4.2: Ejemplo de uso de SPC *on-line* desde Matlab.

para filtrar la señal. En otros casos, hubo que hacer modificaciones menores, por ejemplo, la rutina de extracción de características.

Capítulo 5

Resultados

En este capítulo se presentan los resultados obtenidos. Se dividen en tres partes: a) comparación entre SPC y SPC *on-line*, b) evaluación de la calidad del resultado y tiempo de procesamiento de SPC *on-line* y c) evaluación del desempeño general del método *on-line*. La evaluación del algoritmo SPC *on-line* se realiza solamente sobre datos de espigas ya que las modificaciones fueron motivadas por el desarrollo del método de ordenamiento de espigas *on-line*.

5.1. Evaluación de desempeño

Para evaluar el desempeño del algoritmo se emplearon registros sintéticos de actividad neuronal [23] generados a través de simulaciones que otorgan un alto grado de realismo y sus características se asemejan estrechamente a datos de registros reales [14].

El mayor beneficio que brindan es saber exactamente cuántas espigas hay en la señal, donde están ubicadas en el tiempo y, más importante aún, a cuál clase pertenece cada una. De este modo, se puede medir con precisión la calidad de los grupos retornados por el algoritmo.

Se utilizó un juego de datos de 5 simulaciones, todas de 2 min de duración. En la figura 5.2 se pueden ver las espigas que conforman cada una de

ellas [1]. Todas están compuestas por tres grupos de espigas uno de los cuales representa actividad multi-unidad (en color rojo), los demás representan la actividad de dos neuronas (en color verde y azul). En el cuadro 5.1 se detalla la composición de cada simulación.

Como puede verse, las simulaciones #1 y #2 son similares entre sí, las espigas provenientes de neuronas individuales tienen una mayor amplitud que la actividad multi-unidad. La mayor diferencia está en el la frecuencia de activación de las neuronas, siendo mucho más elevado en la segunda simulación.

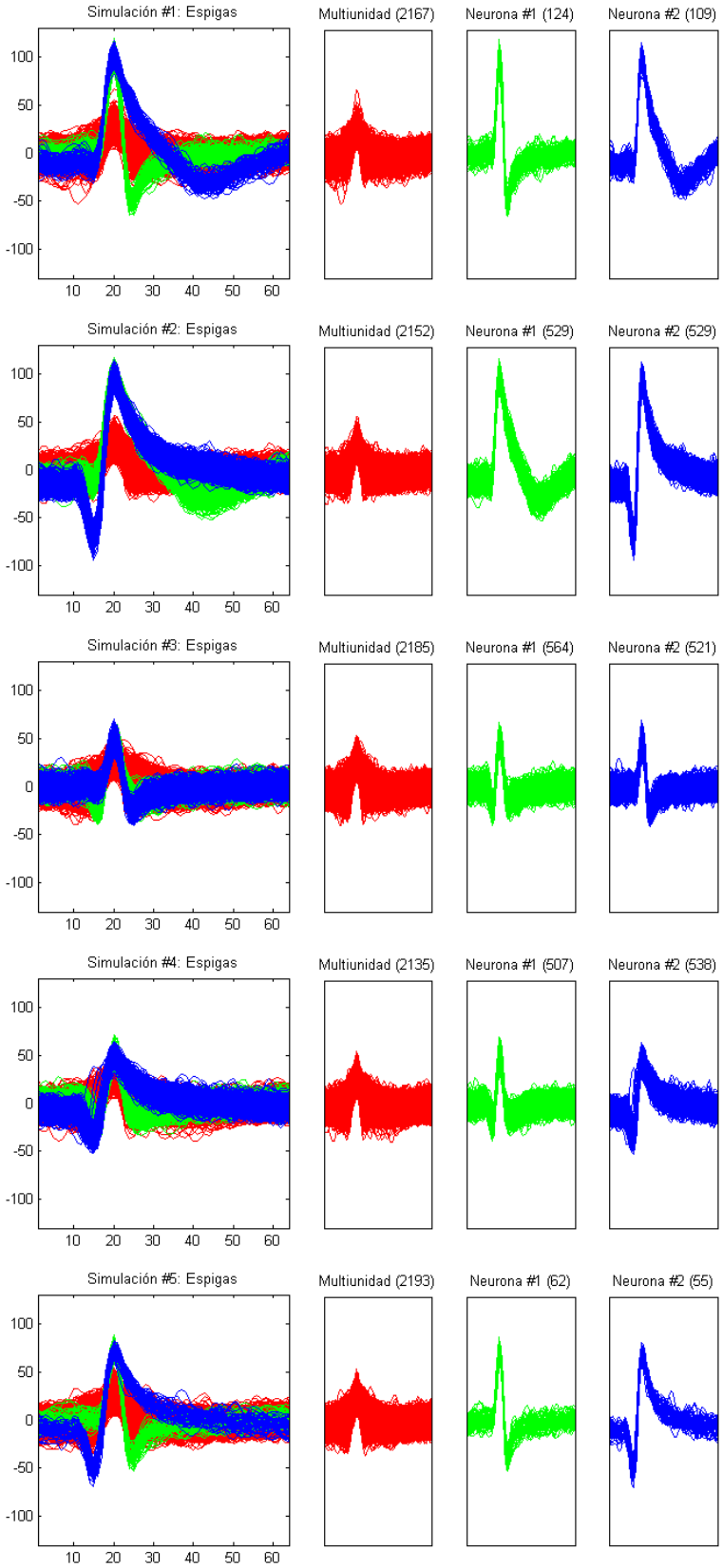
Las simulaciones #3 y #4 también son parecidas entre sí. Ambas presentan neuronas con una alta frecuencia de activación. En este caso, la amplitud es casi idéntica para todas las espigas. Esta característica impone un reto al momento de agruparlas correctamente.

La simulación #5 está en una posición intermedia respecto de la amplitud de las espigas. No es tan alta como en los 2 primeros casos (para las neuronas individuales) pero guarda cierta diferencia con la actividad multi-unidad. La frecuencia de activación es bastante reducida, otra característica que podría dificultar la correcta separación.

	Clase de espiga			
	Multi-unidad	Neurona #1	Neurona #2	Total
Simulación #1	2167	124	109	2400
Simulación #2	2152	529	529	3210
Simulación #3	2185	564	521	3270
Simulación #4	2135	507	538	3180
Simulación #5	2193	62	55	2310

Cuadro 5.1: Detalle de los grupos de espigas de cada simulación.

[1] La cantidad de espigas de cada señal se redondeó a un múltiplo de 30 por cuestiones de simplicidad. Más adelante, quedará claro el motivo.



Cuadro 5.2: Espigas de cada simulación.

5.2. Comparación entre SPC y SPC *on-line*

Para cada una de las simulaciones, se separaron las espigas (reales) de la señal y se aplicó la extracción de características de modo de obtener los 10 coeficientes que representan a cada una de ellas. Luego, sobre ese conjunto de datos (uno por cada simulación) se ejecutó SPC y SPC *on-line*. Se midió la cantidad de espigas clasificadas: a) correctamente, b) incorrectamente y c) no clasificadas. En todos los casos se compara el agrupamiento final (el que involucra a todas las espigas) ya que la versión SPC original brinda *solo* esa información.

Los parámetros del algoritmo SPC fueron los que usa *Wave-Clus* por defecto y son los siguientes:

- *Dimensión de los puntos de entrada*: 10
- *Cantidad de vecinos*: 11
- *Umbral de correlación (θ)*: 0,5
- *Cantidad de iteraciones de MC*: 100
- *Temperatura mínima*: 0,0
- *Temperatura máxima*: 0,2
- *Incremento de temperatura*: 0,01

Para el caso de SPC *on-line* los parámetros fueron los siguientes:

- *Dimensión de los puntos de entrada*: 10
- *Cantidad de vecinos*: 20
- *Umbral de correlación (θ)*: 0,5
- *Ejecución de MC cada X puntos*: 25
- *Correlación inicial*: 10
- *Temperatura mínima*: 0,0

- *Temperatura máxima*: 0,1
- *Incremento de temperatura*: 0,01

Algunos de los parámetros difieren por los siguientes motivos:

- *Cantidad de vecinos*: Se incrementa la cantidad de vecinos para mitigar los efecto de la constante actualización de la matriz de vecinos (comentado en el capítulo anterior).
- *Temperatura máxima*: En el algoritmo SPC *on-line*, los grupos emergen a una menor temperatura y para temperaturas más elevadas que la fijada solo se forman grupos muy pequeños.

Como se pueden ver en la tabla 5.3, los resultados de SPC *on-line* para las simulaciones #1, #2 y #5 son muy buenos y se asemejan mucho a los producidos por la versión original; los porcentajes de espigas clasificadas correctamente casi llegan al 100 %. En cambio, para las simulaciones #3 y #4, el desempeño decae, el porcentaje de clasificadas correctamente está por debajo de la versión original, $\sim 6\,\%$ y $\sim 10\,\%$, respectivamente. Sin embargo, casi no hubo clasificaciones incorrectas, solo se incrementó la cantidad de espigas no clasificadas. Esto se debe a que las espigas son muy similares entre sí, tanto en forma como en amplitud, dificultando mucho la separación. La versión original de SPC también presenta problemas, aunque en menor medida. El motivo de que la versión *on-line* no se acerque a los porcentajes de la original es la alta tasa de actualización de la matriz de vértices vecinos, provocando que se pierdan las correlaciones que se habían conseguido con vértices vecinos (desplazados por otros por una diferencia muy chica en la distancia) con mucha frecuencia. La consecuencia directa, es un gran porcentaje de espigas no clasificadas. Como se verá en la próxima sección, este inconveniente se presenta a cada momento durante el agrupamiento (aunque varían los porcentaje) generando un resultado inestable a través del tiempo.

Un parámetro más que se estableció fue el factor de escalamiento de la distancia (ver sección 4.1.3). En el caso de las simulaciones #1, #2 y #5 fue 1,4 mientras que para #3 y #4 fue 0,65. Un valor conservador para este

parámetro es 0,9 (genera buenos agrupamientos para la mayoría de los casos). Para conjuntos donde los datos son muy parecidos entre sí (por ejemplo, las simulaciones #3 y #4) conviene un valor más pequeño de modo de hacer más estricta la condición para establecer la correlación inicial y así mejorar la separación de grupos muy similares. En conjunto de datos donde hay una mayor diferencia entre sus grupos (como las simulaciones #1, #2 y #5) se puede aumentar de modo de disminuir la cantidad de puntos no agrupados.

	SPC (en %)			SPC *on-line* (en %)		
	Clasif. Correct.	Clasif. Incorrect.	No Clasif.	Clasif. Correct.	Clasif. Incorrect.	No Clasif.
Simulación #1	99.21	0.00	0.79	99.46	0.00	0.54
Simulación #2	99.75	0.00	0.25	100.00	0.00	0.00
Simulación #3	79.85	0.70	19.45	70.76	0.40	28.13
Simulación #4	80.94	0.47	18.58	74.81	0.22	24.97
Simulación #5	99.57	0.00	0.43	99.65	0.00	0.35

Cuadro 5.3: Comparación ente SPC y SPC *on-line*. Porcentajes de clasificación para cada simulación.

5.3. Desempeño de SPC *on-line*

5.3.1. Calidad de agrupamiento

En este caso se midió los porcentajes de clasificación (correcto, incorrecto y no clasificado) a medida que se iban insertando espigas a ser clasificadas de modo de evaluar el comportamiento del algoritmo a través del tiempo.

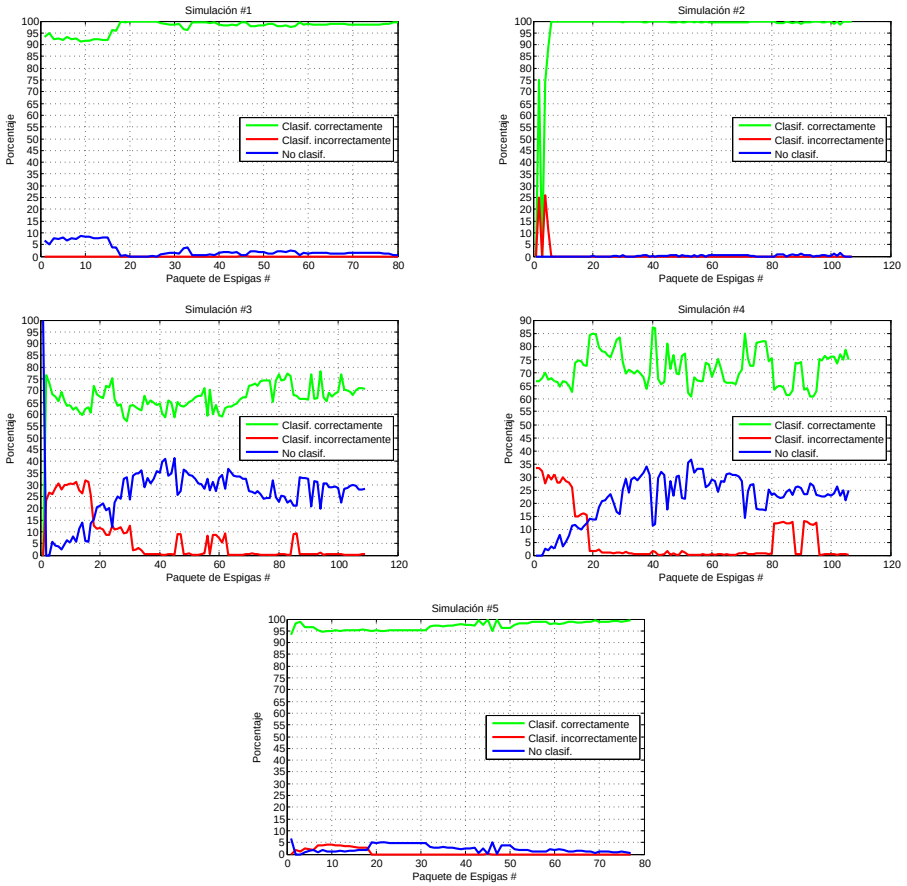
Una observación para hacer es que las espigas se fueron agregando de a *paquetes* de 30. La razón está en cómo es utilizado el algoritmo dentro del método de ordenamiento de espigas *on-line*. En este, se toma 1 s de señal, se detectan las espigas y se las procesan. En las simulaciones, la cantidad máxima de espigas por segundo fueron aproximadamente 30. Vale aclarar que esto no altera en absoluto el resultado final.

Nuevamente, los parámetros utilizados para configurar el algoritmo fueron los mismos que la sección anterior.

En la tabla 5.4 se pueden ver los gráficos de clasificación para cada simulación. Para las simulaciones #1, #2 y #5, se mantienen notablemente estable a medida que se van agregando paquetes de espigas. Es decir, hay una correcta identificación de los distintos grupos casi desde el inicio y, a medida que se van agregan espigas, son clasificadas correctamente. Se puede observar que la cantidad de clasificaciones correctas supera el 95 % mientras que el error se mantiene cercano a 0 % y los no clasificados no superan el $5 - 6\,\%$.

Tal como ocurrió en la sección anterior, el caso de las simulaciones #3 y #4 presenta un mal comportamiento. Se puede ver con mayor claridad el efecto de la alta frecuencia de actualización de la matriz de vecinos. Si bien el error no es muy grande y se mantiene bajo, a excepción de picos que se observan cada tanto, la variabilidad de los porcentajes de clasificación correcta y no clasificados es muy grande. Esto indica que se identifican grupos (en su gran mayoría se corresponden con los correctos) que de un momento a otro pasan a estar no clasificados. Notar la alta relación entre los picos de los porcentajes de clasificación correcta y los valles de los no clasificados y viceversa.

Se podría pensar que aumentando aún más la cantidad de vecinos se



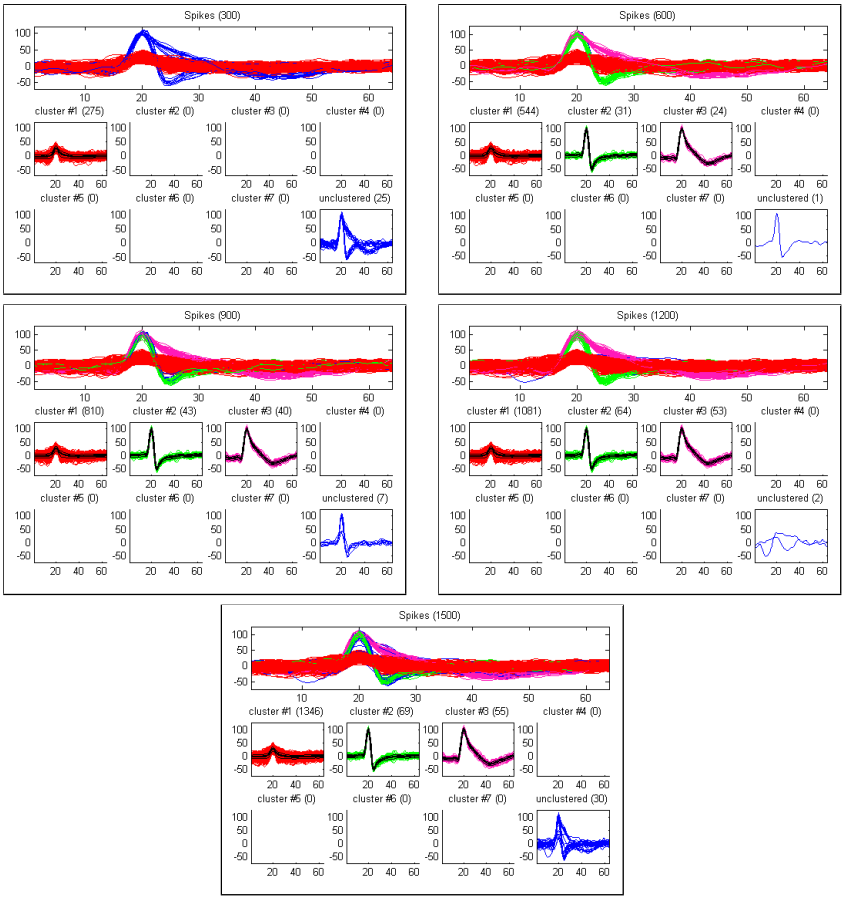
Cuadro 5.4: Desempeño de SPC *on-line*. Porcentajes de clasificación para cada simulación.

podría atenuar este efecto, sin embargo, esto no es así. No solo en la versión *on-line* sino también en la original. La cantidad de vecinos, por lo general, es un parámetro que no necesita ajustarse, sin embargo, depende en cierta medida del tipo de puntos que se está agrupando (dimensionalidad, dispersión en el espacio, etc.). A diferencia de lo que se podría llegar a esperar, aumentar mucho la cantidad de vecinos puede llegar a ser contraproducente. Que un vértice posea muchos vecinos, implica que tiene interacción con un entorno grande alrededor de él. Esto genera distorsiones en la formación de los grupos, que tienden a ser más grandes, generalmente combinando en uno solo dos grupos cercanos (en distancia) o parecidos.

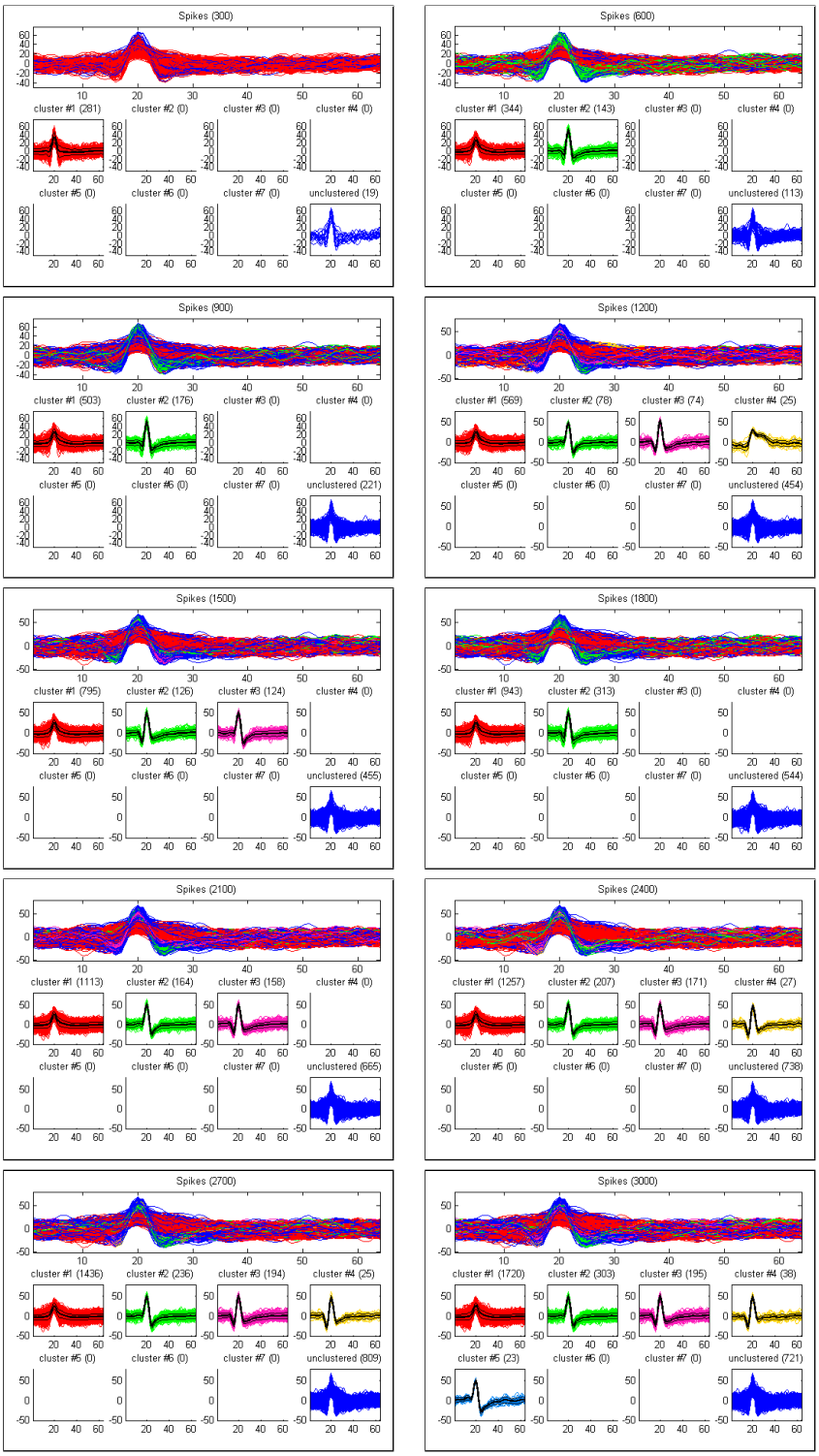
En los cuadros 5.5 y 5.6 se pueden ver los agrupamientos formados a intervalos de 300 espigas para las simulaciones #1 y #3, respectivamente.

En el primer caso, se puede ver que en un principio se reconoce solamente a las espigas provenientes de la actividad multi-unidad. Posteriormente, se identifican las provenientes de las neuronas registradas. Como mostraba el gráfico de clasificación, los grupos se mantienen estables a medida que se van agregando espigas.

La simulación #3 presenta dos inconvenientes: la dificultad de separar bien cada clase y la inestabilidad de los grupos formados. Esto se ve con claridad en las diferentes capturas del cuadro 5.6. Además, puede verse por momentos que algunos grupos se parten en dos, también producto de los constantes cambios en la matriz de vecinos.



Cuadro 5.5: Capturas de pantalla de los ordenamientos que se obtuvieron para la simulación #1 (cada 300 espigas).



Cuadro 5.6: Capturas de pantalla de los ordenamientos que se obtuvieron para la simulación #3 (cada 300 espigas).

5.3.2. Tiempo de procesamiento

En la figura 5.1 se puede ver como aumenta el tiempo de procesamiento [2] a medida que se agregan paquetes de espigas. El crecimiento es lineal en la cantidad total de espigas. Los picos pronunciados a intervalos regulares corresponden a la ejecución del algoritmo de simulación (específicamente, el algoritmo se ejecuta dos veces en esos puntos debido a que hay 30 espigas en cada paquete y la simulación se aplica cada 25 de ellas).

Es evidente el impacto que tiene la ejecución del algoritmo de Monte Carlo y como se profundiza a medida que se agregan más y más espigas. Como es de esperarse, esto será una de las limitaciones a la hora de procesar una gran cantidad de datos.

Cabe destacar que para todas las simulaciones el ordenamiento se realizó en tiempo real. Los tiempos totales de ejecución estuvieron entre 9 y 18 segundos.

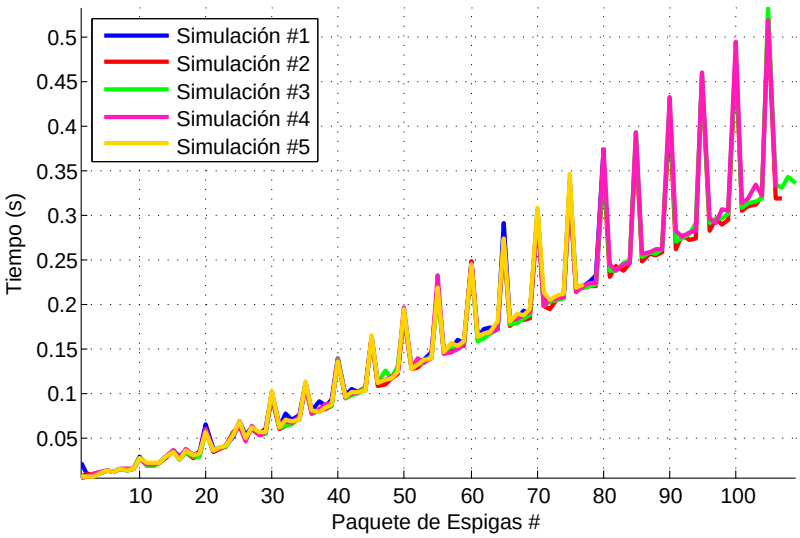


Figura 5.1: Tiempo de procesamiento de SPC *on-line*.

[2] Todas las mediciones de tiempo se realizaron con un procesador Intel Core2Duo de 2 GHz

5.3.3. Limitaciones

Se realizaron pruebas con una simulación de 1200 s para poder evaluar hasta donde era posible aplicar SPC *on-line* antes de empezar a notar una degradación en el desempeño, tanto respecto del tiempo de procesamiento como de la calidad de los agrupamientos. Se obtuvieron las primeras 15000 espigas de la señal, se les aplicó la extracción de características y posteriormente se ejecutó el algoritmo SPC *on-line*. En la figura 5.2 se pueden ver las distintas clases de espigas (la tabla 5.7 muestra la cantidad exacta de cada una).

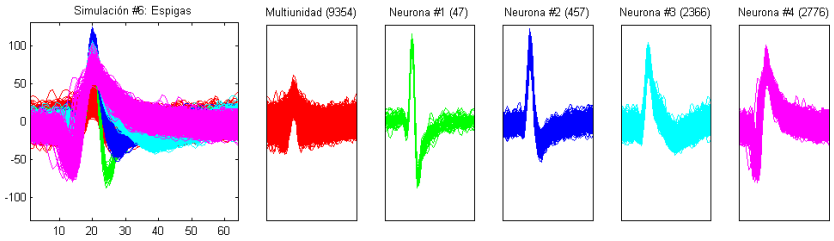


Figura 5.2: Espigas de la simulación (primeras 15000).

	Clase de espiga					
	Multi-unidad	N. #1	N. #2	N. #3	N. #4	Total
Simulación #6	9354	47	457	2366	2776	15000

Cuadro 5.7: Detalle de los grupos de espigas de la simulación (primeras 15000).

Se observó que a medida que se incrementa la cantidad de espigas, empieza a aumentar el porcentaje de no clasificadas, ubicándose entre un 5 y 25 %. Si bien es un porcentaje alto, se mantiene estable a medida que se van agregando. Esto significa que cada vez toma más tiempo clasificar una espiga. Este inconveniente se puede mitigar modificando el parámetro que indica cada cuantas espigas aplicar la simulación, sin embargo, impacta de manera directa en el tiempo de procesamiento. Una solución alternativa es realizar *Template Matching* de las espigas no clasificadas con los grupos identificados. Como puede verse en la figura 5.6 el agrupamiento conseguido es muy bueno. Solo faltó identificar a un grupo, las espigas de la neurona #1, tarea dificultada por el bajo ritmo de activación de la misma (solamente hay 47 de estas en un total de 15000). Por lo tanto, el resto de las espigas no clasificadas se pueden designar rápidamente a alguno de los grupos en-

contrados (recordar que esta técnica es bastante rápida) y con esto reducir al mínimo la cantidad de espigas no clasificadas. En la figura 5.3 se puede ver la evolución de los porcentajes de clasificación a medida que se agregan espigas.

Otra observación es que el tiempo de procesamiento se incrementa notablemente pasadas las 10000 espigas, producto del costo de insertar un nuevo vértice, ya que hay que calcular los vecinos más cercanos al mismo, y de la simulación.

En la figura 5.4 se puede ver como aumenta el tiempo de procesamiento a medida que se agregan los paquetes de espigas. Los picos que se veían en la sección anterior toman mayor amplitud a medida que aumenta la cantidad de espigas y está en relación directa al algoritmo de simulación.

La figura 5.5 muestra la cantidad de paquetes procesados por segundo. La línea roja representa la mínima cantidad de paquetes que deben procesarse para que el procedimiento se haga en tiempo real (es decir, 1). Se puede observar que el límite está cercano a los 175 paquetes (marcado mediante el punto verde), equivalente a más de 5000 espigas.

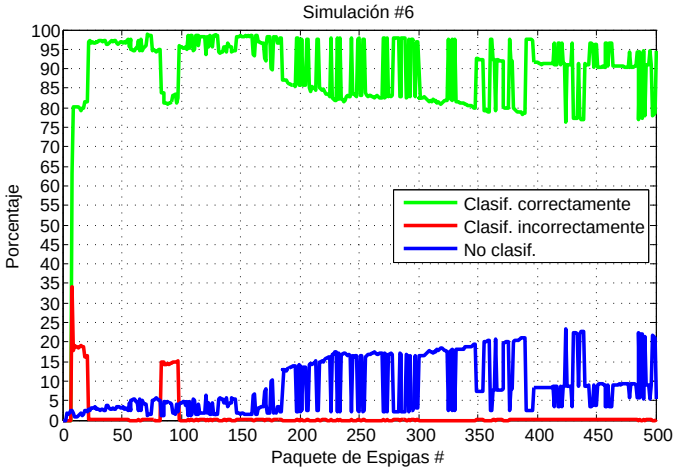


Figura 5.3: Porcentajes de clasificación de SPC *on-line* para 15000 espigas.

Finalmente, vale realizar una aclaración respecto de los niveles de activación de las neuronas. Aquí se asumió una frecuencia de 30 espigas por segundo, puesto que las simulaciones con que se realizaron las pruebas así lo marcaban. En datos experimentales, la frecuencia de activación es bastante

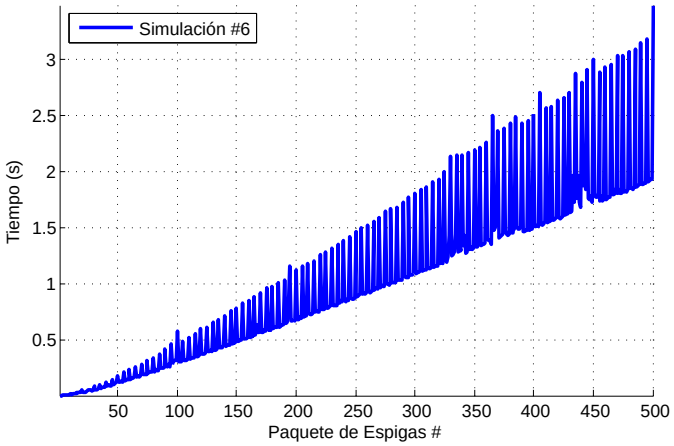


Figura 5.4: Tiempo de procesamiento de SPC *on-line* para 15000 espigas.

menor, aproximadamente 5 Hz [24]. Esto no modifica en absoluto los resultados obtenidos sino que indica que la longitud de la señal sobre la cual se puede aplicar el algortimo en tiempo real es mucho mayor, ya que la cantidad de espigas encontradas en la misma cantidad de tiempo es mucho menor.

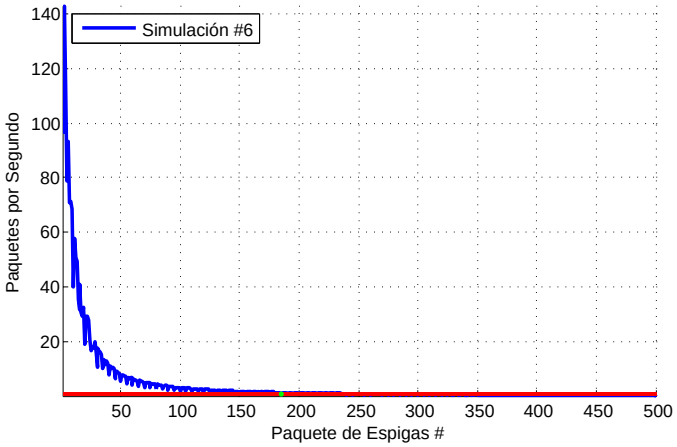


Figura 5.5: Cantidad de paquetes procesados por segundo de SPC *on-line* para 15000 espigas.

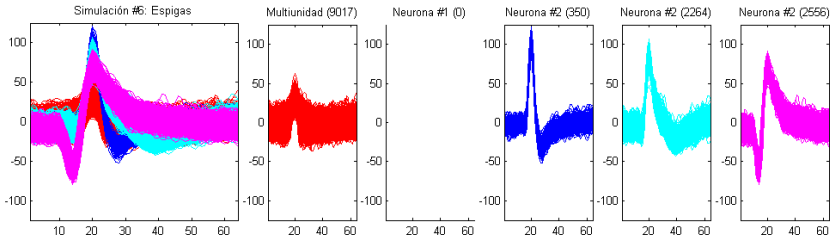


Figura 5.6: Clasificación final de SPC *on-line* para 15000 espigas.

5.4. Desempeño del método de ordenamiento de espigas *on-line*

A continuación se presenta una comparación entre los resultados arrojados por *Wave-Clus* y la versión de ordenamiento de espigas *on-line* aquí presentada.

La tabla 5.8 muestra la cantidad de espigas detectadas por cada implementación junto a la real. Puede verse que en la implementación *on-line* la cantidad es levemente menor a las detectadas por *Wave-Clus*. Probablemente, debido a espigas que no pueden ser detectadas por encontrarse en los bordes de los segmentos de 1 s que procesa la versión *on-line*.

	Wave-Clus	Ordenamiento *on-line*	Cantidad real
Simulación #1	1759	1749	2415
Simulación #2	2462	2451	3214
Simulación #3	2579	2568	3283
Simulación #4	2525	2523	3193
Simulación #5	1599	1593	2328

Cuadro 5.8: Espigas detectadas en ambas implementaciones.

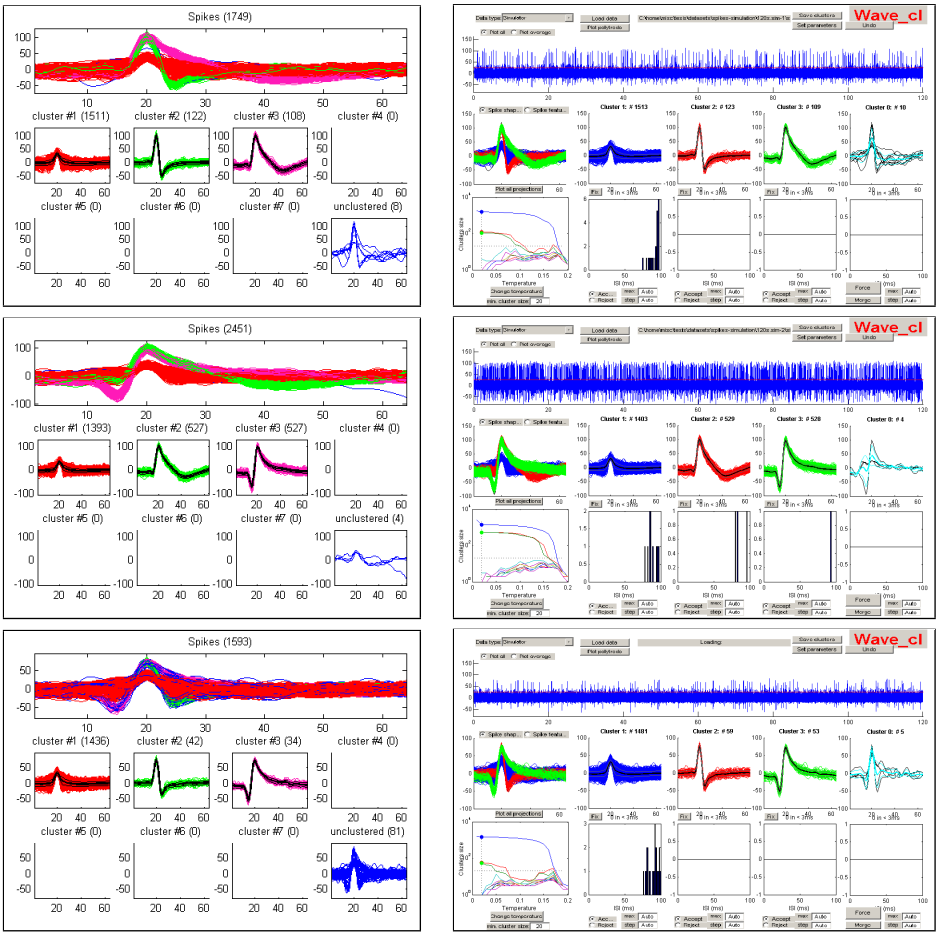
En segundo, lugar se muestra la clasificación de espigas final (puesto que *Wave-Clus* solo muestra esta información). En este caso, como se está realizando el proceso de clasificación sobre las espigas *detectadas*, no es posible hacer una medición exhaustiva de la clasificación (como sí se hizo en la sección de comparación de SPC contra SPC *on-line*). Sin embargo, podemos ver el resultado final de ambas implementaciones y realizar una comparación visual.

Se experimentó con las 4 alternativas planteadas para la extracción de características: a) Utilizar los 64 coeficientes de las espigas, b) Aplicar la transformada wavelet y utilizar los 64 coeficientes de ésta, c) Aplicar la transformada wavelet y aplicar el test de normalidad a todas las espigas para obtener los 10 coeficientes que representan a cada espiga y d) Ídem anterior pero el test se aplica sobre las últimas espigas detectadas.

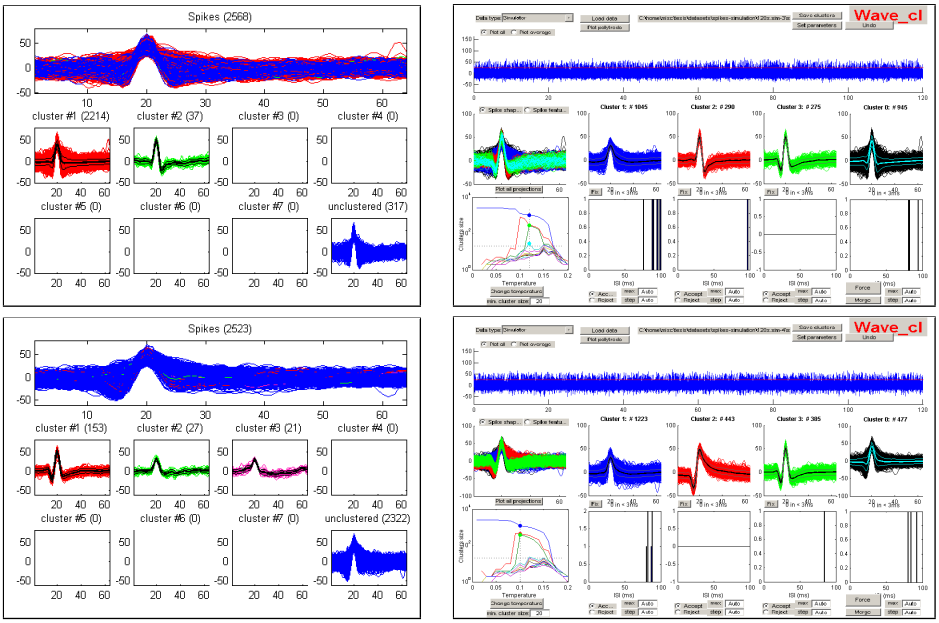
Respecto de las estrategias, tanto la *a)* como la *b)* se comportaron de manera similar y mejor que las *c)* y *d)*. El test de normalidad trae beneficios al aplicarlo tal cual se plantea en el método de Quian Quiroga *et al.* [7] y

los efectos de no usarlos se notan, particularmente, en las simulaciones más difíciles, cuando las espigas son más parecidas entre sí.

Se obtuvieron muy buenos resultados para las simulaciones #1, #2 y #5, aunque el desempeño general fue levemente menor a los presentados en la sección 5.2. Para las simulaciones #3 y #4, el resultado no fue satisfactorio, se acentuó el problema de inestabilidad ya comentado. Los motivos son, en primer lugar, que se está realizando el proceso con espigas detectadas lo cual puede introducir errores en las formas de las mismas. En segundo lugar, el no uso o uso "inadecuado"del test de normalidad dificulta la correcta identificación de las diferentes clases. Esto se acentúa mucho para simulaciones difíciles como la #3 y #4. En los cuadros 5.9 y 5.10 se pueden ver los ordenamientos finales para cada simulación y para cada implementación.



Cuadro 5.9: Capturas de pantalla de los ordenamientos que se obtuvieron para las simulaciones #1, #2 y #5, respectivamente. En la columna de la izquierda se encuentran las generadas por el método propuesto y en la derecha por *Wave-Clus*.



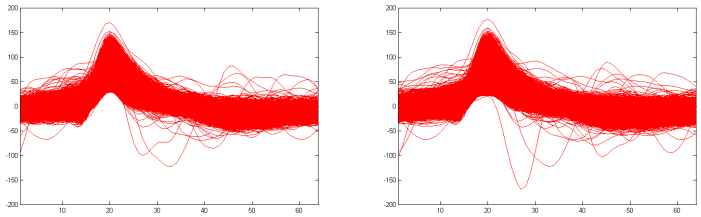
Cuadro 5.10: Capturas de pantalla de los ordenamientos que se obtuvieron para las simulaciones #3 y #4, respectivamente. En la columna de la izquierda se encuentran las generadas por el método propuesto y en la derecha por *Wave-Clus*.

Por último, se aplicaron ambas versiones sobre un registro de actividad neuronal real [25] de 30 min del lóbulo temporal de un paciente epiléptico del laboratorio Itzhak Fried de la Universidad de California, Los Ángeles.

	Wave-Clus	Ordenamiento *on-line*
Registro UCLA #1	12734	8056

Cuadro 5.11: Cantidad de espigas detectadas por ambas implementaciones.

La primera observación para hacer está en la cantidad de espigas detectadas. Se puede ver en la tabla 5.11 que la implementación *on-line* detecta menos que *Wave-Clus*. Esto mismo ocurre con las simulaciones pero aquí se ve intensificado dada la longitud de la señal (30 min contra 2 min). Respecto de la forma de las espigas, puede verse que, en su conjunto, son parecidas entre una implementación y otra (figura 5.12).



Cuadro 5.12: Conjunto de espigas detectadas (las de la izquierda corresponden al método *on-line*, los de la derecha a *Wave-Clus*).

Respecto de la clasificación, se observaron resultados similares a los ya comentados. La separación fue buena y estable. La cantidad de no clasificados empezó a incrementarse a medida que se fueron agregando cada vez más espigas pero se mantuvo dentro del margen previamente discutido.

En cuanto al tiempo de procesamiento, hubo un incremento (respectos de los expuestos en la sección 5.3.2) ya que se están utilizando 64 coeficientes para representar a cada espiga. Aumentar la dimensionalidad de esta manera (más de 6 veces) tiene su costo. Calcular los vecinos más cercanos al punto agregado toma mucho más tiempo y, como es de esperarse, se incrementa a medida que aumentan la cantidad total de espigas.

Capítulo 6

Conclusiones y trabajos futuros

En este trabajo se presentó un método de ordenamiento de espigas *on-line* basado en el de Quian Quiroga *et al.* [7].

Se analizó y modificó el algoritmo SPC [12, 13], principal limitación para llevar el método de Quian Quiroga *et al.* [7] a modo *on-line*, para que soporte la inserción dinámica de nuevos datos. Se compararon los dos algoritmos de Monte Carlo presentados para la simulación del modelo de Potts [17]: Swendsen-Wang [20] y Wolff [21]. También se analizaron distintas estrategias para la actualización de los datos luego de insertar un nuevo punto. Se logró presentar una nueva versión del algoritmo junto con una implementación del mismo que permite procesar en el orden de las 5000 espigas *on-line* en tiempo real.

La implementación presentada fue necesaria para alcanzar el objetivo por dos motivos. El primero involucra al desempeño del método, no se hubiese podido procesar la cantidad de espigas mencionadas si no se hubiese contado con una implementación eficiente. El segundo tiene que ver con brindar un marco de trabajo en donde poder experimentar con las diferentes estrategias y modificaciones propuestas, que permitieron llegar al algoritmo aquí presentado.

Contar con un algoritmo de SPC *on-line* brinda varias ventajas que

pueden ser aprovechadas para un mejor desempeño a nivel global. Además de la mencionadas en la introducción, da la posibilidad de cambiar parámetros de manera dinámica durante el transcurso de la ejecución del mismo. Por ejemplo, se puede cambiar el umbral utilizado para la construcción de los grupos basándose en el ruido actual de la señal, incrementándolo si es elevado o decrementándolo en caso contrario. De esta forma, se puede alcanzar una mayor automatización del método.

Se encontraron límites en la adaptación del algoritmo SPC al modo *on-line*. El tiempo de procesamiento se incrementa, indefectiblemente, con la cantidad de datos agregados. Esto impone restricciones a la hora utilizar el método de ordenamiento de espigas en experimentos de una duración de tiempo no acotada. Las posibilidades de un mejoramiento del algoritmo se ven afectadas por los algoritmos de simulación. Estos tienen un orden de complejidad $\mathcal{O}(NK)$ (donde N es la cantidad de vértices y K es la cantidad de vecinos) el cuál tiene un impacto directo en la performance. Estos algoritmos difícilmente se puedan modificar para mejorar este aspecto. Por lo tanto, los únicos lugares para el mejoramiento son los que fueron cubiertos.

Como una de las conclusiones más importantes, se llegó a que la mejor opción para atacar el problema es el procesamiento por ventanas de tiempo. Se estableció que el método propuesto es suficiente bueno y estable para una gran cantidad de espigas que, de acuerdo a la frecuencia de activación de las neuronas registradas, puede ir desde 3 a 15 min. Por lo tanto, se pueden procesar de esta manera ventanas de X minutos, y combinar el resultado obtenido con los anteriores. De esta manera, la etapa de clasificación se podría llevar a aplicar de manera *on-line* y en tiempo real a una señal *arbitrariamente grande* y con los *mejores resultados*.

Respecto de la integración del método, se mostró que es posible. Se deberán dedicar mayores esfuerzos a encontrar una mejor manera de extraer las características de las espigas. Si bien se considera que la aplicación de la transformada wavelet otorga una buena base para representar a las espigas, se deberá replantear cómo se seleccionan los coeficientes sin la necesidad de contar con todas las espigas de la señal. Como se vio, se pueden utilizar los 64 coeficientes de la transformada, sin embargo, cuando la calidad de la señal es baja o cuando hay dos clases de espigas muy parecidas entre sí, es necesario ayudar al método de clasificación brindando un vector de

características que represente lo mejor posible y separe lo más posible a cada clase. Esto también es deseable desde un punto de vista de tiempo de procesamiento puesto que la dimensionalidad de los datos impacta, aunque en menor medida, en el tiempo total de ejecución de la etapa de clasificación.

Bibliografía

[1] D. Hubel *et al.*, "Tungsten microelectrode for recording from single units," *Science*, vol. 125, no. 3247, pp. 549–550, 1957.

[2] D. Hubel and T. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[3] R. Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried, "Invariant visual representation by single neurons in the human brain," *Nature*, vol. 435, no. 7045, pp. 1102–1107, 2005.

[4] G. Buzsáki, "Large-scale recording of neuronal ensembles," *Nature neuroscience*, vol. 7, no. 5, pp. 446–451, 2004.

[5] I. Jolliffe and MyiLibrary, *Principal component analysis*, vol. 2. Wiley Online Library, 2002.

[6] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, p. 14, California, USA, 1967.

[7] R. Quiroga, Z. Nadasdy, Y. Ben-Shaul, *et al.*, "Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering.," *Neural computation*, vol. 16, no. 8, p. 1661, 2004.

[8] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 11, no. 7, pp. 674–693, 1989.

[9] J. Letelier and P. Weber, "Spike sorting based on discrete wavelet transform coefficients," *Journal of neuroscience methods*, vol. 101, no. 2, pp. 93–106, 2000.

[10] E. Hulata, R. Segev, Y. Shapira, M. Benveniste, and E. Ben-Jacob, "Detection and sorting of neural spikes using wavelet packets," *Physical review letters*, vol. 85, no. 21, pp. 4637–4640, 2000.

[11] E. Hulata, R. Segev, and E. Ben-Jacob, "A method for spike sorting and detection based on wavelet packets and shannon's mutual information," *Journal of Neuroscience methods*, vol. 117, no. 1, pp. 1–12, 2002.

[12] M. Blatt, S. Wiseman, and E. Domany, "Superparamagnetic clustering of data," *Physical review letters*, vol. 76, no. 18, pp. 3251–3254, 1996.

[13] M. Blatt, S. Wiseman, and E. Domany, "Data clustering using a model granular magnet," *Neural Computation*, vol. 9, no. 8, pp. 1805–1842, 1997.

[14] J. Martinez, C. Pedreira, M. Ison, and R. Quian Quiroga, "Realistic simulation of extracellular recordings," *Journal of neuroscience methods*, vol. 184, no. 2, pp. 285–293, 2009.

[15] S. Butterworth, "On the theory of filter amplifiers," *Wireless Engineer*, vol. 7, pp. 536–541, 1930.

[16] B. Flannery, W. Press, S. Teukolsky, and W. Vetterling, "Numerical recipes in c," *Press Syndicate of the University of Cambridge, New York*, 1992.

[17] F. Wu, "The potts model," *Reviews of modern physics*, vol. 54, no. 1, p. 235, 1982.

[18] P. Mitra and H. Bokil, *Observed brain dynamics*. Oxford University Press, USA, 2008.

[19] "Spike sorting." `http://www.scholarpedia.org/article/Spike_sorting`, 2007.

[20] J. Wang and R. Swendsen, "Cluster monte carlo algorithms," *Physica A: Statistical and Theoretical Physics*, vol. 167, no. 3, pp. 565–579, 1990.

[21] U. Wolff, "Comparison between cluster monte carlo algorithms in the ising model," *Physics Letters B*, vol. 228, no. 3, pp. 379–382, 1989.

[22] F. Niedermayer, "Improving the improved estimator in o(n) spin models," *Physics Letters B*, vol. 237, no. 3, pp. 473–475, 1990.

[23] "Simulated extracellular recordings." `http://www2.le.ac.uk/departments/engineering/research/bioengineering/neuroengineering-lab/software`, 2009. Dataset # 3: Simulated extracellular recordings.

[24] M. Ison, F. Mormann, M. Cerf, C. Koch, I. Fried, and R. Quiroga, "Selectivity of pyramidal cells and interneurons in the human medial temporal lobe," *Journal of Neurophysiology*, vol. 106, no. 4, pp. 1713–1721, 2011.

[25] "Human single-cell recording." `http://www.vis.caltech.edu/~rodri/Wave_clus/UCLA_data.zip`. Dataset # 1: Human single-cell recording.