UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DEL MEDIO NATURAL
GRADO EN BIOTECNOLOGÍA - CURSO ACADÉMICO 2018 - 2019

Comparación de métodos de detección de selección positiva

Autora: Marta Benegas Coll
Tutor: Jose Miguel Blanca Postigo

Valencia, Junio 2019.

COMPARACIÓN DE MÉTODOS DE DETECCIÓN DE SELECCIÓN POSITIVA

RESUMEN

Las especies cultivadas son fruto de un proceso de domesticación en el que los agricultores han ejercido una selección artificial en búsqueda de caracteres agronómicos de interés. Este proceso de selección afecta a la estructura de la variación genética presente en estas especies. Por ejemplo, en una región genómica seleccionada por favorecer la aparición de frutos grandes se seleccionará un alelo concreto en detrimento del resto.

El objetivo de este trabajo es la comparación de diferentes *software* de búsqueda de marcas de selección a lo largo del genoma. La identificación de regiones que han sufrido selección podría utilizarse para localizar genes que son importantes durante el proceso de domesticación, información que puede resultar valiosa para los agricultores y mejoradores.

En este trabajo se utiliza el tomate cultivado, *Solanum lycopersicum*, para estudiar el efecto del proceso de selección en el genoma, ya que se dispone de alrededor de 2000 muestras de tomate genotipadas, tanto de variedades tradicionales europeas como de especies silvestres relacionadas.

Palabras clave: selección, barrido selectivo, tomate, *Solanum lycopersicum*

ABSTRACT

Cultivated species are the result of a domestication process in which farmers have made an artificial selection, searching for agronomic characters of interest. This selection process affects the structure of the genetic variation present in these species. For example, in a genomic region selected because it favours the production of larger fruits, a specific allele will be selected above the rest.

The aim of this project is the comparison between different genome-wide selection detection softwares. The identification of regions that have undergone selection could be used to locate important genes for the domestication process, information that may be valuable for farmers and breeders.

In this work, the cultivated tomato, Solanum lycopersicum, is used to study the effect of the selection process on the genome, since there are about 2000 samples of genotyped tomato, both of traditional European varieties and of related wild species.

Keywords: *selection, selective sweep, tomato, Solanum lycopersicum*

Alumna: Marta Benegas Coll
Profesor D. Jose Miguel Blanca Postigo

Valencia, Junio de 2019.

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

El tomate es uno de los cultivos más importantes a nivel mundial (FAOSTAT, 2019). El tomate cultivado (*Solanum lycopersicum L.*) pertenece a la familia de las solanáceas, género *Solanum*, sección *Lycopersicon* (Peralta et. al., 2019). *Solanum lycopersicum* presenta dos variedades: *Solanum lycopersicum* var. *cerasiforme* (SLC), el cual crece en zonas de América del Sur tanto de forma silvestre como cultivada, y *Solanum lycopersicum* var. *lycopersicum* (SLL) (Blanca et. al., 2015), del cual deriva el tomate moderno. El ancestro silvestre más cercano es la especie *Solanum pimpinellifolium L.* (SP), originario de la costa de Perú y Ecuador (Blanca et. al., 2015), a partir del cual se dió el proceso de domesticación del tomate. La domesticación es la modificación genética de las especies silvestres para dar lugar a una nueva planta que se adapta a las necesidades humanas (Doebley et. al., 2006). Esta modificación genética generalmente se produce por selección artificial a partir de plantas silvestres, en la cual los humanos seleccionan y cultivan únicamente plantas con determinados fenotipos que consideran apropiados, como pueden ser un tamaño de fruto grande, resistencias a enfermedades, adaptación a nuevos ambientes de cultivo etc. En consecuencia, la domesticación produce cambios drásticos en el fenotipo, hasta el punto que algunas especies cultivadas son totalmente dependientes de los humanos y ya no son capaces de crecer de forma silvestre (Doebley et. al., 2006).

Así pues, llevando a cabo análisis genómicos comparativos entre las especies silvestres y cultivadas se pueden identificar genes importantes en el proceso de domesticación; de adaptación a nuevos ambientes, productividad, etc., lo que puede resultar una información muy valiosa para los agricultores y mejoradores (Blanca et. al., 2015). Por ello, en este trabajo se pretende identificar regiones genómicas que han sido seleccionadas partiendo de datos de variación genética de diferentes variedades de tomate cultivado Europeo y especies silvestres relacionadas (*Solanum pimpinellifolium*, *Solanum galapagense* y *Solanum chesmaniae*).

Para ello, antes de llevar a cabo análisis genéticos, primero es importante tener una idea de la historia del tomate y de las características de su genoma. Se piensa que el proceso de domesticación del tomate se dió en dos pasos: una primera selección de los SP y SLC primitivos en el norte de Perú y Ecuador y otra posterior selección de los SLC pre-domesticados en Mesoamérica, después de su migración desde Perú y Ecuador (Blanca et. al., 2015). Posteriormente, estos tomates de Mesoamérica se introdujeron en Europa. Una vez domesticado, el cultivo extensivo del tomate ha modificado sus características, centrándose en la mejora de SLL para aumentar la resistencia a enfermedades, aumento del rendimiento y uniformidad (Blanca et. al., 2015). De esta forma, los cuellos de botella sufridos durante la migración a Mesoamérica y Europa, el cultivo extensivo y el hecho de que el tomate es una planta casi autógama (Blanca et. al., 2012) hacen que el tomate cultivado sea una especie con muy poca diversidad genética y con un alto grado de homocigosidad (Blanca et. al., 2012). Además, el genoma del tomate presenta una zona pericentromérica muy grande, la mayor densidad de genes se encuentra a los extremos de los cromosomas (The Tomato Genome Consortium, 2012).

El proceso de domesticación consiste en modificar el material genético presente en la especie silvestre seleccionándolo para obtener las poblaciones cultivadas (Doebley et. al., 2006). Desde el punto de vista de la genética de poblaciones este proceso consiste en modificar las frecuencias alélicas de las poblaciones silvestres para generar las poblaciones domesticadas. La modificación de estas frecuencias se puede describir mediante las fuerzas microevolutivas clásicas: selección, deriva y mutación (Hartl y Clark, 2018).

Cuando un agricultor selecciona un carácter de interés está modificando la eficacia biológica de los distintos alelos de los genes que controlan ese carácter (Doebley et. al., 2006). Esta selección hace que a medida que se suceden las generaciones las frecuencias alélicas de los alelos considerados beneficiosos aumente en la población, de modo que paulatinamente la frecuencia de los alelos silvestres irá disminuyendo (Hartl y Clark, 2018).

Habitualmente asociado al proceso de domesticación hay un cuello de botella, es decir, una disminución en el número de individuos de la población (Hartl y Clark, 2018). Las poblaciones cultivadas suelen fundarse a partir de un número de individuos silvestres y el proceso de selección continuada a lo largo del tiempo reduce en numerosas ocasiones el número de individuos que se crean la siguiente generación. El principal efecto de estos cuellos de botella es aumentar la deriva genética. La deriva puede caracterizarse como el cambio de frecuencias alélicas debido al muestreo aleatorio de individuos a una población (Hartl y Clark, 2018). Por ejemplo, si en una población se parte de cien individuos, cincuenta homocigotos para un alelo *a* y cincuenta homocigotos para otro alelo *A*, en la siguiente generación lo más probable es que la frecuencia de estos alelos varíe ya que no todos los individuos tendrán un mismo número de descendientes. El grado de variación de estas frecuencias alélicas, es decir, la magnitud de la deriva, dependerá de cómo de reducido sea el número de individuos que contribuyen a la siguiente generación. Por lo tanto las poblaciones que sufren un cuello de botella, una reducción en el número de individuos que la componen, son afectadas muy fuertemente por la deriva (Hartl y Clark, 2018). El efecto de la deriva es eliminar variación.

La última de las fuerzas microevolutivas más habituales es la mutación. En una población aislada toda la variación alélica se genera mediante mutación (Hartl y Clark, 2018). Los alelos seleccionados favorablemente durante el proceso de domesticación habrán aparecido por mutación. En algunos casos estas mutaciones se habrían dado en la población silvestre y los primeros agricultores las seleccionaron a partir de las plantas que veían en el campo y en otros habrán ocurrido posteriormente en los campos de cultivo.

Cuando una nueva mutación se da en un individuo de una población, inicialmente se encuentra en muy baja frecuencia. Si la mutación es neutral, lo más común es que termine desapareciendo por la deriva. Sin embargo, puede ocurrir que, por azar, aumente su frecuencia a lo largo de muchas generaciones (Vitti et. al, 2013). Es decir, en una población habrá algunos loci que tendrán alelos en alta frecuencia debido a la deriva. En cambio, si la mutación confiere una ventaja biológica, es decir, se encuentra bajo selección, aumentará rápidamente su frecuencia en la población. Este proceso es conocido como barrido selectivo (*selective sweep*) (Maynard Smith y Haigh, 1974). Este aumento de la frecuencia estará acompañado de un aumento de la homocigosidad (o reducción en la heterocigosidad) en el sitio que ha sido seleccionado y en las regiones colindantes (Maynard Smith y Haigh, 1974; Kim y Stephan, 2002).

Además, las regiones cercanas al locus seleccionado también sufrirán una reducción de diversidad, efecto conocido como *hitchhike effect* (Maynard Smith y Haigh, 1974). Esta disminución en la diversidad se debe a que es poco probable que se den recombinaciones en las regiones cercanas la mutación seleccionada, por lo que también aumenta la frecuencia de los alelos ligados a ésta mutación. Este efecto es menor en regiones más alejadas del sitio seleccionado, debido a que la recombinación rompe la asociación entre el sitio seleccionado y loci más alejados, por lo que el tamaño de la región afectada por el *hitchhike effect* también depende del ratio de recombinación (Kim y Stephan, 2002). Esto

produce, a su vez, un aumento drástico del desequilibrio de ligamiento, otra característica importante de las regiones seleccionadas. Así pues, en una región seleccionada aparecerá un bloque haplotípico con una frecuencia y un desequilibrio de ligamiento muy altos y, por lo tanto, una diversidad pequeña.

Hay que tener en cuenta que a medida que pasa el tiempo aparecen nuevas mutaciones que pueden ir acumulándose en la región que ha sido seleccionada. La rapidez con la que se acumulen dependerá de la tasa de mutación y de si ha dejado de ejercerse la presión selectiva. Estos nuevos alelos, con el tiempo, restablecerán la diversidad, aunque durante un tiempo ésta continuará siendo baja. El desequilibrio de ligamiento también seguirá siendo elevado durante un tiempo porque, aunque se creen nuevos haplotipos ligeramente diferentes al original, estos siguen trasmitiéndose en conjunto.

Otra propiedad característica de las regiones seleccionadas el la alteración en el espectro de frecuencias esperado (Fay y Wu, 2000). El espectro de frecuencias es la distribución del número de sitios polimórficos con una determinada frecuencia. Bajo un supuesto de selección, esta distribución se desplaza hacia un exceso de alelos en alta y baja frecuencia (Braverman et. al., 1995, Fay y Wu, 2000). Los alelos en alta frecuencia son aquéllos que han sido seleccionados y los de baja frecuencia son los alelos seleccionados en contra y los nuevos alelos que han aparecido en loci cercanos al seleccionados y asociados al alelo seleccionado, pero que todavía no han no han sufrido el proceso de deriva durante mucho tiempo. Este efecto es muy característico de las regiones que han sufrido selección, por lo que para detectar estas regiones, es bastante común estudiar el espectro de frecuencias por sitio (SFS, *site frequency spectrum*) buscando un exceso de alelos en alta y baja frecuencia (Ronen et. al, 2013).

Las características descritas hasta el momento son típicas de un barrido selectivo fuerte, en el que se ha seleccionado a favor de un único alelo que aparece *de novo*. Sin embargo, también es muy común el barrido selectivo suave (*soft sweep*), en el que se selecciona para más de un haplotipo (Jensen, 2014). Esto puede producirse por dos causas. Una es por la llamada “selección en la variabilidad preexistente”, es decir, no se selecciona una mutación que ha aparecido *de novo* sino que se selecciona un alelo que en un principio era neutral pero, debido a cambios ambientales u otros, pasa a conferir una ventaja. Este alelo es bastante probable que se encontrara en haplotipos diferentes, por lo que la selección actuará sobre todos ellos (Vitti et. al, 2013; Jensen, 2014). Otra posibilidad es que se seleccionen nuevas mutaciones en cadena, es decir, que la misma mutación que confiere una ventaja se de primero en un haplotipo y posteriormente vuelva a aparecer en el mismo sitio pero en un haplotipo distinto, por lo que, igual que antes, se seleccionará para más de un haplotipo distinto (Vitti et. al, 2013; Jensen, 2014). En ambos casos el desequilibrio de ligamiento será alto dentro de cada haplotipo seleccionado, igual que ocurre con el barrido selectivo fuerte (Vitti et. al, 2013). La diferencia está en que la reducción de la diversidad no es tan pronunciada y, sobre todo, que la desviación del espectro de frecuencias es diferente, ya que en este caso se produce un exceso de alelos neutrales que están en una frecuencia intermedia (Jensen, 2014). Esto último es lo que se suele buscar para distinguir un barrido selectivo fuerte de uno débil.

Además, también hay otros procesos que pueden alterar el genoma de una forma similar. Hasta ahora se ha estado hablando del efecto que produce la propagación de una mutación que confiere una ventaja biológica. Sin embargo, también puede darse el caso contrario: que aparezca una mutación que confiera una desventaja y, por tanto, se seleccione en contra de ésta. Este proceso se conoce como selección negativa o selección purificadora.

De hecho, las mutaciones deletéreas aparecen con bastante frecuencia (Keightley y Eyre-Walker, 1999), por lo que la selección purificadora produce grandes cambios en el genoma (Hudson y Kaplan, 1995; Charlesworth, 1996). La eliminación rápida del nuevo alelo deletéreo también produce la eliminación de los alelos ligados a éste, proceso llamado selección de fondo (*background selection*) (Charlesworth et. al., 1993), el cual produce una reducción drástica de la diversidad en esa región (Charlesworth et. al., 1993; Charlesworth, 2009; Charlesworth, 2012). De esta forma, el aumento de la frecuencia de un alelo beneficioso y los ligados a éste implica una reducción en la frecuencia del otro alelo y de los ligados a éste, y viceversa; la reducción en la frecuencia de un alelo deletéreo y los ligados a éste implica un aumento en la frecuencia del otro alelo y los ligados a éste. Por tanto, tanto el barrido selectivo, explicado anteriormente, como la selección de fondo producen efectos muy similares sobre el genoma y es difícil distinguirlos, por lo que es habitual hablar de *hitchhike effect* para referirse a ambos procesos (Kim y Stephan, 2000; Campos et. al. 2017). Por otra parte, los cuellos de botella poblacionales (*population bottleneck*) también son una causa de la reducción de la diversidad. Como ya se ha explicado, los cuellos de botella son una reducción drástica de los individuos de una población, que puede ser causada por diferentes factores (enfermedades, hambrunas, sequías, etc.), lo que resulta en la desaparición de muchos alelos y, por tanto, una disminución de la diversidad (Nei et. al., 1975; Cornuet y Liukart, 1996). Sin embargo, lo que caracteriza y diferencia un cuello de botella de una selección es que inmediatamente después de éste hay un exceso de heterocigosidad (Nei et. al., 1975; Cornuet y Liukart, 1996), es decir, la heterocigosidad observada es mucho más alta que la esperada con ese número de alelos, a diferencia de un proceso de selección que se caracteriza por un exceso de homocigosidad. Esto ocurre porque la mayoría de alelos que desaparecen son aquéllos que están en baja frecuencia, por lo que el nivel de heterocigosidad no se ve tan afectado (Nei et. al., 1975; Cornuet y Liukart, 1996). Otra diferencia es que un cuello de botella reduce la diversidad de todo el genoma (Cornuet y Liukart, 1996), mientras que la selección actúa en una región en concreto.

A lo largo de la década de los 90 se desarrollaron los varios estadísticos diseñados para identificar regiones seleccionadas (Hudson et. al, 1987; Tajima, 1989; Wolfgang, 1992; Kelly, 1997; Fay y Wu, 2000). Estos estadísticos cuantifican las modificaciones que la selección produce en el genoma para ver si se alejan de los valores supuestos bajo una evolución neutral. Algunos miden el exceso de desequilibrio de ligamiento (Kelly, 1997), otros reducciones drásticas en la diversidad (Wolfgang, 1992) o desviaciones respecto del SFS esperado (Fay y Wu, 2000), etc. Muchos de estos estadísticos dependen del parámetro θ (Fu, 1997), el cual se define como $4 \cdot Ne \cdot \mu$ (Kimura, 1968), siendo *Ne* el tamaño efectivo de la población y μ el ratio de mutación por secuencia por generación. Es decir, θ es una medida de la tasa de mutación neutral. Los valores de *Ne* y μ son difíciles de obtener, por lo que era necesario obtener una estimación de θ de forma indirecta (Fu, 1997) y llevar a cabo muchas simulaciones. Este proceso requiere mucho tiempo, por lo que por lo general se utilizaban para estudiar regiones genómicas pequeñas, para verificar genes que ya se sospecha que han sufrido selección (Tajima, 1989; Fay y Wu, 2000; Begun y Aquadro, 1991, Berry et. al 1991; Guttman y Dykhuizen, 1994).

En la actualidad es posible obtener una gran cantidad de datos genéticos gracias a las nuevas tecnologías de secuenciación. Actualmente es posible obtener la secuencia para una gran cantidad de individuos, además de las que ya hay disponibles en las bases de datos. Esta gran cantidad de información disponible ha facilitado mucho los análisis poblacionales (Nielsen et. al, 2005). Además no sólo ha aumentado la información

disponible, sino también la diversidad y eficiencia de herramientas bioinformáticas para llevar a cabo estos análisis.

Gracias a esto actualmente hay disponibles una gran variedad de programas para la búsqueda en cromosomas o incluso en genomas enteros de regiones que han sido seleccionadas (Nielsen et. al.,2005; Tang et. al, 2007; Antao et. al., 2008; Foll y Gaggiotti, 2008; Alachiotis et. al., 2012; Pavlidis et. al., 2013, Alachiotis y Pavlidis, 2018; Kern y Schrider, 2018). Esta identificación se lleva a cabo con el cálculo de estadísticos a partir de los datos sobre variación. El tipo de estadístico depende del programa utilizado y cada uno conlleva ciertas asunciones. En general, a diferencia de otros tests más antiguos, no utilizan un modelo teórico como modelo neutral frente al cual comparar si ha habido selección o no, sino utilizan la variabilidad media presente en el genoma. De esta forma, se tienen en cuenta procesos que alteran la variabilidad de todo el genoma como por ejemplo los cuellos de botella, a diferencia de la selección que actúa y altera la variabilidad únicamente en una región en concreto.

Habitualmente estos programas analizan archivos con el formato VCF, pero también hay otro tipo de formatos que guardan información sobre variaciones en el genoma, como puede ser el GESTE o GenePop. Algunos de los programas de detección de regiones que han sufrido selección más usados actualmente y que se han utilizado en este trabajo se describen a continuación.

DiploSHIC

DiploSHIC (Kern and Schrider, 2018) es un programa de aprendizaje automático para la detección de regiones que han sufrido un barrido selectivo, tanto fuerte como débil. DiploSHIC, como muchos otros programas, recorre el genoma en ventanas de un determinado número de pares de bases. Estas ventanas, a su vez, se dividen en 11 subventanas. Para cada una de ellas se calcula un total de 12 estadísticos diferentes, construyendo lo que llaman un vector de características (*feature vector*). Posteriormente, los 11 vectores de una ventana se representan en un mapa de calor (*heatmap*), que es analizado mediante una red neuronal convolucional (CNN, *convolutional neural network*) para, finalmente, clasificar la región en una de las 5 clases: barrido fuerte, barrido débil, región ligada a un barrido fuerte, región ligada a un barrido débil y región neutral.

Para poder hacer esta clasificación, dado que es un programa de aprendizaje automático, es necesario un paso previo de entrenamiento. Es decir, hay que indicarle al programa datos de cada una de las clases (especificando en cada caso a qué tipo pertenecen) para que los analice y aprenda a diferenciar unos de otros. Estos datos deben ser simulados con un programa externo.

Los 12 estadísticos calculados se clasifican en tres grupos: los que miden desviaciones en el espectro de frecuencias (SFS), los que miden el desequilibrio de ligamiento (LD) y los que analizan la estructura haplotípica.

SweeD

SweeD (Pavlidis et. al., 2013) es un programa de análisis de genomas enteros para la detección de regiones que han sufrido un proceso de barrido selectivo. Para detectarlo, analizan el espectro de frecuencias por sitio (SFS, *Site Frecuency Spectrum*) a lo largo de todo el cromosoma y buscan desviaciones respecto de la distribución esperada causadas por el barrido selectivo.

Este programa calcula la probabilidad de que una región haya sufrido un proceso de barrido selectivo mediante un ratio de probabilidades compuestas (CLR, *composite likelihood ratio test*). La probabilidad compuesta se calcula multiplicando cada una de las probabilidades marginales, es decir; para cada sitio a lo largo de una región cromosómica se calcula la probabilidad de que ese sitio haya sufrido un barrido selectivo, y estas probabilidades marginales se multiplican para obtener la probabilidad compuesta de toda región. A su vez, la probabilidad de que se haya dado un barrido selectivo se calcula como uno menos la probabilidad de que se encuentre el alelo derivado sin haber sufrido el proceso de selección. Por último, para calcular el CLR, se divide la probabilidad de cada región por el SFS promedio de todo el genoma. Se utiliza este parámetro como hipótesis nula y no un modelo neutral para así tener en cuenta de forma implícita procesos demográficos y otros fenómenos que podrían haber afectado al SFS de todo el genoma de la población como, por ejemplo, cuellos de botella, que reducen el nivel de diversidad a lo largo de todo el genoma en contraposición a un proceso de selección positiva, que actúa en una región en concreto.

BayeScan

BayeScan (Foll y Gaggiotti, 2008) es un programa de detección de regiones que han sufrido selección que se basa en la diferencia de frecuencias alélicas entre cada población con el *pool* genético. Esta diferencia se mide calculando el coeficiente F_{ST} para cada locus de cada una de las poblaciones. A continuación, para cada locus, se calcula el ratio de probabilidades posteriores, es decir, se divide la probabilidad de observar el F_{ST} calculado dado un modelo de selección positiva entre la probabilidad de haber observado ese mismo dato dado un modelo neutral. De esta forma, se obtiene un índice de cuánto más probable es que haya ocurrido un proceso de selección en ese locus respecto del modelo neutral. Estas probabilidades posteriores se calculan implementando un algoritmo de salto reversible MCMC para cada uno de los modelos (selección y neutral).

LOSITAN

LOSITAN (Antao et. al., 2008) es un programa de detección de regiones que han sufrido selección basado en la búsqueda de *locus* con valores atípicos de F_{ST}, comparados con el F_{ST} medio de todo el genoma, que se toma como valor neutral (dado que la mayoría de los *loci* del genoma no han sufrido presión selectiva, son neutrales).

Una primera aproximación del F_{ST} medio se obtiene teniendo en cuenta todos los *loci* del genoma, es decir, se incluyen también aquellos que sí que han sufrido selección positiva. Una vez calculado, todos los *loci* que presentan un valor de F_{ST} significativamente alejado de la media, tanto si son mayores o menores a ésta, son eliminados para volver a calcular el F_{ST} medio, para así obtener un valor más aproximado al F_{ST} medio neutral del genoma. Una vez obtenido este valor, se vuelve a escanear el genoma en busca de *loci* que presenten un valor de F_{ST} alejado de la media y que, por tanto, son candidatos de haber sufrido un proceso de selección.

RaisD

RaisD (Alachiotis y Pavlidis, 2018) es un programa de detección de regiones que han sufrido selección positiva que se basa en el cálculo de un estadístico propio, al que llaman μ, el cual está compuesto por tres factores: un factor que cuantifica los cambios en el SFS, otro que cuantifica el nivel de LD y un último que mide la cantidad de diversidad genética. Estos tres factores se multiplican entre ellos y por la longitud de la ventana para la cual se está calculando μ en pb. De esta forma, se tiene en cuenta en un mismo estadístico las tres principales marcas que el barrido selectivo deja en el genoma.

Este programa recorre cada cromosoma en ventanas de un determinado número de SNPs, en lugar de con ventanas de una determinada longitud en pares de bases, ya que afirman que una aproximación de este segundo tipo puede afectar de forma crítica a los resultados. Esto es debido a que, en caso de que la selección no haya sido muy fuerte o haya ocurrido hace mucho tiempo, hay una mayor probabilidad de que se hayan dado eventos de recombinación en una región de un determinado número de pares de bases, por lo que sea más difícil de identificar la señal de barrido selectivo (habría que reducir el tamaño de la ventana), mientras que, si la selección ha sido muy fuerte, puede ocurrir que no se encuentren SNPs en una región de esa misma longitud (habría que aumentar el tamaño de la ventana). Es por ello que, asumiendo que el ratio de mutación y de recombinación es aproximadamente el mismo a lo largo de todo el cromosoma y fijando el tamaño de la ventana en número de SNPs, se pretende estudiar regiones que proporcionen una señal más homogénea, facilitando la detección de los barridos selectivos.

2. Objetivos

- Realizar un análisis comparativo de diferentes programas de búsqueda de selección
- Identificación de regiones seleccionadas en diferentes subpoblaciones de tomate.

3. Materiales y métodos

3.1. Materiales

3.1.1 Material genómico

En este trabajo se utilizan los datos genómicos del proyecto TRADITOM (TRADITOM, 2019). En este proyecto se han genotipado muestras provenientes de distintas variedades de tomate tradicional Europeo así como de especies silvestres relacionadas (*Solanum pimpinellifolium*, *Solanum galapagense* y *Solanum chesmaniae*). El genotipado se ha realizado con la técnica GBS (*Genotype By Sequencing*) y la información sobre los SNPs identificados se ha almacenado en un archivo VCF (*Variant Call Format*), a partir del cual se llevan a cabo los análisis en este trabajo. En conjunto hay alrededor de 1600 muestras de distintas variedades y hasta 70000 SNPs para cada una de éstas.

La clasificación de las muestras en grupos genéticos (especies, variedades y poblaciones) se realizó mediante diversos análisis poblacionales llevados a cabo por el Doctor Jose Blanca. La información se encuentra en un archivo csv donde para cada muestra se indica el grupo genético al que pertenece.

Los archivos están disponibles en el Anexo XIII.

3.1.2 Herramientas informáticas

Python v.3.5.2

Python (PYTHON, 2019) es el lenguaje de programación utilizado en todos los programas escritos durante este proyecto. Este lenguaje se caracteriza por ser intuitivo y de fácil aprendizaje, además de ser un *software* libre. Por ello es ampliamente utilizado por la comunidad científica y hay disponibles una gran cantidad de librerías.

Para la representación de los datos obtenidos en figuras y gráficas se ha utilizado una de las librerías más ampliamente usadas en Python, Matplotlib (https://matplotlib.org/). Esta librería presenta una gran diversidad de opciones, lo que permite crear gráficas adaptadas al tipo de datos generados y al tipo de información que se pretenda representar. Además se ha utilizado seaborn, una librería que utiliza Matplotlib como base, pero que añade algunos gráficos predefinidos.

Otra librería importante en el desarrollo de este estudio es Scikit-learn (SCIKIT-LEARN, 2019). Esta librería está diseñada para el procesado y análisis de datos. Concretamente, se han utilizado las clases `PCA` (del módulo `decomposition`), para la realización de los análisis de componentes principales, y `StandardScaler` (del módulo `preprocessing`) para estandarizar los datos y así prepararlos para su análisis.

Además, para poder usar Scikit-learn es necesario que los datos se almacenen en matrices, una estructura de datos proporcionada por las librerías NumPy (NUMPY, 2019) y Pandas (PANDAS, 2019).

Por último, cabe destacar que para poder ejecutar comandos en la terminal desde un programa escrito en Python se ha utilizado el módulo `subprocess` de la Python Standard Library (THE PYTHON STANDARD LIBRARY, 2019). Este módulo se ha utilizado para poder ejecutar otros programas externos, lo cual resulta útil cuando queremos automatizar el análisis de una gran cantidad de archivos. De esta forma, escribiendo un único *script*, se puede ejecutar un mismo programa en reiteradas ocasiones automáticamente.

Programas de detección de selección

Para este trabajo se han utilizado los programas DiploSHIC , SweeD v.3.2.1, RaisD v1.6, BayeScan v.2.1 y LOSITAN, los cuales utilizan estadísticos basados en las diferentes marcas de selección que se han comentado en la introducción.

VCFtools v.0.1.5, BCFtools v.1.9

VCFtools (VCFTOOLS, 2019) y BCFtools (BCFTOOLS, 2019) son paquetes de programas que presentan diferentes herramientas para trabajar tanto con archivos de tipo VCF como con su versión binaria, el BCF. Estos formatos de archivos se utilizan para guardar datos de variación genética, los cuales pueden llegar a ser bastante complejos y extensos y, por tanto, difíciles de analizar.

Estos paquetes se pueden utilizar para realizar distintos tipos de filtrado (por cromosoma, región, individuos, etc.), para calcular diferentes estadísticos (frecuencia alélica, cobertura, etc.), para convertir entre distintos tipos de archivos, para concatenar o fusionar varios archivos, etc.

PGDSpider v.2.1.1.5

PGDSpider (PGDSPIDER, 2019) es un *software* de conversión entre diferentes formatos de archivo de datos genómicos. Es capaz de manejar diferentes tipos de datos (NGS, microsatélites, SNP, RFLP, frecuencias, etc.) y es capaz de leer 33 y escribir en 36 formatos de archivo diferentes (VCF, BAM, FASTA, MAF, etc.).

Este programa está escrito en Java y presenta una interfaz de usuario fácil de utilizar. Además, también puede utilizarse en la línea de comandos, por lo que puede integrarse como un paso en *pipelines* complejas.

CurlyWhirly v.1.19.03.26

CurlyWhirly (CURLYWHIRLY, 2019) es un *software* diseñado para la visualización de datos en 3 dimensiones, en concreto, está encaminado a la visualización de datos provenientes de un Análisis de Componentes Principales (PCA, *Principal Component Analysis*) o de un Escalado Multidimensional (MDS, *Multidimensional Scaling*).

Este programa es capaz de mostrar miles de puntos. Además se pueden especificar diferentes categorías y añadir etiquetas a los puntos en cada una de ellas, permitiendo un filtrado y una visualización de los datos sencilla. Por ejemplo, se puede crear la categoría "altura" y la categoría "país" y poner una etiqueta a los datos para cada una de ellas.

3.2 Métodos

3.2.1 DiploSHIC

Este programa puede descargarse e instalarse siguiendo las instrucciones disponibles en su repositorio en GitHub (GITHUB, 2019). En este repositorio también se encuentra una descripción detallada de los pasos a seguir para llevar a cabo el análisis, así como las instrucciones de uso de cada uno de los modos en la línea de comandos. El esquema de trabajo se puede resumir en:

1. Generación de datos simulados, mediante un programa externo. Esto puede llevarse a cabo con Discoal (https://github.com/kern-lab/discoal), desarrollado por los mismos investigadores que diploSHIC.
2. Cálculo de los vectores de características para los datos simulados, mediante el modo `fvecSim` del diploSHIC.
3. Crear los grupos de entrenamiento (*training sets*), es decir, agrupar los datos simulados en 5 archivos diferentes, uno para cada tipo de barrido y uno para datos neutrales, mediante el modo `makeTrainingSets`.

4. Entrenar el programa clasificador con los 5 sets de datos creados anteriormente, mediante el modo `train`.

Una vez el programa de aprendizaje automático ya ha sido entrenado, se puede proceder a analizar los datos experimentales.

5. Cálculo de los vectores de características para los datos experimentales, mediante el modo `fvecVcf`.
6. Clasificación de los datos empíricos, mediante el modo `predict`.

Para la generación mediante Discoal (o cualquier otro programa) de datos simulados para cada tipo de barrido es necesario tener una idea *a priori* de diferentes características de la población y del valor de diferentes parámetros, como por ejemplo θ (tasa de mutación poblacional), N_0 (población efectiva), α (intensidad de la selección), τ (tiempo desde que se fijó el alelo seleccionado), etc. Como en el caso que nos ocupa no se han hecho estimas de estos parámetros, se ha decidido utilizar únicamente el modo `fvecVcf`, el cual calcula los vectores de características (*feature vectors*). De esta forma, se obtiene el valor para 12 estadísticos diferentes para todas las regiones del genoma.

Para correr el modo `fvecVcf` se especifican los siguientes parámetros:

- shicMode: especifica si se quiere correr el programa para organismos haploides o diploides. En este caso hay que especificar 'diploid'.
- chrArmvcffile: ruta al fichero VCF con los datos para un cromosoma en concreto.
- chrarm: nombre del cromosoma.
- chrlen: longitud del cromosoma.
- fvecfilename: ruta al fichero donde se escribirán los resultados.
- --targetPop (opcional): nombre de la población o del set de muestras para las que se quiere calcular los vectores de características.
- --sampleToPopFileName (opcional): ruta al fichero donde se especifica a qué población o set pertenece cada una de las muestras.

3.2.2 SweeD

El programa puede descargarse a través de su repositorio en GitHub (GITHUB, 2019). Aquí también se puede encontrar un manual (`sweed3.0_manual.pdf`) con las instrucciones para compilar e instalar el programa, así como sus instrucciones de uso.

Para ejecutar el programa, se especifican los siguientes parámetros:

- SweeD-P: para utilizar la versión de SweeD que se ejecuta en paralelo.
- -name: nombre para los archivos que se van a generar. En este caso será el nombre de la población.
- -input: ruta a un archivo VCF. En este caso el VCF contiene datos para una población en concreto.
- -grid: tamaño de la ventana para la que se calculará el CLR, en pares de bases. Basándose en los ejemplos proporcionados en el manual, se utiliza un tamaño de 100.
- -folded (opcional): para indicar que el alelo ancestral y el derivado no pueden distinguirse.

3.2.3 BayeScan

El programa BayeScan puede descargarse a través de su página web (BAYESCAN, 2019). Tanto las instrucciones para su instalación y compilación como para su uso se encuentran en el manual disponible en la página web (en el apartado download > manual) y en el archivo `BayeScan2.1_manual.pdf` disponible junto con los demás archivos descargados.

Como este programa realiza los cálculos sobre un archivo con los datos para todas las poblaciones, sólo es necesario ejecutar el programa una vez. Esto puede hacerse directamente en la línea de comandos ejecutando la siguiente orden:

```
$ bayescan_2.1 all_pops.txt
```

Siendo `all_pops.txt` en archivo en formato GESTE de todas las poblaciones.

3.2.4 LOSITAN

LOSITAN es una aplicación de Java que se utiliza directamente desde una plataforma web, aunque todos los cálculos se realizan en el ordenador del usuario y no en el servidor. Sin embargo, al intentar acceder a la página web aparece un mensaje de error indicando que el servidor no ha sido encontrado. Esto es debido a que LOSITAN es un proyecto que empezó en 2008 y actualmente está abandonado.

Sin embargo, debido a su utilidad hay usuarios que han recogido los programas necesarios para instalar y poder ejecutar LOSITAN de forma local en el propio ordenador (sin necesidad de acceder a la aplicación web) como es el caso de Tiago Antao (GITHUB, 2019) y, más recientemente, Baskar Lingam (GITHUB, 2019). En el repositorio de Baskar Lingam, en la carpeta `OfflineVersion`, en el archivo `InstallationStepsTaken` se indica el orden en el que se tienen que ejecutar los programas para poder instalar el LOSITAN, teniendo en cuenta que los programas con la extensión '.sh' deben ejecutarse escribiendo el comando `sh` en la terminal y los programas con la extensión '.bat' con el comando `wineconsole` (para ello es necesario instalar previamente Wine, un *software* para ejecutar en Linux programas escritos para Windows).

3.2.5 RaisD

Este programa puede descargarse a través de su repositorio en GitHub (GITHUB, 2019), donde también se pueden encontrar las instrucciones para compilarlo. Para ejecutar RaisD es necesario especificar los siguientes parámetros:

- -n : proporciona un ID a la ejecución. Este ID se utiliza para nombrar a los archivos generados. En este caso, se corresponde con el nombre de la población que está siendo analizada.
- -I : especifica directorio del VCF que va a ser analizado.
- -S : especifica el directorio a un archivo con la lista de muestras que quieren analizarse. En este caso, se analizarán las muestras pertenecientes a una población.

- -P : genera un archivo PDF para cada uno de los cromosomas de la población, en el que se dibujan cuatro figuras: una para cada uno de los tres factores del estadístico μ (SFS, LD y diversidad) y el valor final de μ.
- -f : reescribe los archivos con el mismo ID, en el caso de que los haya.

4. Resultados y discusión

4.1 DiploSHIC

4.1.1 Obtención y procesamiento de los resultados

En cada ejecución DiploSHIC únicamente puede calcular los estadísticos para un cromosoma de una población. El tomate tiene 12 cromosomas y hay un total de 29 poblaciones, por lo que el número de ejecuciones es muy alto. Para automatizar este proceso, se ha escrito un programa en Python llamado `calculate_statistics.py` (Apéndice I) el cual ejecuta sucesivamente el DiploSHIC para cada cada cromosoma de cada población. Además, como es necesario especificar la longitud del cromosoma, este programa incorpora una función para calcularla a partir de un archivo bed. Por otra parte, también es necesario proporcionarle un archivo de texto que especifique la población a la que pertenece cada una de las muestras. Este archivo se ha creado mediante el programa `do_sample_to_pop_file.py` (Apéndice II), que toma la información de un archivo csv donde se indica la clasificación de las muestras llevada a cabo por el Doctor José Blanca.

Una vez ejecutado el programa `calculate_statistics.py`, se obtiene un archivo de texto para cada cromosoma de cada población con los resultados. Todos los archivos están disponibles en el Anexo I. En estos archivos, en cada una de las líneas (o filas de una hipotética tabla) se guarda información para una ventana del cromosoma de $11 \cdot 10^5$ pb. Esta ventana, a su vez, se divide en once subventanas más pequeñas de $1 \cdot 10^5$ pb. Para cada una de estas subventanas, se calculan 12 estadísticos, por lo que hay un total de 11 x 12 = 132 datos (o columnas) para cada una de las regiones. Se puede ver un esquema de la estructura de datos obtenida en la Tabla 1.

Tabla 1. Representación esquemática de la estructura de datos obtenida utilizando el modo `fvecVcf` del programa DiploSHIC. Cada una de las filas se corresponde con una región (o ventana) de 1100000 pb del cromosoma. Cada una de estas ventanas se divide a su vez en 11 subventanas de 100000 pb, para las cuales se calculan los 12 estadísticos. En esta tabla únicamente se representa el estadístico pi para facilitar la comprensión de la estructura de datos obtenida.

chr	Big Win Range	pi win0	pi win1	pi win2	pi win3	pi win4	pi win5	pi win6	pi win7	pi win8	pi win9	pi win10
01	87400001-88500000	0,04	0,03	0,01	0,05	0,04	0,08	0,1	0,1	0,3	0,009	0,03
01	87500001-88600000	0,02	0,01	0,03	0,03	0,06	0,1	0,1	0,2	0,006	0,02	0,3

La diferencia en pares de bases entre una ventana y la siguiente es de $1 \cdot 10^5$ pb, es decir, no son consecutivas sino que solapan entre ellas (Figura 1). Por tanto, para cada una de las subventanas en las que está dividida una región, se calculan todos los estadísticos un total de 11 veces, es decir, hay varios datos para cada subventana. Debido a esto y a la

estructura de datos tan compleja, se ha escrito el programa `reorganize_and_plot_results.py` (Apéndice III). Este programa recoge todos los valores para cada subventana y calcula la media. De esta forma, en cada subventana, que consiste en una región de $1 \cdot 10^5$ pb, se obtiene un único valor para cada uno de los estadísticos.

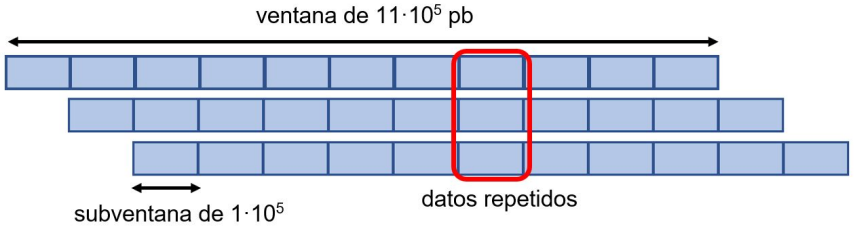


Figura 1. Representación esquemática de la división en ventanas y subventanas llevado a cabo por el programa DiploSHIC para llevar a cabo el cálculo de los estadísticos.

Además, como la cantidad de datos generados es muy grande estos se representan en un gráfico para facilitar el análisis. La generación de estos gráficos está implementada en una función del programa `reorganize_and_plot_results.py` (Apéndice III). Esta función representa, para cada cromosoma de cada población, 12 gráficos diferentes (uno para cada uno de los estadísticos), en el cual en el eje de abscisas se encuentra la región cromosómica y en el eje de ordenadas el valor del dicho estadístico. Esto se lleva a cabo utilizando las herramientas disponibles en la librería `matplotlib`. Todas las gráficas están disponibles en el Anexo II.

En la mayoría de los casos, no se han obtenido resultados para la totalidad del cromosoma, sino únicamente para una o unas pocas regiones. Cuando estas regiones están muy separadas entre sí, los resultados no se pueden observar de forma clara (Figura 2a), por lo que es necesario truncar la gráfica, es decir, eliminar de la gráfica las regiones para las que no se obtienen datos (Figura 2b). Esto está implementado dentro del programa `reorganize_and_plot_results.py`, no es necesario modificar las gráficas *a posteriori* ya que éste genera tanto la gráfica original como la truncada.

a)

b)

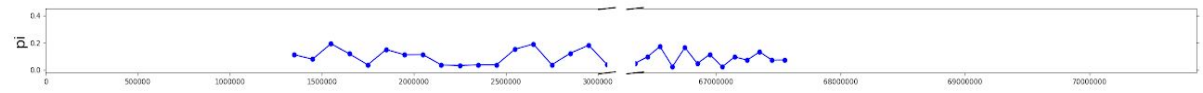


Figura 2. Valor de uno de los estadísticos (eje de ordenadas) a lo largo de diferentes regiones cromosómicas (eje de abscisas). **a)** Representación del valor del estadístico pi a lo largo del cromosoma. **b)** Representación del valor del estadístico pi a lo largo de diferentes regiones cromosómicas, eliminando las regiones que no presentan ningún valor del estadístico pi.

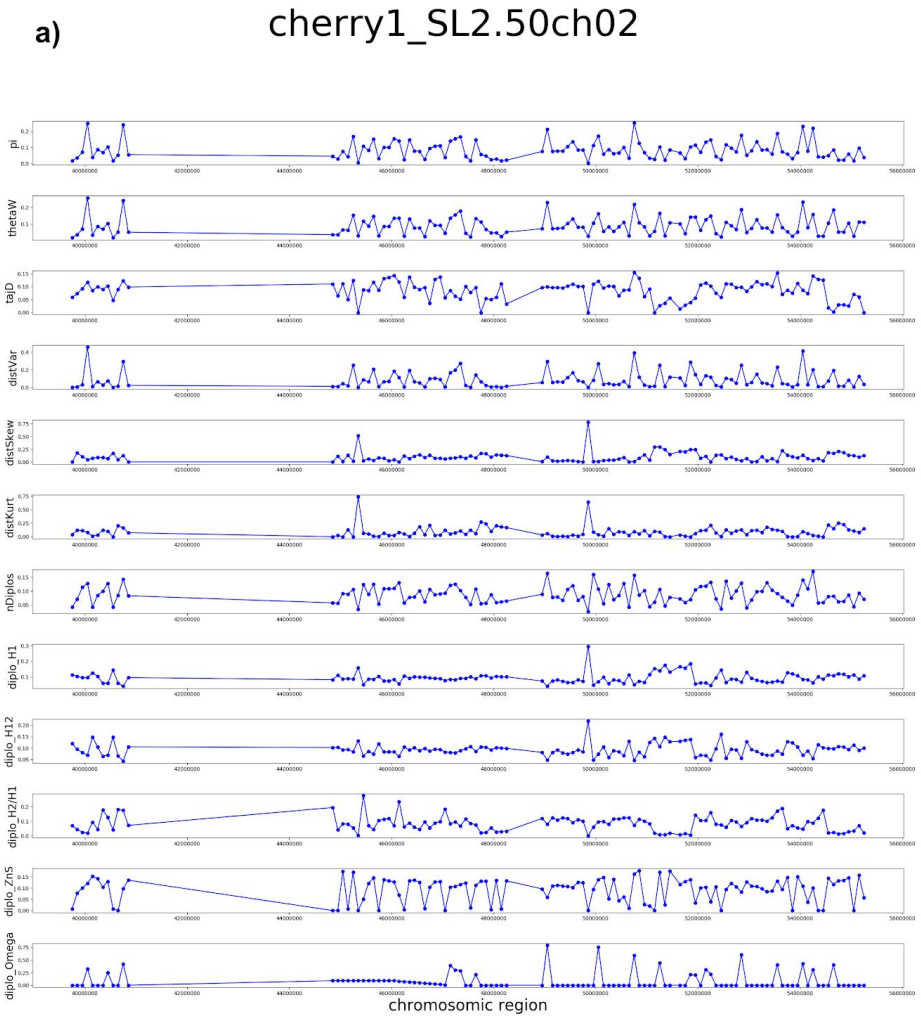
4.1.2 Análisis de las gráficas generadas

DiploSHIC recorre cada uno de los cromosomas de cada una de las poblaciones en ventanas de 1100000 de pares de bases. Para cada una de ellas, calcula un total de 12

estadísticos que miden diferentes marcas que la selección positiva deja en el genoma. Así pues, siguiendo el procedimiento descrito en el apartado anterior, para cada cromosoma de cada población se obtiene un total de 12 gráficos con los valores de cada uno de los estadísticos a lo largo de las regiones cromosómicas, tal como se muestra en la Figura 3a.

Las distintas regiones presentan patrones muy variados para los parámetros calculados por el DiploSHIC. Esto hace que sea muy difícil determinar las regiones que pueden haber sufrido un proceso de selección, ya que no se encuentra ningún pico o valle que se distinga claramente del resto, incluso en las gráficas con poca densidad de datos (Figura 3b). En la Figura 3 se muestran, como ejemplo representativo, las gráficas obtenidas para dos cromosomas de la población cherry1, pero este comportamiento también se cumple para todo el resto de poblaciones.

a)



b) cherry1_SL2.50ch03

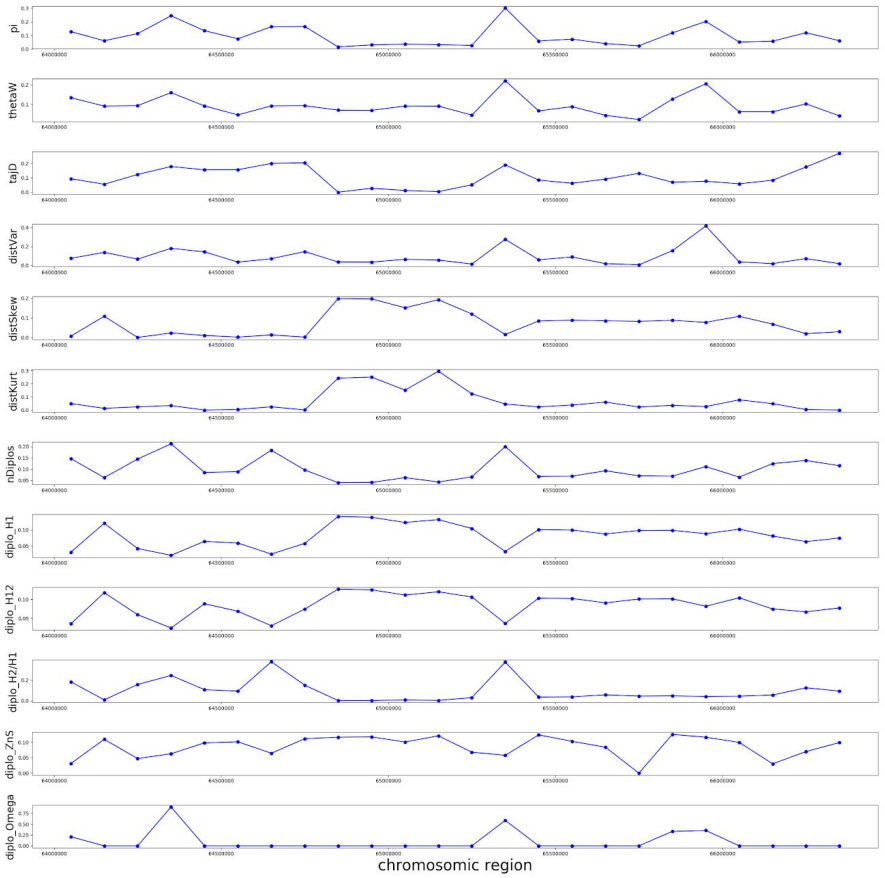


Figura 3. Representación del valor de cada uno de los estadísticos calculados por el programa diploSHIC, en diferentes regiones de un cromosoma de una población. **a)** representación de dichos valores para el cromosoma 2 de la población cherry1. **b)** representación de dichos valores para el cromosoma 3 de la población cherry1, con muy poca densidad de datos. Las gráficas para el resto de cromosomas y poblaciones se encuentran en el Anexo II.

4.1.3. Análisis de las regiones con un bajo número de SNPs

En algunos casos el número las regiones para las que se obtienen resultados es reducido si se compara con el número total de regiones en un cromosoma. Esto puede observarse en la Figura 3b para un cromosoma en una población, aunque sucede algo similar para otros cromosomas de otras poblaciones (Anexo II). Esto puede ser debido a que en algunas regiones no hay SNPs suficientes para llevar a cabo el cálculo de los estadísticos.

Es común que los programas que analizan este tipo de datos genéticos realicen filtrados para quedarse con SNPs que cumplen ciertas características. Para comprobar si este era el caso del DiploSHIC, se examinó el programa `makeFeatureVecsForChrArmFromVcfDiploid.py` en búsqueda de filtros que pudieran estar eliminando determinados sitios. Este programa es el encargado de calcular los vectores de características a partir del VCF y está disponible en el repositorio de DiploSHIC en GitHub (GITHUB, 2019). Al examinarlo se observó que en este programa los únicos filtros que se aplican son: que la proporción de sitios no enmascarados sea mayor

que un umbral (0.75 por defecto), que la proporción de datos faltantes sea menor que un umbral (0.25 por defecto) y que el SNP sea bialélico.

Estos filtros se pueden reproducir con VCFtools (VCFTOOLS, 2019), un paquete con diferentes herramientas para trabajar con archivos de tipo VCF. Las opciones utilizadas son:

- --min-alleles. Mantiene únicamente los sitios que tienen un número de alelos igual o mayor al número especificado. Para filtrar los sitios bialélicos, este número debe ser 2.
- --max-alleles. Mantiene únicamente los sitios que tienen un número de alelos menor o igual al número especificado. Para filtrar los sitios bialélicos, este número debe ser 2.
- --max-missing. Excluye sitios basándose en la proporción de datos faltantes: 0 significa que se permiten sitios que estén formados completamente por datos faltantes y 1 significa que no permite sitios que tengan algún dato faltante. Para aplicar el mismo filtro que en diploSHIC, este número debe ser 0.75.
- --kept-sites. Genera una archivo con una lista de todos los sitios que se han conservado después de aplicar los filtros.

De esta forma, se generan dos archivos: uno con el sufijo '.log' y otro con el sufijo '.kept.sites'. Todos los archivos se encuentran en el Anexo III. En los archivos con el sufijo '.log' hay una línea que especifica el número de sitios antes y después de aplicar los filtros. Esta información se utiliza para construir un gráfico de barras para cada población en la que, para cada cromosoma, se representa el número de SNPs totales antes y después de filtrar. Todas las gráficas están disponibles en el Anexo IV. Los archivos con el sufijo '.kept.sites' contienen los sitios que se han conservado después de aplicar los filtros. Estos ficheros constan de dos columnas: una con el cromosoma y otra con el sitio conservado en pares de bases, separadas por un tabulador. Esta información se utiliza para dibujar una gráfica por cada cromosoma de cada población, en las que se representan las posiciones cromosómicas en el eje de abscisas y en el eje de ordenadas un valor arbitrario, el mismo para todos los sitios. Esta representación se hace tanto de los sitio del VCF sin filtrar y después del filtrado. De esta forma, se puede observar de una forma mucho más clara en qué regiones cromosómicas se mantienen los SNPs después del filtrado. Todas las gráficas están disponibles en el Anexo V.

Este análisis se lleva a cabo mediante el programa `low_n_snps_test.py` (Apéndice IV). Este programa presenta una función para llamar al VCFtools, aplicar los filtros descritos y generar los archivos '.log' y '.kept.sites'. A continuación, presenta una función para leer estos archivos '.log' y guardar la información necesaria en una matriz `pandas` y otra diferente para, a partir de esta matriz, construir las gráficas mencionadas utilizando la librería `seaborn`. Por otro lado, también presenta sendas funciones para leer los archivos '.kept.sites', guardar la información y generar las gráficas mencionadas utilizando la librería `matplotlib`.

Este programa trabaja con VCFs que contienen datos genómicos de un único cromosoma de una determinada población, puesto que el DiploSHIC genera los resultados de esta forma. Por tanto, como paso previo a la aplicación de los filtros es necesario separar el VCF original en VCFs más pequeños, uno para cada cromosoma de cada población. Este tipo de filtrado no está disponible en VCFtools, pero sí en BCFtools (BCFTOOLS, 2019). Este paquete de programas presenta un módulo de llamado `view` con diferentes opciones para filtrar y generar subconjuntos de un VCF dado. En este caso, las opciones utilizadas son:

- -s, --samples. Incluye únicamente los datos para las muestras especificadas, que en este caso serán las pertenecientes una población en concreto.
- -t, --targets. Descarta las posiciones no especificadas en este argumento. Si se especifica el nombre de un cromosoma, se descartan todos los sitios que no estén en ese cromosoma.

Para poder generar todos los VCFs de forma automática, se utiliza el programa `do_vcf_per_population_per_chromosome.py` (Apéndice V). Este programa, presenta una función para generar una lista de las muestras pertenecientes a cada una de las poblaciones, a partir del archivo csv con la clasificación de las muestras llevada a cabo por el Doctor José Blanca. Además, contiene una función que toma las listas generadas y el VCF original y ejecuta el BCFtools con las opciones de filtrado para cada una de las poblaciones y cromosomas de forma consecutiva.

Así pues, siguiendo el procedimiento descrito, se generan dos tipos de gráficas: una que indica el número de SNPs antes y después del filtrado (Figura 4) para cada cromosoma y otra que indica la distribución de los SNPs a lo largo del cromosoma (Figura 5). Como ejemplo representativo se muestran gráficas para la población cherry1.

Como puede observarse en la Figura 4, el número de SNPs después de aplicar los filtros se reduce significativamente. Para ver si esta reducción coincide con las regiones ignoradas por el DiploSHIC se comparan las Figura 5a y Figura 5b. En estas gráficas, cada punto representa un SNP en una posición en el cromosoma, indicada en el eje de abscisas. En la Figura 5a se encuentran los SNPs en el VCF original, sin filtrar, mientras que en la Figura 5b los SNPs que se han mantenido después de aplicar los filtros. En ambas figuras los SNPs están repartidos en las mismas zonas del cromosoma, no hay una reducción significativa en ninguna zona en concreto. Además, estos SNPs se concentran en los extremos del cromosoma. Este patrón es característico del tomate, que presenta una zona pericentromérica muy grande y la mayor densidad de genes se encuentra en estas regiones a los extremos (The Tomato Genome Consortium, 2012).

Aún después de este filtrado, el número de sitios que se conservan es mucho mayor que el número de sitios para los cuales se obtienen datos de los estadísticos calculados por DiploSHIC. Además, en algunos casos las regiones para las que se obtienen datos son muy pequeñas, mientras que se ha visto que incluso después de aplicar los filtros los SNPs conservados se distribuyen por todo el cromosoma (exceptuando la región pericentromérica). Por tanto, además de en este filtrado inicial debe haber otro punto en el DiploSHIC donde descartan un gran número de SNPs o de regiones cromosómicas y que no se ve reflejado en el *script* que calcula los vectores de características del DiploSHIC. Probablemente las regiones para las que no se obtienen datos sean aquéllas que tienen una densidad de SNPs muy baja, hecho bastante común en tomate, de forma que no haya suficientes datos para calcular los estadísticos.

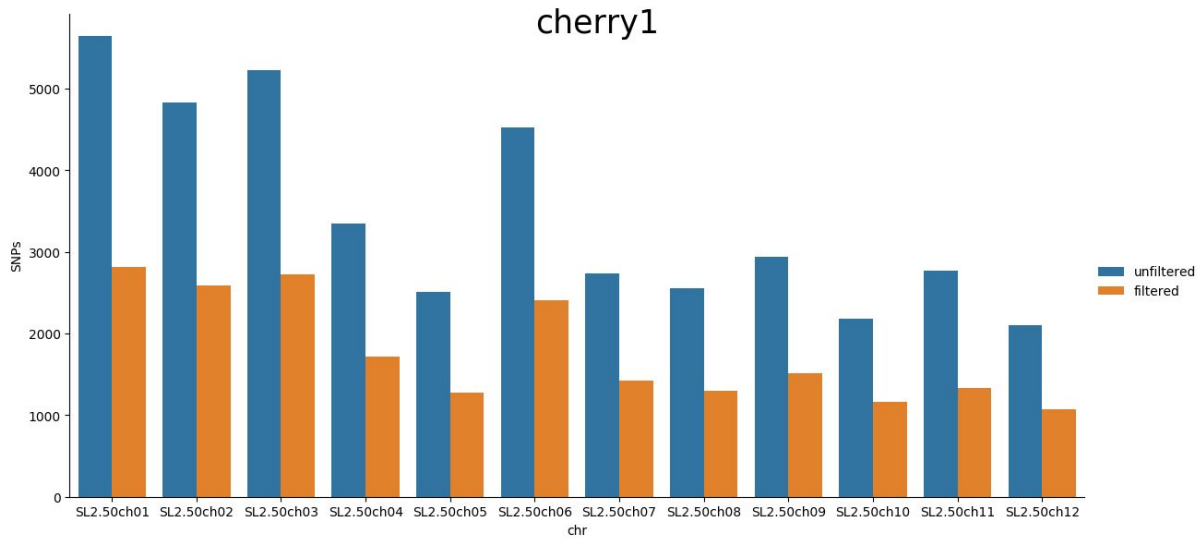


Figura 4. Número de SNPs para cada uno de los cromosomas de la población cherry1, antes y después del filtrado. El filtrado se lleva a cabo con el VCFtools y se conservan los sitios que sean bialélicos y presenten un ratio de sitios enmascarados y de datos faltantes menor a 0.25, mimetizando los filtros llevados a cabo en el *script* que calcula los vectores de características (`makeFeatureVecsForChrArmFromVcfDiploid.py`) en el programa DiploSHIC.

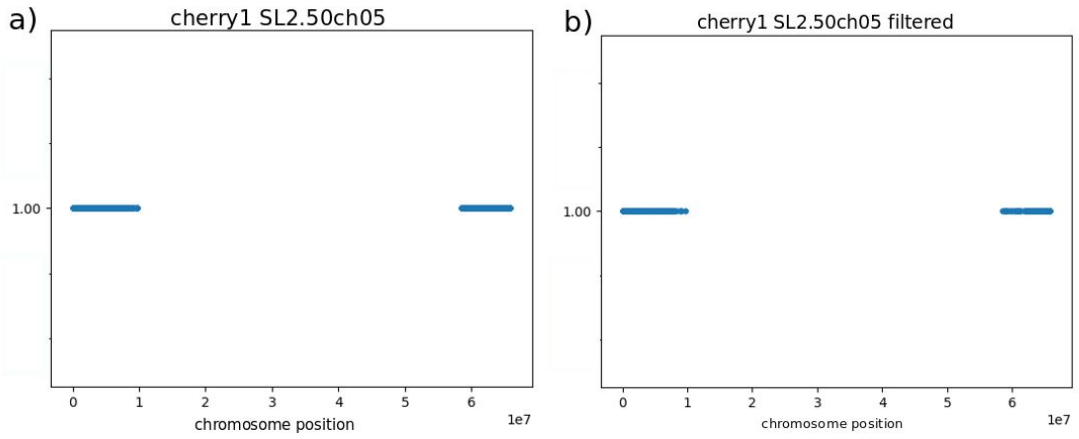


Figura 5. Representación de la cobertura en SNPs a lo largo del cromosoma 05 de la población cherry1. Cada punto representa un sitio en el cromosoma (eje de abscisas) en el cual se encuentra un SNP, al que se le ha dado un valor arbitrario en el eje de ordenadas (1 en este caso) para poder representarlo en el gráfico y observar de forma clara las regiones cromosómicas para las cuales hay SNPs. **a)** cobertura en SNPs antes de filtrar. **b)** cobertura en SNPs después de haber filtrado con el VCFtools los sitios bialélicos y que presenten un ratio de sitios enmascarados y con datos faltantes menor a 0.25.

4.1.4 Determinación de regiones genómicas con comportamientos diferenciados

Como se ha podido observar en el apartado anterior, a partir de todas las gráficas generadas es difícil clasificar las regiones que tienen comportamientos diferenciados de acuerdo con los parámetros calculados por el DiploSHIC. Por ello, se decidió llevar a cabo un análisis de componentes principales (PCA, *Principal Component Analysis*) para agrupar

las distintas regiones genómicas según los valores obtenidos para los estadísticos calculados por el DiploSHIC.

Se realizó un PCA para cada población, tomando los valores de los estadísticos como variables. El programa `do_pca_per_pop.py` (Apéndice VI) contiene todos los pasos necesarios para llevar a cabo el análisis, organizados en funciones:

- Organización los datos en una matriz, una por cada población. Cada una de las filas de la matriz se corresponde a una región cromosómica diferente y en cada una de las columnas se encuentra el valor en esa región para cada uno de los estadísticos. Además, también se genera una lista con los identificadores para cada una de las regiones, que se componen por el nombre del cromosoma más la posición en ese cromosoma en pares de bases. El número de elementos de esta lista es el mismo que el número de filas en la matriz, además de conservar el mismo orden. Esta matriz se genera con NumPy (NUMPY, 2019), un paquete de herramientas para trabajar con matrices multidimensionales.
- Estandarización de la matriz. Cada uno de los estadísticos puede tener unidades de medida y órdenes de magnitud diferentes, por lo que una misma cantidad de variación entre dos datos puede ser significativa o no dependiendo de cada caso. Es muy importante en este tipo de análisis estandarizar los datos para que todas las variables sigan una distribución con una media centrada en 0 y una desviación estándar de 1 (SCIKIT-LEARN, 2019). La estandarización se lleva a cabo con la función `StandardScaler` disponible en el módulo `preprocessing` del Scikit-learn (SCIKIT-LEARN, 2019), un paquete de herramientas para el procesamiento y análisis de datos.
- Análisis de componentes principales. Una vez los datos están organizados en una matriz y estandarizados, se realiza el análisis con la función `PCA` del módulo `decomposition` del Scikit-learn. Esta función permite escoger el número de componentes principales, 3 en este análisis, y devuelve las coordenadas de cada una de las regiones en cada uno de los componentes. Además, también presenta el atributo `components`, el cual devuelve los valores propios de cada uno de los componentes principales y el atributo `explained_variance_ratio`, el cual devuelve el porcentaje de variación explicado por cada uno de los componentes principales.
- Generación de un informe del PCA. En este archivo se escriben los valores propios junto con el nombre de la variable a la que pertenecen, además de las variables más representativas de cada uno de los componentes principales. También se escribe el porcentaje de variación explicado por cada uno de los componentes principales.
- Generación de un archivo para visualizar los datos en CurlyWhirly (CURLYWHIRLY, 2019). Este programa de visualización de datos en 3 dimensiones requiere un formato de fichero propio. En este caso, en cada línea se especifica la información para una región cromosómica, una etiqueta que identifica a ésta región, una o varias categorías a las que pertenece (opcional) y las coordenadas en cada uno de los componentes principales.
- Representación de los valores propios en gráficas de 2 dimensiones; gráfico de pesos (*loading plot*). En el eje de abscisas se representa el valor de cada valor propio en una componente principal y en el eje de ordenadas esos mismos valores para otra componente principal, de forma que se generan un total de tres gráficas para cada población: PC2 vs PC1, PC1 vs PC3 y PC3 vs PC2. El objetivo es poder visualizar de forma clara las variables que tienen más peso en cada una de las componentes tanto en la parte positiva como negativa del eje.

Todos los archivos y plots generados se encuentran en el Anexo VI.

De esta forma, en los PCAs generados cada punto representa una región genómica. Lo esperable es que la mayoría de las regiones sean neutrales y que tengan un comportamiento similar, es decir, que tengan un valor similar para todos los estadísticos y por tanto que se agrupen en el PCA. Así pues, el objetivo de este análisis es comprobar si hay regiones que tengan comportamientos diferentes al grueso de regiones, ya que es posible que algunas de estas regiones hayan sufrido algún tipo de selección.

En la Figura 6 se encuentran diferentes proyecciones del PCA para la población cherry1. La Figura 6a es una visión global del PCA, donde se pueden ver las 3 dimensiones. Los puntos, que se corresponden con regiones genómicas, están coloreados según el cromosoma al que pertenecen. En la Figura 6b los ejes están rotados de forma que únicamente se observan los ejes Y (PC2) y X (PC1). En esta proyección se identifican 5 regiones (puntos) que se alejan bastante del cúmulo principal, las cuales se han rodeado con un círculo en la imagen. Se puede averiguar qué estadístico ha hecho que estas regiones se diferencien del resto observando la composición de los componentes principales en el gráfico de pesos (Figura 7). De esta forma, se observa que las regiones 45350000 y 49850000 del cromosoma 02 y 62050000 del cromosoma 04 (rodeadas con un círculo de color rojo en la Figura 6b) están alejadas porque presentan valores atípicos de los estadísticos diplo_H1 y diplo_H12. Sabemos que esto es así porque en el gráfico de pesos (Figura 7a) los valores propios de estos estadísticos apuntan a la misma zona del PCA en la que se encuentran estos puntos (Figura 6b). Siguiendo el mismo procedimiento, se obtiene que la región rodeada con un círculo de color rosa (46050000 del cromosoma 06) está separada debido a los estadísticos diplo_Omega, thetaW y distVar y la región rodeada por el círculo de color azul (92050000 del cromosoma 01) por los estadísticos tajD y diplo_H1/H2.

La siguiente perspectiva a observar es X (PC1) vs Z (PC3) (Figura 6c). Los puntos identificados en la anterior perspectiva se deseleccionan para que aparezcan en gris y, así, no volver a estudiarlos. Ahora pueden identificarse 3 nuevas regiones, todas en la misma zona del gráfico (rodeadas con un círculo en la figura): 42950000 del cromosoma 6, 150000 del cromosoma 7 y 1750000 del cromosoma 12. En la Figura 7b, se observa que estos puntos están separados debido a valores atípicos de distSkew.

Por último, en la perspectiva Z (PC3) vs Y (PC2) (Figura 6d) se identifican 6 nuevas regiones: las pertenecientes a los puntos rodeados con un círculo de color rojo (66050000 del cromosoma 4, 45750000 del cromosoma 6 y 950000 del cromosoma 7) que presentan valores atípicos del estadístico diplo_H2/H1 (Figura 7c), los rodeados con un círculo de color amarillo (42550000 y 43850000 del cromosoma 6) que presentan valores atípicos del estadístico tajD y, por último, el punto rodeado por un círculo de color azul (51850000 del cromosoma 12) con un valor atípico de los estadísticos diplo_H12 y diplo_ZnS.

Siguiendo este procedimiento para el resto de poblaciones, se anotan en una tabla (Apéndice VII) todas las regiones que potencialmente han sufrido una evolución distinta al grueso del genoma. En esta tabla se recogen los datos de población, cromosoma, región cromosómica y estadístico o estadísticos para los cuales presenta un valor atípico.

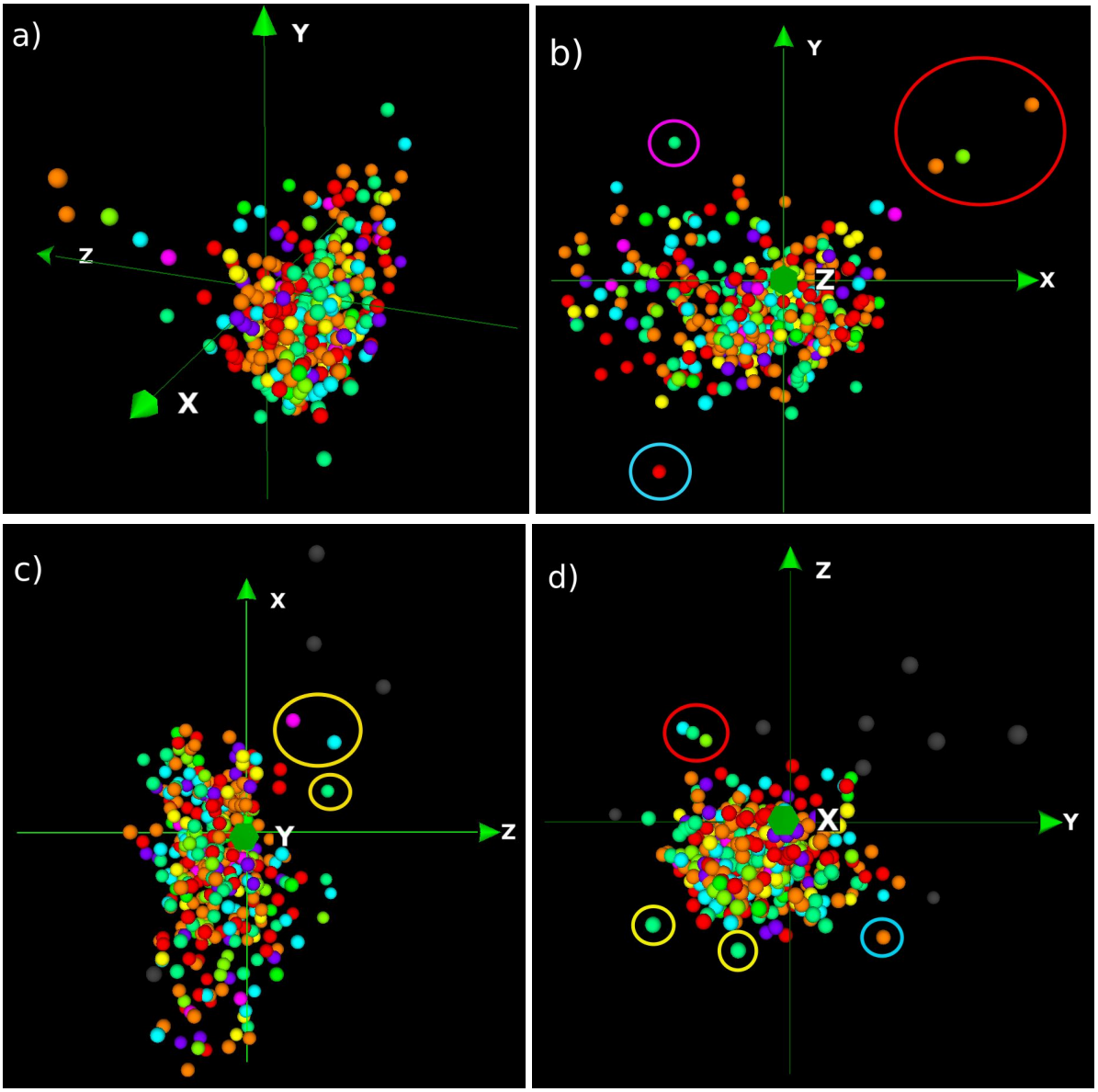


Figura 6. PCA para la población cherry1, obtenido tomando como variables los valores de los 12 estadísticos calculados a lo largo de diferentes regiones cromosómicas de la población. Cada uno de los puntos se corresponde con una región cromosómica en concreto y están coloreados según al cromosoma al que pertenecen. El eje X se corresponde con la componente principal 1 (PC1), el eje Y con la componente principal 2 (PC2) y el eje Z con la componente principal 3 (PC3). **a)** visión general de la distribución de puntos en las tres dimensiones del PCA. En la leyenda se especifica a qué cromosoma corresponde cada uno de los colores. **b)** proyección del PCA en la cual únicamente se observa la variabilidad para el PC2 vs PC1. **c)** proyección del PCA en la cual únicamente se observa la variabilidad para el PC1 vs PC3, con los puntos identificados como atípicos en la anterior proyección deseleccionados, por lo que aparecen de color gris. **d)** proyección del PCA en la cual únicamente se observa la variabilidad para el PC3 vs PC2, con los puntos identificados como atípicos en las anteriores proyecciones deseleccionados, por lo que aparecen de color gris.

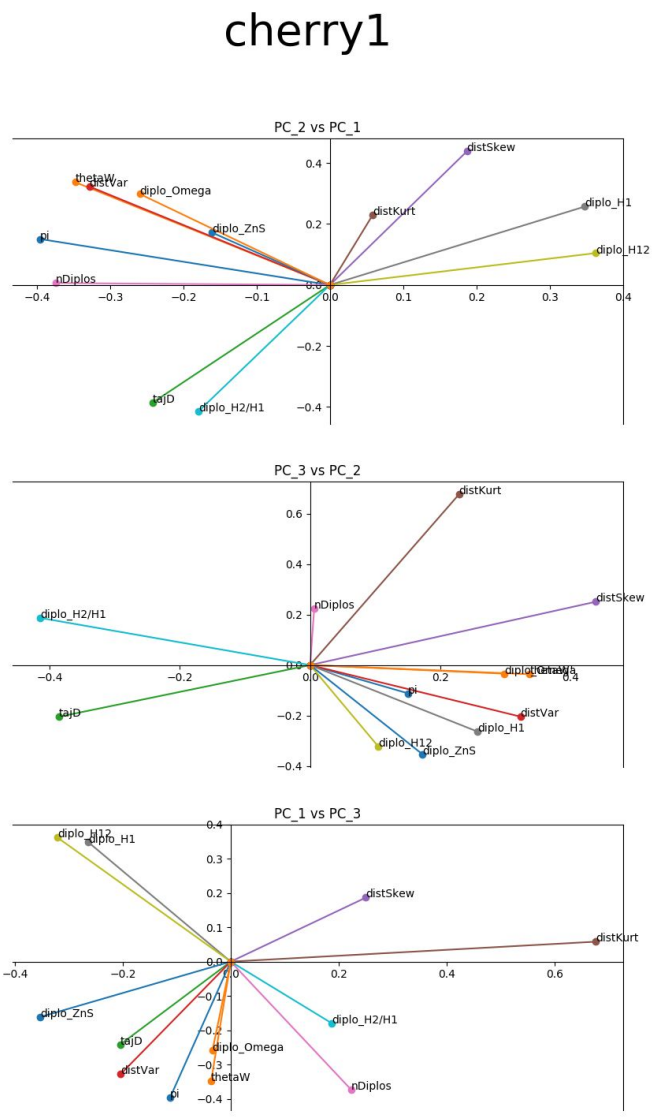


Figura 7. Gráfico de cargas. Se representan los valores propios de cada uno de los estadísticos en una componente principal frente a otra componente. Esto da un punto en el gráfico, que se une con el origen de coordenadas para que sea más visual. **a)** valores propios de la componente principal 2 frente la componente principal 1. **b)** valores propios de la componente principal 3 frente la componente principal 2. **c)** valores propios de la componente principal 1 frente la componente principal 3.

Los estadísticos diplo_H1 y diplo_H12 suelen estar correlacionados en casi todas las poblaciones. En los gráficos de cargas suelen aparecer juntos apuntando en la misma dirección y sentido. Otro grupo de estadísticos bastante correlacionados es el formado por thetaW, distVar y diplo_Omega, aunque en ocasiones hay alguno de ellos que apunta en una dirección un poco diferente. El estadístico diplo_ZnS tiene un comportamiento más dispar y suele encontrarse solo, aunque en ocasiones aparece junto con el grupo anterior. Por otra parte, se encuentran los grupos formados por pi - nDiplos y tajD - diplo_H2/H1, que si bien casi siempre se encuentran en el mismo cuadrante, suelen tener pendientes un tanto distintas, por lo que se puede decir que están correlacionados pero no en la medida que lo están diplo_H1 y diplo_H12. Por último, el grupo formado por distKurt y distSkew tiene un comportamiento muy peculiar, debido a que en algunos casos aparecen totalmente

correlacionados (apuntando en la misma dirección y sentido), mientras que en otros aparecen en la misma dirección pero sentido contrario, es decir tienen una correlación negativa.

Tanto diplo_H1 como diplo_H12 son estadísticos que miden el nivel de homocigosidad del haplotipo. Diplo_H1 es la frecuencia del haplotipo más común y diplo_H12 es la suma de la frecuencia del primer y segundo haplotipos más frecuentes. Por tanto es esperable que estén correlacionados. Estos estadísticos tratan de detectar tanto barridos selectivos fuertes como suaves (en los que se ha seleccionado más de un haplotipo). Si se da un evento de selección positiva, lo esperable es que un haplotipo (o en algunos casos, más de uno) sea el seleccionado lo cual hará aumentar su frecuencia en la población y el número de homocigotos.

En cuanto al grupo thetaW, distVar, y diplo_Omega; thetaW es una medida de la diversidad genética, que se obtiene contando el número de sitios polimórficos dividido por un factor de corrección, distVar es la varianza de la distribución de las medidas de distancia entre todas las secuencias de una población obtenidas dos a dos (calculada como número de diferencias nucleotídicas) y diplo_Omega es una medida que tiene en cuenta la distribución espacial del desequilibrio de ligamiento en una región que ha sido seleccionada. Cuando una región sufre un proceso de selección positiva o cuando una población sufre un cuello de botella el número de polimorfismos tiende a reducirse. En el caso de la selección a favor de un alelo tanto ese locus como los que están ligados aumentan su grado de fijación. Al reducirse la diversidad genética, la distancia genética entre las secuencias de los individuos de la misma población también tenderá a reducirse, ya que la mayoría de ellas compartirán los mismos alelos y, por tanto, la varianza de la distribución de las distancias será menor. Así pues, si se diese un proceso de selección tanto thetaW como distVar disminuirán respecto al resto del genoma. Además, cuando se selecciona un sitio del genoma la región flanqueante se selecciona junto a éste, proceso conocido como barrido de genes. Este fenómeno aumenta el desequilibrio de ligamiento, hecho que debería reflejarse en el estadístico diplo_Omega que tiene en cuenta este factor.

Así pues, en un principio thetaW y distVar disminuyen bajo un proceso de selección, mientras que diplo_Omega aumenta, por lo que deberían estar correlacionados negativamente. Sin embargo, lo que se observa es que las regiones con una menor diversidad tienen un menor desequilibrio de ligamiento. Este puede parecer, en principio, un resultado extraño, pero puede ser debido a que el tomate es una especie cultivada. El tomate cultivado tiene, en general, una diversidad muy baja debido probablemente a que ha sufrido dos cuellos de botella, uno en la migración de la región andina a Mesoamérica y otro por la migración de Mesoamérica a Europa (Blanca et. al., 2012). Durante todo este proceso puede que los agricultores no seleccionasen a favor de un único alelo sino de varios. Por ejemplo, puede que no seleccionasen sólo a favor de frutos grandes, sino a favor de distintos alelos para distintos tamaños en diferentes variedades. Es común, por ejemplo, que unas variedades se seleccionen para fruto grande y otras para fruto pequeño. Esto haría que estas regiones seleccionadas fuesen las más diversas del genoma, lo cual explicaría porque las regiones más variables tienen un mayor desequilibrio de ligamiento.

Otra posibilidad es que las asunciones en las que se basa el cálculo de diplo_Omega no sean ciertas. Como se ha mencionado anteriormente, diplo_Omega tiene en cuenta la distribución espacial del desequilibrio de ligamiento, bajo la suposición de que éste es mayor si se calcula a un lado y al otro del sitio seleccionado que si se calcula entre estos dos sitios, a lo largo de toda la región. Por ello, diplo_Omega se calcula de tal forma que aumenta conforme mayor es el LD a cada lado y menor es el LD calculado a lo largo de

toda la región, consiguiendo así tener en cuenta no solo la intensidad de este desequilibrio sino también su distribución espacial. Si esta asunción no fuera cierta y el desequilibrio de ligamiento fuera mayor calculado a lo largo de toda la secuencia, diplo_Omega disminuiría su valor conforme mayor fuera este LD. De esta forma, los tres estadísticos supuestamente correlacionados disminuirían con una selección mayor.

Por otra parte, diplo_ZnS es también otra medida de desequilibrio de ligamiento, que es utilizada para calcular diplo_Omega. De hecho diplo_Omega es una modificación de diplo_ZnS que tiene en cuenta la distribución espacial del desequilibrio en una región. Por ello no es de extrañar que diplo_ZnS aparezca correlacionado en algunos PCAs con diplo_Omega. Sin embargo esto no siempre es así. Diplo_ZnS mide el desequilibrio a lo largo de toda una región en cuyo centro se encuentra el sitio seleccionado, mientras que diplo_Omega calcula el LD a un lado del sitio seleccionado y al otro y lo divide por el LD entre esos dos sitios. Se ha observado en modelos teóricos que el desequilibrio de ligamiento suele ser muy alto a ambos lados del sitio seleccionado pero más bajo entre esos dos sitios. En los sitios en los que el desequilibrio entre ambos lados sea más bajo diplo_ZnS no estará correlacionado con diplo_Omega, mientras en los sitios en los que el LD entre los dos sitios siga siendo muy alto, sí.

Pi y nDiplos suelen estar correlacionados y esto es esperable, ya que pi es una medida de la diversidad genética, que se define como el número medio de diferencias nucleotídicas entre secuencias en una región genómica, y nDiplos es el número de haplotipos diferentes en una región. Cuando una región sufre selección positiva, el número total de haplotipos desciende (porque se selecciona a favor de uno o unos pocos) y, por tanto, al haber solo uno o unos pocos haplotipos muy frecuentes el número medio de diferencias desciende también.

Los estadísticos tajD y diplo_H2/H1 están diseñados para detectar regiones con un exceso de alelos en baja frecuencia. Un evento de selección inicialmente barre la diversidad alrededor del sitio seleccionado, pero a medida que pasa el tiempo van apareciendo mutaciones cercanas en baja frecuencia. Por lo tanto, lo que esperamos ver tras un evento de selección es un exceso de sitios con alelos en baja frecuencia. TajD se calcula como la diferencia estandarizada entre: el número medio de diferencias entre secuencias (θ_{π}) y el número total de sitios segregantes (θ_{ω}). Un exceso de alelos en baja frecuencia aumenta el número total de sitios segregantes mientras que no contribuye mucho a aumentar el número medio de diferencias, por lo que el valor de tajD se ve disminuído bajo un supuesto de selección. Por otra parte, diplo_H2/H1 se calcula como la suma de la frecuencia de todos los haplotipos menos la del haplotipo más frecuente, dividido por la frecuencia del haplotipo más frecuente. Por tanto, en el caso de haber un alelo mayoritario y el resto en baja frecuencia, el valor de este estadístico será pequeño. De esta forma, tanto tajD como diplo_H2/H1 son sensibles a los excesos de sitios con alelos en baja frecuencia y, por tanto, es esperable que ambos estadísticos se encuentren correlacionados.

Los estadísticos distKurt, distSkew y distVar están relacionados con la distribución de las distancias genéticas entre parejas de individuos dentro de la población. Siendo distVar la varianza, distKurt la curtosis y distSkew la asimetría de esta distribución. En un caso de selección la distribución de distancias será muy asimétrica, ya que estará centrada en el 0 y solo en unos pocos casos la distancia será mayor. Además, esta distribución está acotada por el 0 (el número de diferencias entre dos secuencias no puede ser menor a 0), por lo que en conjunto esta distribución tendrá una asimetría positiva. En cuanto a la curtosis, esta es una medida que determina en qué grado los valores se agrupan alrededor del valor central de la distribución, es decir, altos valores de curtosis indican que la distribución presenta

unas colas muy pequeñas y los valores están muy agrupados en un valor central y bajos valores de curtosis indican una distribución poco centrada y con unas colas muy anchas, es decir, con muchos valores atípicos (*outliers*). Por tanto, en el caso de que la selección fuera muy fuerte o reciente, los valores de la distancia estarían muy centrados en el 0 y la curtosis sería muy alta, mientras que si la selección ha sido más suave o ha pasado suficiente tiempo para que se hayan acumulado mutaciones en baja frecuencia, la distribución de las distancias presentará algunos valores atípicos consecuencia de estas mutaciones y, por tanto, la curtosis será menor. Esto explicaría el comportamiento de los estadísticos distSkew y distKurt en los PCAs: en los casos en los que la selección ha sido fuerte la distribución de las distancias presentará tanto una asimetría como una curtosis altas, mientras que si la selección ha sido débil o ha pasado bastante tiempo la asimetría seguirá siendo alta mientras que la curtosis será pequeña, produciendo que en algunos PCAs distSkew y distKurt estén correlacionados y en otros sean contrarios.

4.2 SweeD

SweeD realiza los cálculos para cada población por separado, por lo que hay que ejecutar el programa una vez para cada una de ellas. Para automatizar el proceso, se utiliza el programa `run_sweed_pops.py` (Apéndice VIII), que ejecuta SweeD de forma sucesiva, especificando en cada caso el nombre de la población y el VCF con datos únicamente de esa población. Por tanto, es necesario un paso previo para separar el VCF original en VCFs que contengan datos solo de una población.

Para generar estos VCFs se utiliza el programa `do_vcf_per_pop.py` (Apéndice IX), el cual presenta una función para obtener un listado de las muestras pertenecientes a cada una de las poblaciones y otra para separar el VCF original con la herramienta `view` del BCFtools (BCFTOOLS, 2019), diseñada para filtrar y separar archivos VCF. Para este caso los argumentos utilizados son:

- -o, --output-file. Para especificar el nombre del archivo generado. En este caso se corresponderá con el nombre de la población para la cual queremos obtener el VCF.
- -s, --samples. Para especificar una lista con las muestras que se quieren incluir en el nuevo VCF generado. En este caso se corresponde con las muestras pertenecientes a cada una de las poblaciones, obtenidas con la anterior función.

Como resultado de la ejecución de SweeD se obtienen dos tipos de archivo: uno con el prefijo “SweeD_Info” que proporciona datos sobre la ejecución realizada (comando utilizado, tiempo de ejecución, sitios descartados, etc.) y otro con el prefijo “SweeD_Report” que, para cada uno de los cromosomas de la población, presenta tres columnas: una para la posición en el cromosoma, otra para la probabilidad de que en esa región se haya producido un barrido selectivo y otra para el valor de α calculado. Todos los archivos se encuentran en el Anexo VII.

Estos resultados se representan en un gráfico. Para ello se escribe el programa `plot_sweed_results.py` (Apéndice X) que, utilizando las herramientas disponibles en la librería `matplotlib`, genera un gráfico para cada cromosoma de cada una de las poblaciones, con la región cromosómica en el eje de abscisas y la probabilidad de que en esa región se haya producido un barrido selectivo en el eje de ordenadas. Todas las gráficas se encuentran en el Anexo VIII. En la Figura 8 se recogen algunos ejemplos significativos.

Se observan diversos patrones de comportamiento. Un ejemplo de regiones en las que ha podido haber selección se observa en las Figura 8a y Figura 8d. En este caso en la mayor parte del cromosoma el SweeD ha calculado una probabilidad baja mientras que existe una región con una probabilidad mucho mayor. Esta sería, según el SweeD la región seleccionada. Es típico encontrar este tipo de patrones en las regiones cercanas al extremo de los cromosomas. En tomate estas son las regiones eucromáticas y en ellas hay una mayor densidad de genes.

Otro caso típico es el observado en la Figura 8b. La probabilidad es muy alta a lo largo de una gran parte del cromosoma. Puede que este patrón sea debido a que se ha dado selección en la heterocromatina, que en los cromosomas de tomate es una región muy grande. Por último, en otros casos se encuentran probabilidades altas dispersas a lo largo del cromosoma (Figura 8c). Esto puede deberse a que los cálculos que realiza el programa no puedan hacerse cuando el número de SNPs es muy pequeño, hecho que puede ocurrir fácilmente en tomate ya que en general las poblaciones presentan muy poca diversidad.

Todas las regiones que presentan un comportamiento como el descrito en las Figura 8a y Figura 8d y, por tanto, se considera por el SweeD que han sido seleccionadas, están anotadas en una tabla presente en el Apéndice XI.

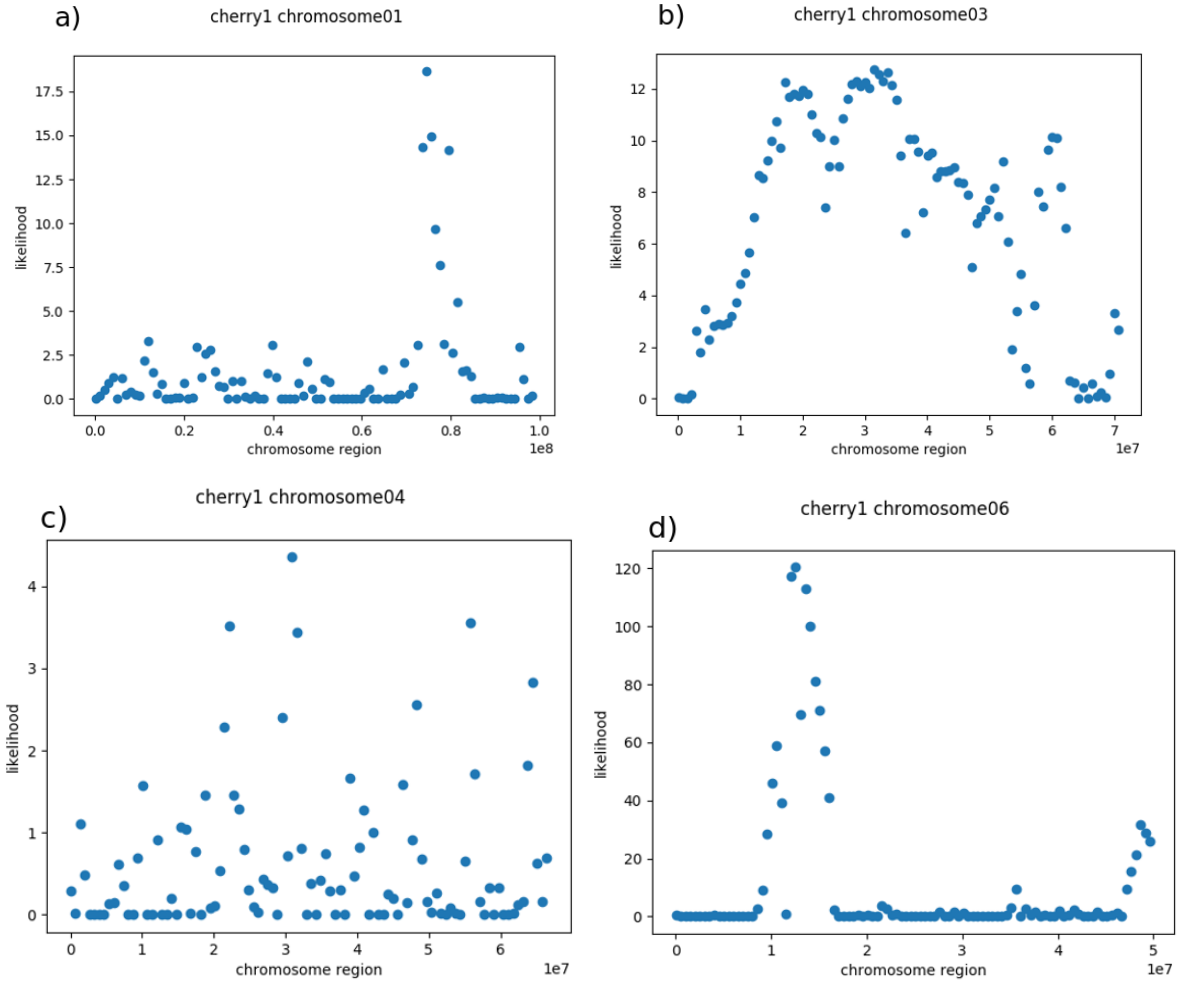


Figura 8. Representación de la probabilidad (eje de ordenadas) de que una región haya sufrido un proceso de selección positiva a lo largo del cromosoma (eje de abscisas, en pb). Esta probabilidad se calcula mediante el programa SweeD, que obtiene un ratio de probabilidades compuestas (CLR, *composite likelihood ratio*) para una determinada región. **a)** representación para el cromosoma 01 de la población cherry1. **b)** representación para el cromosoma 03 de la población cherry1. **c)**

representación para el cromosoma 04 de la población cherry1. **d)** representación para el cromosoma 06 de la población cherry1.

4.3 RaisD

RaisD calcula un estadístico propio, μ, que refleja la probabilidad de que una región haya sufrido selección. Este estadístico está compuesto por tres factores: uno que mide los cambios en el espectro de frecuencias alélicas respecto al esperado, otro que mide el desequilibrio de ligamiento y otro que mide la cantidad de variación. Estos tres se multiplican entre ellos y por la longitud de la ventana para la que se calcula μ.

RaisD analiza los datos para una sola población, por lo que es necesario ejecutar el programa una vez por cada una de ellas. Para automatizar el proceso, se utiliza el programa `run_raisd.py` (Apéndice XII), el cual presenta una función para obtener, para cada población, una lista de las muestras pertenecientes a ésta y otra función que ejecuta RaisD con los parámetros necesarios.

Como resultado de la ejecución, se generan tres tipos de archivo: un archivo con el prefijo 'RAiSD.Info' con información sobre la ejecución (comando utilizado, número de muestras, información general para cada uno de los cromosomas, etc.), un archivo para cada uno de los cromosomas de la población con el prefijo 'RAiSD.Report', con datos de la localización genómica y el valor de μ para ésta, así como el valor de los tres factores que lo componen y, por último, un archivo para cada cromosoma con el prefijo 'RAiSD_Plot' que presenta cuatro plots (uno para μ y otro para cada uno de los factores que lo componen) con la localización cromosómica en el eje de abscisas y el valor del estadístico calculado en el eje de ordenadas. En el Anexo IX se encuentran todos los archivos generados.

En la Figura 9 hay algunos ejemplos representativos. Tal como sucedía en el programa SweeD, se obtienen diferentes patrones. Se espera que las regiones que hayan sufrido selección tengan un valor más alto de μ, como ocurre en la Figura 9a y en la Figura 9b. En el Apéndice XIII se encuentran en una tabla las regiones que han presentado este patrón.

En otros casos, como en la Figura 9c y la Figura 9e, el valor de μ es alto a lo largo de todo o una mayor parte del cromosoma. Por último en la Figura 9d, se representa una región en la que los valores de μ están muy dispersos. Como se puede observar, todos estos comportamientos son bastante parecidos a los obtenidos con el programa SweeD. Del mismo modo, esto puede deberse a que debido al bajo número de SNPs los cálculos no puedan hacerse correctamente. Además, el programa RaisD, en concreto, calcula el estadístico μ en ventanas de un determinado número de SNPs, por lo que en zonas del cromosoma donde haya muy poca densidad de SNPs esta ventana será muy grande y, dado que uno de los multiplicandos es la longitud de esa ventana en pb, este hecho puede aumentar el valor de μ desproporcionadamente, dándose el patrón observado en la Figura 9c y la Figura 9e. Este patrón es el obtenido en la mayoría de las poblaciones, realmente se han podido identificar muy pocas regiones como seleccionadas. Por tanto, la aproximación de calcular por ventanas de un determinado número de SNPs en lugar de pares de bases puede no ser el más adecuado para poblaciones como el tomate, que tiene en general una densidad de SNPs pequeña.

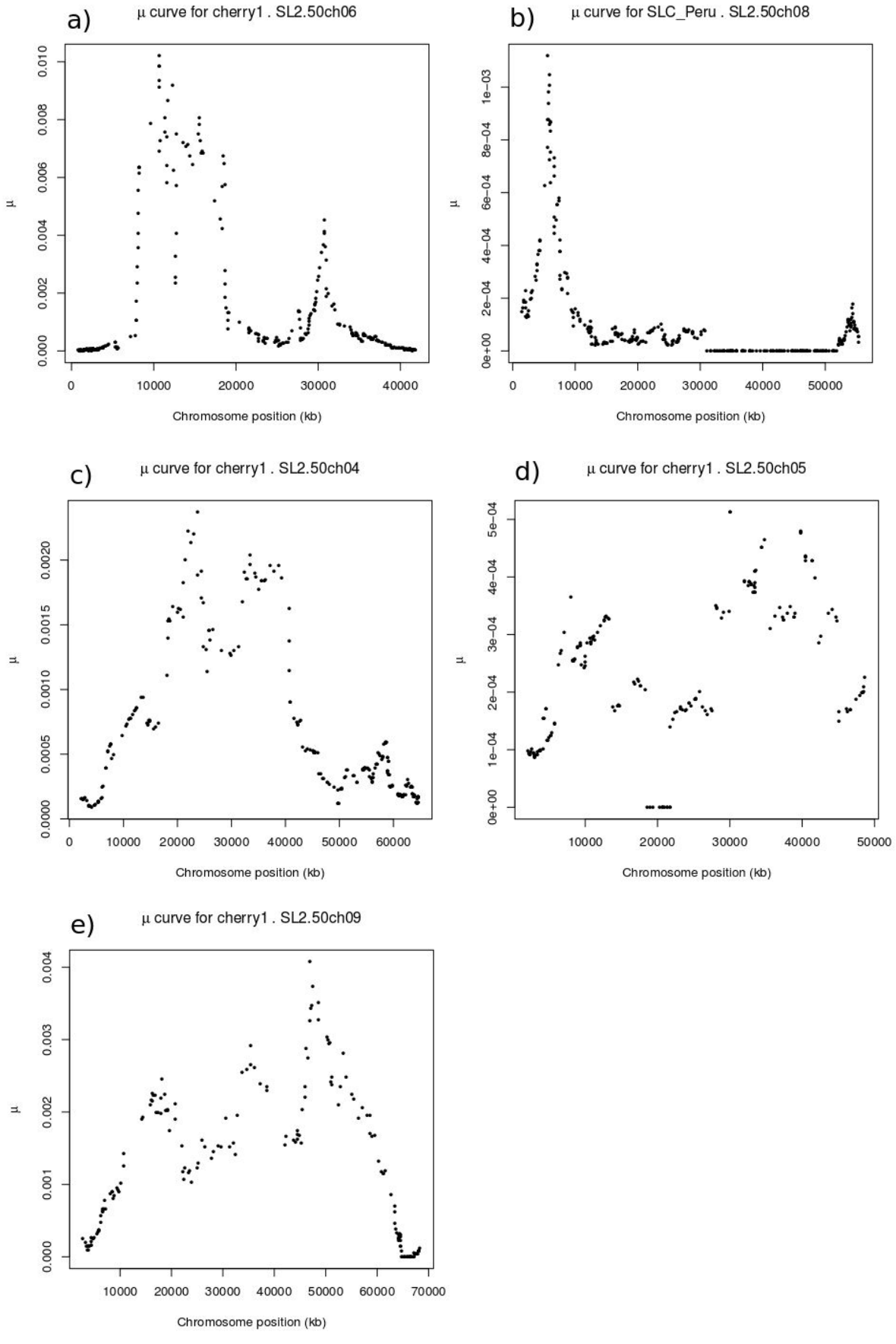

a)
μ curve for cherry1 . SL2.50ch06
μ
Chromosome position (kb)
b)
μ curve for SLC_Peru . SL2.50ch08
μ
Chromosome position (kb)
c)
μ curve for cherry1 . SL2.50ch04
μ
Chromosome position (kb)
d)
μ curve for cherry1 . SL2.50ch05
μ
Chromosome position (kb)
e)
μ curve for cherry1 . SL2.50ch09
μ
Chromosome position (kb)

Figura 9. Representación del valor del estadístico μ para las diferentes regiones cromosómicas. Cada gráfica se corresponde con un cromosoma de una población en concreto. El estadístico μ es propio del programa RaisD y tiene en cuenta tanto los cambios en el SFS como el LD y la cantidad de variación. **a)** representación para el cromosoma 06 de la población cherry1. **b)** representación para el cromosoma 08 de la población SLC_Peru. **c)** representación para el cromosoma 04 de la población cherry1. **d)** representación para el cromosoma 05 de la población cherry1. **e)** representación para el cromosoma 09 de la población cherry1.

4.4 BayeScan

BayeScan utiliza un formato de archivo propio llamado GESTE, así que antes de ejecutarlo es necesario convertir los VCF a este formato. Para ello se utiliza el programa PGDSpider, que puede descargarse a través de su página web (PGDSPIDER, 2019), donde se encuentran también las instrucciones para instalarlo y un manual con las instrucciones de uso (en el apartado *help*).

BayeScan necesita un archivo GESTE con la información para todas las poblaciones, por lo que una vez se han convertido cada uno de los VCF de las poblaciones, es necesario recogerlo todo en un mismo archivo, dado que no se puede hacer todo en un mismo paso con el PGDSpider. Para ello se escribe el programa `vcf_to_bayescan_format.py` (Apéndice XIV), que presenta una función que ejecuta el PGDSpider una vez por cada VCF de cada población para convertirlo al formato GESTE y, posteriormente, presenta una función que recoge todos los GESTEs en un mismo archivo, siguiendo las características del formato. Todos los archivos están disponibles en el Anexo XI.

El BayeScan es un programa clásico de búsqueda de regiones seleccionadas. Cuando se ejecuta este programa, tal como se describe en el apartado materiales y métodos, se detiene su ejecución de forma anómala y se obtiene el siguiente error:

```
Using 2 threads (2 cpu detected on this machine)

Pilot runs...

Violación de segmento (`core' generado)
```

Los errores de tipo “violacion de segmento” se dan cuando un programa trata de acceder a un sitio de la memoria para la cual no tiene acceso. En la documentación no se encuentra ninguna solución para este problema, por lo que la única manera de solucionarlo sería buscar en el código fuente los posibles sitios en los que se podría dar este error.

4.5 LOSITAN

LOSITAN utiliza un formato de archivo llamado Genepop, por lo que hay que convertir los VCF de cada población a este formato, mediante el programa PGDSpider. Para ello se escribe el programa `vcf2genepop_pgdspider.py` (Apéndice XV), que presenta una función que ejecuta el PGDSpider una vez por cada VCF de cada población para convertirlo al formato Genepop y, posteriormente, presenta una función que recoge todos los Genepops en un mismo archivo, siguiendo las características del formato, puesto que el PGDSpider no lo hace todo en un mismo paso. Todos los archivos están disponibles en el Anexo XII.

Para instalar LOSITAN se ha intendado seguir las recomendaciones del GitHub de Baskar Lingam (ver el apartado materiales y métodos). El instalador trata de descargar los componentes necesarios, pero la mayoría de los links de descarga a los que hace referencia este programa están obsoletos y, aún habiendo cambiado los links por otros que que apuntan a versiones nuevas, sigue habiendo errores al ejecutar el segundo programa para instalar la versión *offline*, `prepareLibs.sh`.

Por tanto, debido a que el proyecto de LOSITAN ha sido abandonado y no está actualizado, no ha sido posible ejecutarlo.

4.6 Comparación de los resultados obtenidos

Para comparar si las regiones que presentan comportamientos típicos de regiones seleccionadas en los diferentes programas coinciden, se escribe el programa `compare_selected_regions.py` (Apéndice XVI). Este programa tiene una función para, a partir de uno de los csv con las regiones anotadas para un programa, construir un diccionario donde se guarden, para cada cromosoma de cada población, los rangos en los cuales se encuentran los picos seleccionados. Por ejemplo, si se ha identificado una región seleccionada en la población cherry1, en el cromosoma 06 en la posición 50000000, se guarda en el diccionario que hay una sitio seleccionado en la región entre (45000000, 55000000), para así permitir un margen de error. Posteriormente, se comprueba en otra función si las regiones identificadas en otro programa y anotadas en otro csv se encuentra dentro de los rangos guardados, lo que significa que se han identificado como seleccionadas por los dos programas.

Al comparar las regiones identificadas como seleccionadas por DiploSHIC y SweeD, se obtiene que coinciden en los sitios anotados en la Tabla 2.

Tabla 2. Sitios identificados como seleccionados tanto por DiploSHIC como por SweeD.

población	**cromosoma**	**sitio (pb)**
cherry1	SL2.50ch06	45750000
cherry1	SL2.50ch06	46050000
cherry1	SL2.50ch07	150000
cherry1	SL2.50ch07	950000
cherry1	SL2.50ch12	1750000
modern1	SL2.50ch08	64450000
modern2	SL2.50ch02	50550000
modern2	SL2.50ch06	40450000
modern2	SL2.50ch06	46050000
modern2	SL2.50ch09	65950000
modern2	SL2.50ch09	1450000

out_core_b	SL2.50ch02	47550000
out_core_f	SL2.50ch12	4450000
processing	SL2.50ch04	4150000
processing	SL2.50ch04	61950000

Los sitios coincidentes son muy pocos, comparado con el total de sitios identificados por cada uno de los programas (alrededor de 100 en cada caso). Además, se ha visto que mediante el SweeD se han podido identificar sitios seleccionados en poblaciones para las cuales se obtienen muy pocos resultados con el programa DiploSHIC y, por tanto, no se puede sacar conclusiones a partir de los PCAs. Incluso en algunos casos con el SweeD se identifican sitios en cromosomas que con el SweeD no, y viceversa.

Comparando los sitios obtenidos con DiploSHIC y RaisD con el mismo programa no se obtiene ninguna región coincidente. Con el programa RaisD se pudieron anotar muy pocos sitios seleccionados y estos, como ocurría con el SweeD, pertenecen muchas veces a poblaciones para las cuales no se ha podido obtener resultados con DiploSHIC.

Por último, al comparar RaisD y SweeD con el mismo programa se obtiene que coinciden para los sitios anotados en la Tabla 3.

Tabla 3. Sitios identificados como seleccionados tanto por RaisD como por SweeD.

población	**cromosoma**	**sitio (pb)**
SLC_Ecu	SL2.50ch03	55000000
SLC_Ecu	SL2.50ch06	35000000

Como se puede observar, el número de regiones coincidentes entre estos dos programas también es pequeño, aunque cabe puntualizar que el número de regiones anotadas con el programa RaisD es bastante reducido.

Con los programas RaisD y SweeD se han identificado regiones como seleccionadas en poblaciones que con DiploSHIC no se ha podido evaluar debido a los pocos resultados obtenidos. Además, a nivel de cromosoma con DiploSHIC se obtenían resultados únicamente para algunas regiones, mientras que con los otros dos programas se obtenían a lo largo de todo el cromosoma. Esto podría ser indicativo de que SweeD y RaisD son más tolerantes que DiploSHIC a un número pequeño de muestras y de datos genéticos. Esto no tiene por que ser una ventaja ya que puede ocurrir que, aunque SweeD y RaisD obtuvieran resultados con pocos datos, éstos fueran estimaciones erróneas. En general, cuantos más datos de partida haya más ajustadas serán las estimas de los distintos parámetros. En este caso, DiploSHIC estaría siendo más restrictivo y generaría un menor número de falsos positivos por lo que, aunque el número de resultados fuera menor, estos serían más fiables.

Además, cada uno de los programas mide parámetros distintos que se basan en asunciones diferentes, por eso podrían no estar dando los mismos resultados. Los estadísticos usados normalmente se diseñan para detectar selección en poblaciones

ideales (en equilibrio, con un tamaño de población efectivo constante, sin migración, etc.) y cada uno es más o menos sensible a la violación de estas asunciones.

Otro posible motivo por el cual las regiones detectadas han sido diferentes podría ser debido a la diferencia en el tamaño de la ventana utilizada para recorrer el cromosoma en cada uno de los programas. En DiploSHIC el tamaño es 100000 pb, en SweeD 100 pb y en RaisD la ventana era de un determinado número de SNPs, por lo que el tamaño en pb es variable. El valor del o de los estadísticos de cada programa se calcula utilizando todos los SNPs de cada ventana, pero el valor calculado se le asigna al sitio que se encuentra en el centro de ésta. Por ello, el tamaño de la ventana también podría influir en el cálculo y en el sitio en el que se identifica la selección.

Además, los programas de detección de selección utilizan estadísticos que se basan en ciertas asunciones, por ejemplo, suelen estar pensados para detectar selección en poblaciones ideales (en equilibrio Hardy-Weinberg, sin migración, con un tamaño efectivo de la población constante, etc.). Sin embargo, la mayoría de las poblaciones se alejan bastante de estos modelos teóricos, por lo que los programas pueden generar resultados espurios al analizar estas poblaciones, dependiendo de lo sensibles que sean a la violación de las asunciones en las que se basan.

Además, el tomate es una especie que durante su proceso de domesticación ha sufrido dos grandes cuellos de botella, ha sido fuertemente seleccionada por los agricultores y los mejoradores y es casi autógama. Esto hace que el tomate cultivado sea una especie con una diversidad genética muy baja y un desequilibrio de ligamiento muy alto en todo el genoma, por lo que resulta muy difícil buscar marcas de selección en este fondo genético.

5. Conclusiones.

1. Se han realizado búsquedas de regiones genómicas potencialmente seleccionadas durante el proceso de domesticación utilizando distintas aproximaciones.
2. Se han creado programas para realizar automáticamente cientos de análisis con los programas DiploSHIC, SweeD, y RaisD.
3. Se ha creado un conjunto de programas que permite hacer un análisis comparativo de los resultados obtenidos a partir de las distintas aproximaciones de detección de regiones seleccionadas.
4. Los programas BayeScan y LOSITAN han sido abandonados por sus autores y es difícil realizar análisis con estos algoritmos clásicos.
5. Hay una coincidencia baja entre los análisis de los diferentes programas.
6. El tomate cultivado tiene una diversidad genética muy reducida que ha afectado negativamente, debido a la falta de SNPs, a la detección de las regiones seleccionadas.
7. Los distintos programas hacen predicciones basándose en asunciones diferentes sobre el proceso evolutivo y pueden ser sensibles en distinto grado a la violación de estas asunciones en una especie con una historia compleja que incluye varios cuellos de botella y numerosas migraciones e hibridaciones. En estos casos tal vez sea más conveniente establecer las conclusiones en base a análisis no paramétricos, como por ejemplo el análisis de componentes principales.

6. Bibliografía

ALACHIOTIS, N. AND PAVLIDIS, P. (2018). RAiSD detects positive selection based on multiple signatures of a selective sweep and SNP vectors. *Communications Biology*, 1(1).

ALACHIOTIS, N., STAMATAKIS, A. AND PAVLIDIS, P. (2012). OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets. *Bioinformatics*, 28(17), pp.2274-2275.

ANTAO, T., LOPES, A., LOPES, R., BEJA-PEREIRA, A. AND LUIKART, G. (2008). LOSITAN: A workbench to detect molecular adaptation based on a Fst-outlier method. *BMC Bioinformatics*, 9(1).

BEGUN, D. AND AQUADRO, C. (1991). Molecular population genetics of the distal portion of the X chromosome in Drosophila: evidence for genetic hitchhiking of the yellow-achaete region. *Genetics*, 129(4), pp.1147-1158.

BERRY, A., AJIOKA, J. AND KREITMAN, M. (1991). Lack of polymorphism on the Drosophila fourth chromosome resulting from selection. *Genetics*, 129(4), pp.1111-1117.

BLANCA, J., MONTERO-PAU, J., SAUVAGE, C., BAUCHET, G., ILLA, E., DÍEZ, M., FRANCIS, D., CAUSSE, M., VAN DER KNAAP, E. AND CAÑIZARES, J. (2015). Genomic variation in tomato, from wild ancestors to contemporary breeding accessions. *BMC Genomics*, 16(1).

BRAVERMAN, J., HUDSON, R., KAPLAN, N., LANGLEY, C. AND STEPHAN, W. (1995). The Hitchhiking Effect on the Site Frequency Spectrum of DNA Polymorphisms. *Genetics*, 140(2), pp.783-796.

CAMPOS, J., ZHAO, L. AND CHARLESWORTH, B. (2017). Estimating the parameters of background selection and selective sweeps inDrosophilain the presence of gene conversion. *Proceedings of the National Academy of Sciences*, 114(24), pp.E4762-E4771.

CHARLESWORTH, B. (1994). The effect of background selection against deleterious mutations on weakly selected, linked variants. *Genetical Research*, 63(3), pp.213-227.

CHARLESWORTH, B. (1996). Background selection and patterns of genetic diversity in Drosophila melanogaster. *Genetical Research*, 68(2), pp.131-149.

CHARLESWORTH, B. (2012). The Effects of Deleterious Mutations on Evolution at Linked Sites. *Genetics*, 190(1), pp.5-22.

CHARLESWORTH, B., MORGAN, M. AND CHARLESWORTH, D. (1993). The effect of deleterious mutations on neutral molecular variation. *Genetics*, 134(4), pp.1289-1303.

CMPG.UNIBE.CH. (2019). *BayeScan*. [online] Available at: http://cmpg.unibe.ch/software/BayeScan/ [Accessed 19 Jun. 2019].

CMPG.UNIBE.CH. (2019). *PGDSpider version 2.1.1.5*. [online] Available at: http://www.cmpg.unibe.ch/software/PGDSpider/ [Accessed 19 Jun. 2019].

CORNUET, J. AND LIUKART, G. (1996). Description and Power Analysis of Two Tests for Detecting Recent Population Bottlenecks From Allele Frequency Data. *Genetics*, 144(4), pp.2001-2004.

DOCS.PYTHON.ORG. (2019). *The Python Standard Library — Python 2.7.16 documentation*. [online] Available at: https://docs.python.org/2/library/index.html [Accessed 19 Jun. 2019].

DOEBLEY, J., GAUT, B. AND SMITH, B. (2006). The Molecular Genetics of Crop Domestication. *Cell*, 127(7), pp.1309-1321.

FAO.ORG. (2019). *FAOSTAT*. [online] Available at: http://www.fao.org/faostat/en/#home [Accessed 19 Jun. 2019].

FAY, J. AND WU, C. (2002). Hitchhiking Under Positive Darwinian Selection. *Genetics*, 155(3), pp.1405-1413.

FOLL, M. AND GAGGIOTTI, O. (2008). A Genome-Scan Method to Identify Selected Loci Appropriate for Both Dominant and Codominant Markers: A Bayesian Perspective. *Genetics*, 180(2), pp.977-993.

FU, Y. (2019). Statistical Tests of Neutrality of Mutations Against Population Growth, Hitchhiking and Background Selection. *Genetics*, 147(2), pp.915-925.

GITHUB. (2019). *alachins/raisd*. [online] Available at: https://github.com/alachins/raisd [Accessed 19 Jun. 2019].

GITHUB. (2019). *alachins/sweed*. [online] Available at: https://github.com/alachins/sweed [Accessed 19 Jun. 2019].

GITHUB. (2019). *BaskiGIT/lositan*. [online] Available at: https://github.com/BaskiGIT/lositan [Accessed 19 Jun. 2019].

GITHUB. (2019). *kern-lab/diploSHIC*. [online] Available at: https://github.com/kern-lab/diploSHIC [Accessed 19 Jun. 2019].

GITHUB. (2019). *tiagoantao/lositan*. [online] Available at: https://github.com/tiagoantao/lositan [Accessed 19 Jun. 2019].

GUTTMAN, D. AND DYKHUIZEN, D. (1994). Detecting selective sweeps in naturally occurring Escherichia coli. *Genetics*, 138(4), pp.993-1003.

HARTL, D. AND CLARK, A. (2018). *Principles of population genetics*. Sunderland, Mass.: Sinauer Associates.

HUDSON, R. AND KAPLAN, N. (1995). Deleterious Background Selection with Recombination. *Genetics*, 141(4), pp.1605-1617.

HUDSON, R., KREITMAN, M. AND AGUADÉ, M. (1987). A Test of Neutral Molecular Evolution Based on Nucleotide Data. *Genetics*, 166(1), pp.153-159.

ICS.HUTTON.AC.UK. (2019). *CurlyWhirly : Information & Computational Sciences*. [online] Available at: https://ics.hutton.ac.uk/curlywhirly/ [Accessed 19 Jun. 2019].

JENSEN, J. (2014). On the unfounded enthusiasm for soft selective sweeps. *Nature Communications*, 5(1).

KEIGHTLEY, P. AND EYRE-WALKER, A. (1999). Terumi Mukai and the Riddle of Deleterious Mutation Rates. *Genetics*, 153(2), pp.515-523.

KELLY, J. (1997). A Test of Neutrality Based on Interlocus Associations. *Genetics*, 146(3), pp.1197-1206.

KERN, A. AND SCHRIDER, D. (2018). diploS/HIC: An Updated Approach to Classifying Selective Sweeps. *G3: Genes|Genomes|Genetics*, 8(6), pp.1959-1970.

KIM, Y. AND STEPHAN, W. (2000). Joint effects of genetic hitchhiking and background selection on neutral variation. *Genetics*, 155(3), pp.1415-1423.

KIM, Y. AND STEPHAN, W. (2002). Detecting a Local Signature of Genetic Hitchhiking Along a Recombining Chromosome. *Genetics*, 160(2), pp.765-777.

KIMURA, M. (1968). Evolutionary Rate at the Molecular Level. *Nature*, 217(5129), pp.624-626.

MAYNARD, J. and HAIGH, J. (2007). The hitch-hiking effect of a favourable gene. *Genetics Research*, 89(5-6), pp.391-403.

NEI, M., MARUYAMA, T. AND CHAKRABORTY, R. (1975). THE BOTTLENECK EFFECT AND GENETIC VARIABILITY IN POPULATIONS. *Evolution*, 29(1), pp.1-10.

NIELSEN, R. (2005). Genomic scans for selective sweeps using SNP data. *Genome Research*, 15(11), pp.1566-1575.

NUMPY.ORG. (2019). *NumPy — NumPy*. [online] Available at: http://www.numpy.org/ [Accessed 19 Jun. 2019].

PANDAS.PYDATA.ORG. (2019). *Python Data Analysis Library — pandas: Python Data Analysis Library*. [online] Available at: https://pandas.pydata.org/ [Accessed 19 Jun. 2019].

PAVLIDIS, P., ŽIVKOVIĆ, D., STAMATAKIS, A. AND ALACHIOTIS, N. (2013). SweeD: Likelihood-Based Detection of Selective Sweeps in Thousands of Genomes. *Molecular Biology and Evolution*, 30(9), pp.2224-2234.

PERALTA, I., SPOONER, D. AND KNAPP, S. (2019). *Taxonomy of wild tomatoes and their relatives (Solanum sect. Lycopersicoides, sect. Juglandifolia, sect. Lycopersicon; Solanaceae)*.. [online] Cabdirect.org. Available at: https://www.cabdirect.org/cabdirect/abstract/20103064022 [Accessed 19 Jun. 2019].

PYTHON.ORG. (2019). *Welcome to Python.org*. [online] Available at: https://www.python.org/ [Accessed 19 Jun. 2019].

RONEN, R., UDPA, N., HALPERIN, E. AND BAFNA, V. (2013). Learning Natural Selection from the Site Frequency Spectrum. *Genetics*, 195(1), pp.181-193.

SAMTOOLS.GITHUB.IO. (2019). *bcftools*. [online] Available at: https://samtools.github.io/bcftools/bcftools.html [Accessed 19 Jun. 2019].

SCIKIT-LEARN.ORG. (2019). *scikit-learn: machine learning in Python — scikit-learn 0.21.2 documentation*. [online] Available at: https://scikit-learn.org/stable/ [Accessed 19 Jun. 2019].

SCIKIT-LEARN.ORG. (2019). *Importance of Feature Scaling — scikit-learn 0.21.2 documentation*. [online] Available at: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py [Accessed 19 Jun. 2019].

STEPHAN, W., WIEHE, T. AND LENZ, M. (1992). The effect of strongly selected substitutions on neutral polymorphism: Analytical results based on diffusion theory. *Theoretical Population Biology*, 41(2), pp.237-254.

TAJIMA, F. (1989). Statistical Method for Testing the Neutral Mutation Hypothesis by DNA Polymorphism. *Genetics*, 123(3), pp.585-595.

TANG, K., THORNTON, K. AND STONEKING, M. (2007). A New Approach for Using Genome Scans to Detect Recent Positive Selection in the Human Genome. *PLoS Biology*, 5(7), p.e171.

THE TOMATO GENOME CONSORTIUM (2012). The tomato genome sequence provides insights into fleshy fruit evolution. *Nature*, 485(7400), pp.635-641.

TRADITOM.EU. (2019). *TRADITOM*. [online] Available at: http://traditom.eu/es/ [Accessed 19 Jun. 2019].

VCFTOOLS.GITHUB.IO. (2019). *VCFtools*. [online] Available at: https://vcftools.github.io/man_latest.html [Accessed 19 Jun. 2019].

VITTI, J., GROSSMAN, S. AND SABETI, P. (2013). Detecting Natural Selection in Genomic Data. *Annual Review of Genetics*, 47(1), pp.97-120

Apéndice I

calculate_statistics.py

```
from pathlib import Path
import subprocess
from collections import defaultdict

BASE_DIR = Path('/home/marta/devel')
SHIC_DIR = BASE_DIR / 'diploSHIC'
POPS_VCFS_PATH = SHIC_DIR / 'pop_chr_vcfs'
CALC_STATS_PY_PATH = SHIC_DIR /
'diploSHIC/makeFeatureVecsForChrArmFromVcfDiploid.py'
NO_PERIC_FPATH = BASE_DIR / 'tomato/no_pericentromeric_regions.bed'

TOMATO_DIR = Path('/home/marta/devel/tomato')
TRADITOM_DIR = TOMATO_DIR / 'traditom'
VCF_FPATH = TRADITOM_DIR /
'snps/traditom-20180531_onlygbs.tier1.no_bad_samples.marta.vcf.gz'
SAMPLE_TO_POP_FPATH = SHIC_DIR / 'sample_to_pop.txt'
RESULTS_DIR = SHIC_DIR / 'results'


# shicmode chrArmvcffile chrarm chrlen fvecfilename --targetPop --sampleToPopFileName
def calculate_stats_per_pop_and_chr(vcf_fpath, no_peric_fpath, sample_to_pop_fpath,
results_dir):
        chr_lengths = calculate_chr_length(no_peric_fpath)
        chromosomes = list(chr_lengths.keys())
        pops = [line.strip().split('\t')[1] for line in sample_to_pop_fpath.open()]
        pops = set(pops)
        for pop in pops:
        for chr in chromosomes:
        output_name = pop + '_' + chr
        dshic_command = ['python', 'diploSHIC/diploSHIC.py', 'fvecVcf',
                        'diploid',
                        str(vcf_fpath),
                        chr,
                        str(chr_lengths[chr]),
                        str(results_dir/output_name),
                        '--targetPop', pop,
                        '--sampleToPopFileName', str(sample_to_pop_fpath)]
        subprocess.run(dshic_command)


def calculate_stats_all_pops_per_chr(vcf_fpath, no_peric_fpath,results_dir):
        chr_lengths = calculate_chr_length(no_peric_fpath)
        chromosomes = list(chr_lengths.keys())
        for chr in chromosomes:
        output_name = 'all_pops_' + chr
        dshic_command = ['python', '/home/marta/devel/diploSHIC/diploSHIC/diploSHIC.py',
'fvecVcf',
```

```
                'diploid',
                str(vcf_fpath),
                chr,
                str(chr_lengths[chr]),
                str(results_dir/output_name)]
        subprocess.run(dshic_command)


def calculate_chr_length(no_peric_fpath):
        chromosomes_lengths = defaultdict(int)
        for line in no_peric_fpath.open():
        chr, start, end = line.strip().split('\t')
        chromosomes_lengths[chr] = int(end)
        return chromosomes_lengths


def main():
        vcf_fpath = VCF_FPATH
        no_peric_fpath = NO_PERIC_FPATH
        calc_stats = CALC_STATS_PY_PATH
        shic_dir = SHIC_DIR
        sample_to_pop_fpath = SAMPLE_TO_POP_FPATH
        results_dir = RESULTS_DIR
        calculate_stats_per_pop_and_chr(vcf_fpath, no_peric_fpath, sample_to_pop_fpath,
results_dir)


if __name__ == '__main__':
        main()
```

Apéndice II

do_sample_to_pop_file.py

```
from pathlib import Path
from csv import DictReader

TOMATO_DIR = Path('/home/marta/devel/tomato')
TRADITOM_DIR = TOMATO_DIR / 'traditom'
CLASSIFICATIONS_FPATH = TRADITOM_DIR / 'traditom_classifications.csv'

def make_sample_to_pop_file(classif_fpath):
	classif_fhand = classif_fpath.open()
	samp_to_pop_f = open('sample_to_pop.txt', 'w')
	for accesion in DictReader(classif_fhand):
	sample_id = accesion['sample']
	pop = accesion['rank1']
	line_to_write = sample_id + '\t' + pop + '\n'
	samp_to_pop_f.write(line_to_write)

def main():
	classif_fpath = CLASSIFICATIONS_FPATH
	make_sample_to_pop_file(classif_fpath)

if __name__ == '__main__':
	main()
```

Apéndice III

reorganize_and_plot_results.py

```
from pathlib import Path
from collections import defaultdict, OrderedDict
from statistics import mean
import matplotlib.pyplot as plt

RESULTS_DPATH = Path('/home/marta/devel/diploSHIC/results_stats')
STATS_NAMES = ['pi', 'thetaW', 'tajD', 'distVar', 'distSkew', 'distKurt', 'nDiplos', 'diplo_H1',
'diplo_H12', 'diplo_H2/H1', 'diplo_ZnS', 'diplo_Omega' ]



def reorganize_results(results_dpath):
        for pop_chr_res_file in results_dpath.iterdir():
        pop_chr_name = str(pop_chr_res_file).split('/')[-1]
        stats_per_subwin_dic = _obtain_stats_per_subwin(pop_chr_res_file)
        if not stats_per_subwin_dic:
        yield (pop_chr_name, None)
        continue
        mean_stats_per_subwin = _obtain_mean_stat_per_subwin(stats_per_subwin_dic)
        yield (pop_chr_name, mean_stats_per_subwin)


def plot_results(results_dpath, stats_names):
        for pop_chr_name, mean_stats_per_subwin in reorganize_results(results_dpath):
        if not mean_stats_per_subwin:
        print('NO DATA FOR ', pop_chr_name)
        continue
        mean_stats_per_subwin = OrderedDict(sorted(mean_stats_per_subwin.items()),
key=lambda t: t[0][0])

        #gather x and y values
        x_values = []
        y_values_dic = defaultdict(list)
        for region, stats_values in mean_stats_per_subwin.items():
        if region == 'key':
                continue
        region_start, region_end = region
        x_value = (region_start + region_end)/2
        x_values.append(x_value)
        for stat, value in stats_values.items():
                y_values_dic[stat].append(value)

        #establish x limits if needed
        x_lims = []
        n = 0
        while n < len(x_values)-1:
        if x_values[n+1] - x_values[n]>= 10000000:
                x_lims.append((x_values[n], x_values[n+1]))
        n +=1
```

```
#plot truncated x axis
if x_lims:
fig, axs = plt.subplots(12, len(x_lims)+1, sharey=True)
plt.subplots_adjust(hspace=0.7, wspace=0.05)
plt.xlabel('chromosomic region', fontsize=30)
fig.set_size_inches(30,30)
plt.suptitle(pop_chr_name+' truncated', fontsize = 70)
for idx_y, stat_name in enumerate(stats_names):
        y_values = y_values_dic[stat_name]
        axs[idx_y, 0].set_ylabel(stat_name, fontsize=20)
        axs[idx_y, 0].set_xlim(left=0)
        for idx_x, x_lim in enumerate(x_lims):
        axs[idx_y, idx_x].plot(x_values, y_values, linestyle='-', color='b', marker='o')
        axs[idx_y, idx_x].set_xscale('linear')
        axs[idx_y, idx_x].set_xlim(right=x_lim[0])
        axs[idx_y, idx_x].spines['right'].set_visible(False)
        axs[idx_y, idx_x].yaxis.tick_left()
        axs[idx_y, idx_x].tick_params(labelright='off')
        axs[idx_y, idx_x].ticklabel_format(axis='x', style='plain')

        axs[idx_y, idx_x+1].plot(x_values, y_values, linestyle='-', color='b', marker='o')
        axs[idx_y, idx_x+1].set_xscale('linear')
        axs[idx_y, idx_x+1].set_xlim(left=x_lim[1])
        axs[idx_y, idx_x+1].spines['left'].set_visible(False)
        axs[idx_y, idx_x+1].yaxis.tick_right()
        axs[idx_y, idx_x+1].ticklabel_format(axis='x', style='plain')

        d = .015
        kwargs = dict(transform=axs[idx_y, idx_x].transAxes, color='k', clip_on=False)
        axs[idx_y, idx_x].plot((1-d,1+d), (-d,+d), **kwargs)
        axs[idx_y, idx_x].plot((1-d,1+d),(1-d,1+d), **kwargs)
        kwargs.update(transform=axs[idx_y, idx_x+1].transAxes)
        axs[idx_y, idx_x+1].plot((-d,+d), (1-d,1+d), **kwargs)
        axs[idx_y, idx_x+1].plot((-d,+d), (-d,+d), **kwargs)
plt.savefig(pop_chr_name + '_truncated.png')
plt.close()

#plot normal x axis
fig, axs = plt.subplots(12,1)
plt.subplots_adjust(hspace = 0.7)
plt.xlabel('chromosomic region', fontsize=30)
fig.set_size_inches(30,30)
plt.suptitle(pop_chr_name, fontsize = 70)
for idx, stat_name in enumerate(stats_names):
y_values = y_values_dic[stat_name]
axs[idx].plot(x_values, y_values, linestyle='-', color='b',
        marker='o')
axs[idx].set_ylabel(stat_name, fontsize=20)
axs[idx].ticklabel_format(axis='x', style='plain')
plt.savefig(pop_chr_name + '.png')
plt.close()
```

```
def _obtain_mean_stat_per_subwin(stats_per_subwin_dic):
        mean_stat_per_subwin = defaultdict(dict)
        for region, stats in stats_per_subwin_dic.items():
        for stat, values in stats.items():
        mean_value = mean(values)
        mean_stat_per_subwin[region][stat] = mean_value
        return mean_stat_per_subwin


def _obtain_stats_per_subwin(pop_chr_res_file):
        header = []
        first_line = True
        subwin_stats = defaultdict(dict)
        for line in pop_chr_res_file.open():
        if first_line:
        header = line.strip().split('\t')
        first_line = False
        continue

        stats = line.strip().split('\t')
        if not stats:
        return None

        pos = header.index('bigWinRange')
        big_win_range = stats[pos]
        win_start, win_end = big_win_range.split('-')
        win_start, win_end = int(win_start)-1, int(win_end)
        step = int((win_end - win_start)/11)
        sub_wins = []
        sub_win_start = win_start
        while sub_win_start < win_end:
        sub_win_end = sub_win_start + step
        sub_win = (sub_win_start, sub_win_end)
        sub_wins.append(sub_win)
        sub_win_start += step

        for idx, stat in enumerate(stats):
        stat_name = header[idx]
        if '_win' in stat_name:
                if len(stat_name[stat_name.find('_win'):]) == 6:
                subwin_pos = int(stat_name[-2:])
                else:
                subwin_pos = int(stat_name[-1])
                subwin = sub_wins[subwin_pos]
                stat_name = stat_name[:stat_name.find('_win')]
                if stat_name not in subwin_stats[subwin]:
                subwin_stats[subwin][stat_name] = []
                subwin_stats[subwin][stat_name].append(float(stat))
        return subwin_stats
```

```
def main():
	results_dpath = RESULTS_DPATH
	stats_names = STATS_NAMES
	plot_results(results_dpath, stats_names)


if __name__ == '__main__':
	main()
```

Apéndice IV

low_n_snps_test.py

```
from pathlib import Path
import matplotlib.pyplot as plt
import subprocess
import sys


VCFS_DIR = Path('/home/marta/devel/diploSHIC/pop_chr_vcfs')
TEST_DIR = Path('/home/marta/devel/diploSHIC/low_n_snps_test')
print(sys.argv)
POP_CHR_NAME = sys.argv[1]
FILTER = sys.argv[2]

def do_kept_sites_file(pop_chr_name, vcfs_dir, filter):
	file_name = pop_chr_name + '.vcf'
	vcf_fpath = vcfs_dir / file_name
	vcftools_cmd = ['vcftools',
		'--vcf', str(vcf_fpath),
		'--kept-sites']
	if filter:
	vcftools_cmd += ['--min-alleles', '2',
		'--max-alleles', '2',
		'--max-missing', '0.75',
		'--out', pop_chr_name]
	else:
	vcftools_cmd += ['--out', pop_chr_name + '_no_filter']

	subprocess.run(vcftools_cmd)
	print(' '.join(vcftools_cmd))


def plot_kept_sites(pop_chr_name, test_dir, filter):
	if filter:
	file_name = pop_chr_name + '.kept.sites'
	else:
	file_name = pop_chr_name + '_no_filter' + '.kept.sites'
	kept_sites_fpath = test_dir / file_name

	regions = []
	first_line = True
	for line in kept_sites_fpath.open():
	if first_line:
	first_line = False
	continue
	chr, region = line.strip().split()
	regions.append(int(region))

	y_values = [1 for x in range(len(regions))]
	pop_name = str(kept_sites_fpath).split('/')[-1]
```

```
        pos = pop_name.find('.k')
        pop_name = pop_name[:pos]

        graphic = plt.plot(regions, y_values, '.')
        plt.savefig(pop_name + '.png')


def get_pandas_num_snps(res_dir):
        num_snps_dic = defaultdict(list)
        all_pops_list = []
        for kept_file_fpath in res_dir.iterdir():
        if 'log' not in str(kept_file_fpath):
        continue
        pop_name = str(kept_file_fpath).split('/')[-1].split('_')[:-1]
        pop_name = '_'.join(pop_name)
        if pop_name not in all_pops_list:
        all_pops_list.append(pop_name)
        chr_name =  str(kept_file_fpath).split('_')[-1].split('.log')[0]
        for line in kept_file_fpath.open():
        if "filtering," in line and "Sites" in line:
                line = line.split(' ')
                num_filtered_snps = (int(line[3]),'filtered')
                num_total_snps = (int(line[8]), 'unfiltered')
        for num_snps, filtered in [num_total_snps, num_filtered_snps]:
        num_snps_dic['pop'].append(pop_name)
        num_snps_dic['chr'].append(chr_name)
        num_snps_dic['SNPs'].append(num_snps)
        num_snps_dic[' '].append(filtered)
        num_snps_pandas = pd.DataFrame(num_snps_dic)
        num_snps_pandas_sorted = num_snps_pandas.sort_values(['pop','chr'])
        return (all_pops_list, num_snps_pandas_sorted)


def plot_num_snps_pop(res_dir):
        all_pops, snps_pandas_allpops = get_pandas_num_snps(res_dir)
        for pop in all_pops:
        snps_pop = snps_pandas_allpops[snps_pandas_allpops['pop']==pop]
        pop_plot = sb.catplot(x='chr', y='SNPs', hue=' ',
                        data=snps_pop, kind='bar', aspect=2, height=6)
        pop_plot.fig.suptitle(pop, fontsize=25)
        pop_plot.savefig(pop)


def main():
        vcfs_dir = VCFS_DIR
        test_dir = TEST_DIR
        pop_chr_name = POP_CHR_NAME
        filter = True
        if FILTER == 'no_filter':
        filter = False
        do_kept_sites_file(pop_chr_name, vcfs_dir, filter)
```

```
        plot_kept_sites(pop_chr_name, test_dir, filter)
        obtain_kept_sites_file(vcfs_dir)
        plot_num_snps_pop(test_dir)

if __name__ == '__main__':
        main()
```

Apéndice V

do_vcf_per_population_per_chromosome.py

```
from pathlib import Path
from csv import DictReader
from collections import defaultdict
import subprocess

TOMATO_DIR = Path('/home/marta/devel/tomato')
TRADITOM_DIR = TOMATO_DIR / 'traditom'
VCF_FPATH = TRADITOM_DIR /
'snps/traditom-20180531_onlygbs.tier1.no_bad_samples.marta.vcf.gz'
CLASSIFICATIONS_FPATH = TRADITOM_DIR / 'traditom_classifications.csv'
NO_PERIC_FPATH = TOMATO_DIR / 'no_pericentromeric_regions.bed'


def get_samples_per_pop(classif_fpath):
        class_fhand = classif_fpath.open()
        samples_per_population = defaultdict(list)
        for sample in DictReader(class_fhand):
        sample_name = sample['sample'].rstrip()
        population = sample['rank1']
        if not population:
        continue
        samples_per_population[population].append(sample_name)
        return samples_per_population


def do_vcf_per_pop_per_chr(vcf_fpath, no_peric_fpath, classif_fpath):
        samples_per_pop = get_samples_per_pop(classif_fpath)
        chromosomes_list = [line.strip().split('\t')[0] for line in no_peric_fpath.open()]
        chromosomes_set = set(chromosomes_list)
        for population, samples in samples_per_pop.items():
        samples = ','.join(samples)
        for chromosome in chromosomes_set:
        output_name = population + '_' + chromosome + '.vcf'
        filter_vcf_args =  ['bcftools', 'view', str(vcf_fpath),
                        '-R', str(no_peric_fpath),
                        '-s', samples,
                        '-t', chromosome,
                        '-o', output_name ]
        subprocess.run(filter_vcf_args)

def main():
        vcf_fpath = VCF_FPATH
        classif_fpath = CLASSIFICATIONS_FPATH
        no_peric_fpath = NO_PERIC_FPATH
        do_vcf_per_pop_per_chr(vcf_fpath, no_peric_fpath, classif_fpath)

if __name__ == '__main__':
        main()
```

Apéndice VI

do_pca_per_pop.py

```
from pathlib import Path
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from reorganize_and_plot_results import reorganize_results
from collections import defaultdict
import matplotlib.pyplot as plt


RESULTS_DPATH = Path('/home/marta/devel/diploSHIC/results_stats')
STATS_NAMES = ['pi', 'thetaW', 'tajD', 'distVar', 'distSkew', 'distKurt',
        'nDiplos', 'diplo_H1', 'diplo_H12', 'diplo_H2/H1', 'diplo_ZnS',
        'diplo_Omega']


def turn_data_into_per_pop_matrix(results_dpath, stats_names):
        all_pops = []
        total = 0
        eliminated = 0
        first_pop = True
        stats_matrix_per_pop = {}

        for pop_chr_name, stats_per_subwins in reorganize_results(results_dpath):
        total +=1
        pop_name = pop_chr_name.split('.')[0][:-4]
        if not stats_per_subwins:
        continue
        subwins_array = []
        regions_id_list = []
        for subwin, stats in stats_per_subwins.items():
        subwin_array = []
        mean_subwin = int((int(subwin[0]) + int(subwin[1]))/2)
        region_id = str(mean_subwin)+'_'+pop_chr_name.split('_')[-1]
        regions_id_list.append(region_id)
        for stat_name in stats_names:
                subwin_array.append(stats[stat_name])
        subwins_array.append(subwin_array)
        subwin_stats_matrix = np.matrix(subwins_array)
        stats_matrix_standardized = standardize_matrix_per_column(subwin_stats_matrix)
        #matrix: each row is a region of the chromosome and each column
        #a different statistic

        if np.any(np.isnan(stats_matrix_standardized)):
        eliminated += 1
        continue

        if pop_name not in stats_matrix_per_pop.keys():
        stats_matrix_per_pop[pop_name] = [regions_id_list, stats_matrix_standardized]
        else:
```

```
        stats_matrix_per_pop[pop_name][0] += regions_id_list
        stats_matrix_per_pop[pop_name][1] =
np.append(stats_matrix_per_pop[pop_name][1], stats_matrix_standardized, axis=0)

        for pop in stats_matrix_per_pop:
        regions_id_list = stats_matrix_per_pop[pop][0]
        standardized_pop_matrix = stats_matrix_per_pop[pop][1]
        yield(pop, regions_id_list, standardized_pop_matrix)

        print('TOTAL REGIONS:', total)
        print('ELIMINATED REGIONS:', eliminated, '\n')


def standardize_matrix_per_column(matrix):
        num_columns = matrix.shape[1]
        for idx in range(num_columns):
        column = matrix[:,idx]
        std_column = StandardScaler().fit_transform(column)
        matrix[:,idx] = std_column
        return matrix


def do_pca(standardized_matrix):
        pca = PCA(n_components=3)
        regions_coordinates = pca.fit_transform(standardized_matrix)
        ppal_components = pca.components_
        components_variance = pca.explained_variance_ratio_
        return(regions_coordinates, ppal_components, components_variance)


def write_data_curlywhirly(results_dpath, stats_names):
        header = ['categories:population','categories:chromosome','label','PC01',
        'PC02','PC03']
        header = '\t'.join(header)
        for matrix_data in turn_data_into_per_pop_matrix(results_dpath, stats_names):
        pop_name = matrix_data[0]
        regions_ids = matrix_data[1]
        standardized_pop_matrix = matrix_data[2]
        if 'all' in pop_name:
        continue
        pca_data = do_pca(standardized_pop_matrix)
        pca_file = open(pop_name+'_pcafile', 'w')
        pca_file.write(header + '\n')
        regions_coordinates = pca_data[0]
        for idx in range(len(regions_ids)):
        region_id = regions_ids[idx]
        coordinates = regions_coordinates[idx,:]
        chr_name = region_id.split('_')[-1]
        line = [pop_name, chr_name, region_id, str(coordinates[0]),
                str(coordinates[1]), str(coordinates[2])]
        line = '\t'.join(line)
```

```
        pca_file.write(line+'\n')
        ppal_components, components_variance = pca_data[1], pca_data[2]
        write_pca_report(pop_name, ppal_components, components_variance,
stats_names)
        plot_eigenvalues(pop_name, ppal_components, stats_names)


def write_pca_report(pop_name, ppal_components, components_variance, stats_names):
        report_file = open(pop_name+'_pcareport', 'w')
        variance = ['VARIANCE EXPLAINED:\nprincipal component 1: ',
                str(components_variance[0]), '\nprincipal component 2: ',
                str(components_variance[1]), '\nprincipal component 3: ',
                str(components_variance[2]), '\ntotal: ',
                str(sum(components_variance)), '\n\n\n']
        report_file.write(''.join(variance))
        report_file.write('PRINCIPAL COMPONENTS\n\n')
        for pc_num, ppal_component in enumerate(ppal_components):
        name_and_value = [[stats_names[idx], eigen_value] for idx, eigen_value in
enumerate(ppal_component)]
        name_and_value = sorted(name_and_value, key=lambda x: x[1])
        name_and_value_round = [(tup[0], str(round(tup[1],4))) for tup in name_and_value]
        report_file.write('principal component {} most explained by:\n'.format(pc_num+1))
        most_positive = [' '.join(name_val) for name_val in name_and_value_round[:3]]
        most_negative = [' '.join(name_val) for name_val in
name_and_value_round[-4:-1][-1::-1]]
        report_file.write('\t'.join(most_positive)+'\n')
        report_file.write('\t'.join(most_negative)+'\n\n')


def plot_eigenvalues(pop_name, ppal_components, stats_names):
        plot = plt.subplots(nrows=3,ncols=1, figsize=[10,16])
        fig, axes = plot[0], plot[1]
        num_ppal_comp = [(idx+1,ppal_component) for idx,ppal_component in
enumerate(ppal_components)]
        num_ppal_comp_shifted = num_ppal_comp[1:]
        num_ppal_comp_shifted.append(num_ppal_comp[0])
        for ax_idx, ppal_comp_pair in enumerate(zip(num_ppal_comp,
num_ppal_comp_shifted)):
        for idx, eigen_coordinates in enumerate(zip(ppal_comp_pair[0][1],
ppal_comp_pair[1][1])):
        eigen_name = stats_names[idx]
        axes[ax_idx].plot([0, eigen_coordinates[0]], [0, eigen_coordinates[1]], marker='o')
        axes[ax_idx].set_title('PC_'+str(ppal_comp_pair[1][0])+' vs
PC_'+str(ppal_comp_pair[0][0]))
        axes[ax_idx].text(eigen_coordinates[0], eigen_coordinates[1], eigen_name)
        axes[ax_idx].spines['left'].set_position(('data',0.0))
        axes[ax_idx].spines['bottom'].set_position(('data',0.0))
        fig.suptitle(pop_name, fontsize=40)
        fig.savefig(pop_name+'.png')
```

```
def main():
	results_dpath = RESULTS_DPATH
	stats_names = STATS_NAMES
	write_data_curlywhirly(results_dpath, stats_names)


if __name__ == '__main__':
	main()
```

Apéndice VII

regiones que han sufrido selección identificadas por DiploSHIC

POP	CHR	REGION	STATS
SP	SL2.50ch01	95250000	distSkew,distKurt
SP	SL2.50ch01	97150000	thetaW,distVar
SP	SL2.50ch01	92850000	diplo_Omega
SP	SL2.50ch02	40250000	distVar
SP	SL2.50ch02	44050000	distKurt,distSkew
SP	SL2.50ch02	46550000	diplo_ZnS,diplo_H12,diplo_H1
SP	SL2.50ch03	5050000	nDiplos,pi
SP	SL2.50ch03	67650000	distVar
SP	SL2.50ch05	64150000	diplo_ZnS,diplo_H12,diplo_H1
SP	SL2.50ch09	40250000	diplo_ZnS,diplo_H12,diplo_H1
SP	SL2.50ch09	66250000	diplo_ZnS,diplo_H12,diplo_H1
SP	SL2.50ch10	59950000	distVar
SP	SL2.50ch11	6250000	distVar
cherry1	SL2.50ch01	92050000	pi,distVar
cherry1	SL2.50ch02	45350000	diplo_H1,diplo_H12
cherry1	SL2.50ch02	49850000	diplo_H1,diplo_H12
cherry1	SL2.50ch04	62050000	diplo_H1,diplo_H12
cherry1	SL2.50ch04	66050000	diplo_H2/H1
cherry1	SL2.50ch06	43850000	diplo_H2/H1,nDiplos
cherry1	SL2.50ch06	42550000	tajD
cherry1	SL2.50ch06	42950000	distKurt,distSkew
cherry1	SL2.50ch06	45750000	diplo_H2/H1
cherry1	SL2.50ch06	46050000	diplo_Omega,thetaW,distVar

cherry1	SL2.50ch07	150000	distKurt,distSkew
cherry1	SL2.50ch07	950000	diplo_H2/H1
cherry2	SL2.50ch01	89750000	pi,distVar
cherry2	SL2.50ch01	91650000	distKurt,diploH2/H1
cherry2	SL2.50ch01	92350000	diplo_Omega,distSkew,thetaW
cherry2	SL2.50ch01	96350000	diplo_H1,diplo_H12
cherry2	SL2.50ch02	40050000	distVar,thetaW
cherry2	SL2.50ch02	40150000	tajD
cherry2	SL2.50ch02	47950000	diplo_H1,diplo_H12
cherry2	SL2.50ch02	50750000	pi,distVar
cherry2	SL2.50ch02	51850000	diplo_H1,diplo_H12
cherry2	SL2.50ch04	54750000	distSkew
cherry2	SL2.50ch04	65650000	pi,distVar
cherry2	SL2.50ch05	3250000	distKurt,diploH2/H1
cherry2	SL2.50ch06	46050000	tajD,diplo_ZnS
cherry2	SL2.50ch11	4950000	nDiplos,tajD
green	SL2.50ch03	63250000	distVar,distKurt
green	SL2.50ch03	67650000	diplo_Zns
green	SL2.50ch05	1350000	nDiplos,diplo_H2/H1
green	SL2.50ch05	850000	thetaW,diplo_Omega,pi
green	SL2.50ch07	250000	thetaW,pi,diplo_Omega
green	SL2.50ch07	2950000	diplo_H1,diplo_H12
green	SL2.50ch09	350000	diplo_H1,diplo_H12
green	SL2.50ch09	63150000	diplo_H1,diplo_H12
green	SL2.50ch10	61150000	diplo_H1,diplo_H12
green	SL2.50ch10	63150000	diplo_H1,diplo_H12

green	SL2.50ch11	150000	distVar
modern1	SL2.50ch01	92250000	distKurt,distSkew
modern1	SL2.50ch01	92750000	tajD.diplo_H2/H1
modern1	SL2.50ch01	93650000	nDiplos,diplo_Omega
modern1	SL2.50ch01	96050000	distKurt,distSkew
modern1	SL2.50ch02	41850000	tajD.diplo_H2/H1
modern1	SL2.50ch02	50550000	tajD.diplo_H2/H1
modern1	SL2.50ch02	45350000	distKurt,distSkew
modern1	SL2.50ch03	65350000	diplo_H2/H1
modern1	SL2.50ch05	62550000	diplo_H1,diplo_H2
modern1	SL2.50ch06	650000	diplo_H1,diplo_H2
modern1	SL2.50ch06	1750000	diplo_H1,diplo_H12
modern1	SL2.50ch08	64450000	tajD.diplo_H2/H1
modern1	SL2.50ch10	64350000	nDiplos
modern2	SL2.50ch01	97650000	distSkew,distKurt
modern2	SL2.50ch02	50550000	pi
modern2	SL2.50ch06	40450000	distSkew,distKurt
modern2	SL2.50ch06	46050000	diplo_Omega
modern2	SL2.50ch07	67050000	tajD,diplo_H2/H1
modern2	SL2.50ch09	65950000	tajD,diplo_H2/H1
modern2	SL2.50ch09	1450000	diplo_Omega
SLC_Ecu	SL2.50ch01	84150000	pi,nDiplos
SLC_Ecu	SL2.50ch01	85450000	nDiplos
SLC_Ecu	SL2.50ch02	48650000	diplo_Zns
SLC_or_SLL_mex	SL2.50ch05	65750000	distVar,thetaW,diplo_Omega

SLC_Peru	SL2.50ch01	92250000	diplo_H2/H1,tajD
SLC_Peru	SL2.50ch01	92350000	diplo_H2/H1,tajD
SLC_Peru	SL2.50ch01	98450000	diplo_H1,diplo_H12
SLC_Peru	SL2.50ch03	53250000	distVar,thetaW,diplo_Omega
SLC_Peru	SL2.50ch02	40050000	distVar,thetaW,diplo_Omega
SLC_Peru	SL2.50ch06	35650000	pi,distVar
SLC_Peru	SL2.50ch06	39450000	nDiplos,distKurt
SLC_Peru	SL2.50ch06	44150000	diplo_H1,diplo_H13
SLC_Peru	SL2.50ch06	45550000	diplo_H1,diplo_H14
SLC_Peru	SL2.50ch09	68050000	diplo_H1,diplo_H15
SLC_Peru	SL2.50ch11	55350000	diplo_H1,diplo_H16
SP	SL2.50ch01	95250000	distSkew
SP	SL2.50ch01	97150000	distVar,thetaW,diplo_Omega
SP	SL2.50ch02	40250000	diplo_H1,diplo_H12
SP	SL2.50ch02	44050000	distKurt
SP	SL2.50ch05	64150000	diplo_H1,diplo_H12
SP	SL2.50ch09	66250000	diplo_H1,diplo_H12
SP_Necu	SL2.50ch02	40050000	pi,thetaW,diplo_Omega
SP_Necu	SL2.50ch03	53050000	distVar,thetaW,diplo_Omega,diplo_Zns
SP_Necu	SL2.50ch03	53250000	tajD
SP_Necu	SL2.50ch09	64050000	distVar,thetaW,diplo_Omega,diplo_Zns
SP_Necu	SL2.50ch11	5550000	pi,nDiplos
SP_Necu	SL2.50ch12	64250000	distVar,thetaW,diplo_Omega,diplo_Zns
SpxSLL	SL2.50ch01	550000	diplo_H1,diplo_H12
SpxSLL	SL2.50ch04	5250000	distVar,thetaW,diplo_Omega,diplo_Zns
SpxSLL	SL2.50ch07	60050000	distVar,thetaW,diplo_Omega,diplo_Zns

SpxSLL	SL2.50ch12	61950000	pi,nDiplos
control2	SL2.50ch04	2050000	diplo_H1,diplo_H12,tajD
control2	SL2.50ch07	60250000	pi,nDiplos,diplo_H2/H1
control2	SL2.50ch07	67450000	diplo_H1,diplo_H12,distSkew
control2	SL2.50ch11	5050000	diplo_H2/H1,tajD
control2	SL2.50ch12	4450000	pi,nDiplos
out_core_b	SL2.50ch01	79850000	pi
out_core_b	SL2.50ch02	47550000	diplo_H1,diplo_H12
out_core_b	SL2.50ch03	62150000	distVar,thetaW,diplo_Omega,diplo_Zns
out_core_b	SL2.50ch09	1450000	diplo_Zns
out_core_b	SL2.50ch11	3550000	tajD
out_core_c	SL2.50ch03	65350000	diplo_H2/H1
out_core_d	SL2.50ch01	89550000	diplo_H2/H1,nDiplos
out_core_d	SL2.50ch01	90550000	distVar,thetaW,diplo_Omega,diplo_Zns
out_core_d	SL2.50ch04	2950000	diplo_H2/H1
out_core_d	SL2.50ch04	66050000	nDiplos,diplo_H2/H1
out_core_d	SL2.50ch06	40850000	distVar,thetaW,diplo_Omega,diplo_Zns
out_core_d	SL2.50ch10	63550000	distVar,thetaW,diplo_Omega,diplo_Zns
out_core_f	SL2.50ch03	65350000	pi,nDiplos
out_core_f	SL2.50ch12	4450000	nDiplos,diplo_H2/H1
out_core_i	SL2.50ch06	40550000	diplo_H2/H1
out_core_j	SL2.50ch07	60250000	pi,nDiplos
out_core_j	SL2.50ch11	55150000	diplo_H2/H1
out_core_j	SL2.50ch06	41450000	distVar
out_core_k	SL2.50ch02	40550000	diplo_H1,diplo_H12,distSkew
out_core_k	SL2.50ch02	46850000	diplo_H2/H1,distKurt

processing	SL2.50ch04	4150000	distSkew,distKurt
processing	SL2.50ch04	61950000	diplo_Zns
processing	SL2.50ch10	1650000	diplo_H2/H1,tajD

Apéndice VIII

run_sweed_pops.py

```
from pathlib import Path
import subprocess

POP_VCFS_DIR =
Path('/home/jope/tomato/population_traditom/snps/tfg_Marta/BayeScan_test/pop_vcfs')
SWEED_PATH = Path('/home/jope/soft/sweed/SweeD-P')


def run_sweed(pop_vcfs_dir, sweed_path):
        for pop_vcf_fpath in pop_vcfs_dir.iterdir():
        pop_name = str(pop_vcf_fpath).split('/')[-1].split('.')[0]
        sweed_cmd = [str(sweed_path),
                '-name', pop_name,
                '-input', str(pop_vcf_fpath),
                '-grid', '100',
                '-folded']
        print('running SweeD for ', pop_name)
        subprocess.run(sweed_cmd)

def main():
        pop_vcfs_dir = POP_VCFS_DIR
        sweed_path = SWEED_PATH
        run_sweed(pop_vcfs_dir, sweed_path)


if __name__ == '__main__':
        main()
```

Apéndice IX

do_vcf_per_pop.py

```
from pathlib import Path
from csv import DictReader
from collections import defaultdict
import subprocess

TOMATO_DIR = Path('/home/marta/devel/tomato')
TRADITOM_DIR = TOMATO_DIR / 'traditom'
VCF_FPATH = TRADITOM_DIR /
'snps/traditom-20180531_onlygbs.tier1.no_bad_samples.marta.vcf.gz'
CLASSIFICATIONS_FPATH = TRADITOM_DIR / 'traditom_classifications.csv'
NO_PERIC_FPATH = TOMATO_DIR / 'no_pericentromeric_regions.bed'



def get_samples_per_pop(classif_fpath):
        class_fhand = classif_fpath.open()
        samples_per_population = defaultdict(list)
        for sample in DictReader(class_fhand):
        sample_name = sample['sample'].rstrip()
        population = sample['rank1']
        if not population:
        continue
        samples_per_population[population].append(sample_name)
        return samples_per_population


def do_vcf_per_pop(vcf_fpath, classif_fpath):
        samples_per_pop = get_samples_per_pop(classif_fpath)
        for population, samples in samples_per_pop.items():
        samples = ','.join(samples)
        output_name = population + '_' + chromosome + '.vcf'
        filter_vcf_args =  ['bcftools', 'view', str(vcf_fpath),
                        '-s', samples,
                        '-o', output_name ]
        subprocess.run(filter_vcf_args)


def main():
        vcf_fpath = VCF_FPATH
        classif_fpath = CLASSIFICATIONS_FPATH
        do_vcf_per_pop(vcf_fpath, classif_fpath)

if __name__ == '__main__':
        main()
```

Apéndice X

plot_sweed_results.py

```
import matplotlib
matplotlib.use('Agg')
from pathlib import Path
import matplotlib.pyplot as plt


RESULTS_DIR =
Path('/home/jope/tomato/population_traditom/snps/tfg_Marta/sweed_test/results')

def plot_sweed_results(results_dir):
	cl_ratio_all_pops = []
	for pop_result_fdir in results_dir.iterdir():
	if "Info" in str(pop_result_fdir):
	continue
	pop = str(pop_result_fdir).split('/')[-1].split('.')[-1]
	x_values, y_values = [], []
	first_chr = True
	for line in pop_result_fdir.open():
	if not line.strip():
		continue
	if line.strip().startswith('Position'):
		continue
	if line.startswith('//') and first_chr:
		chr = '00'
		first_chr = False
		continue
	if line.startswith('//'):
		plt.plot(x_values, y_values, 'o')
		plt.xlabel('chromosome region')
		plt.ylabel('likelihood')
		plt.suptitle(pop+' chromosome'+chr)
		plt.savefig(pop+'_SL2.50ch'+chr+'.png')
		plt.close()
		x_values, y_values = [], []
		alignment = int(line.strip()[2:])
		chr = str(alignment-1)
		if len(chr) == 1:
		chr = '0'+chr
		continue
	values = line.strip().split()
	if len(values) == 3:
		chr_region, cl_ratio = float(values[0]), float(values[1])
		cl_ratio_all_pops.append(cl_ratio)
		x_values.append(chr_region)
		y_values.append(cl_ratio)
	plt.plot(x_values, y_values, 'o')
	plt.xlabel('chromosome region')
	plt.ylabel('likelihood')
	plt.suptitle(pop+' chromosome'+chr)
```

```
        plt.savefig(pop+'_SL2.50ch'+chr+'.png')
        plt.close()


def main():
        results_dir = RESULTS_DIR
        plot_sweed_results(results_dir)

if __name__ == '__main__':
        main()
```

Apéndice XI

regiones que han sufrido selección identificadas por SweeD

POP	CHR	REGION
cherry1	SL2.50ch01	80000000
cherry1	SL2.50ch06	13000000
cherry1	SL2.50ch06	50000000
cherry1	SL2.50ch07	5000000
cherry1	SL2.50ch10	60000000
cherry1	SL2.50ch11	2000000
cherry1	SL2.50ch11	30000000
cherry1	SL2.50ch12	2000000
cherry1	SL2.50ch12	65000000
cherry2	SL2.50ch12	2000000
control3	SL2.50ch06	10000000
control3	SL2.50ch09	20000000
modern1	SL2.50ch01	80000000
modern1	SL2.50ch04	60000000
modern1	SL2.50ch05	10000000
modern1	SL2.50ch06	45000000
modern1	SL2.50ch08	10000000
modern1	SL2.50ch08	60000000
modern1	SL2.50ch09	2000000
modern1	SL2.50ch11	2000000
modern1	SL2.50ch12	2000000
modern1	SL2.50ch12	65000000
modern2	SL2.50ch02	39000000

modern2	SL2.50ch02	53000000
modern2	SL2.50ch03	70000000
modern2	SL2.50ch05	5000000
modern2	SL2.50ch05	65000000
modern2	SL2.50ch06	10000000
modern2	SL2.50ch06	39000000
modern2	SL2.50ch06	48000000
modern2	SL2.50ch08	2000000
modern2	SL2.50ch08	55000000
modern2	SL2.50ch08	65000000
modern2	SL2.50ch09	2000000
modern2	SL2.50ch09	22000000
modern2	SL2.50ch09	69000000
out_core_a	SL2.50ch05	2000000
out_core_a	SL2.50ch05	6000000
out_core_a	SL2.50ch05	60000000
out_core_a	SL2.50ch12	65000000
out_core_b	SL2.50ch01	90000000
out_core_b	SL2.50ch02	45000000
out_core_b	SL2.50ch02	55000000
out_core_b	SL2.50ch05	2000000
out_core_b	SL2.50ch05	22000000
out_core_b	SL2.50ch10	2000000
out_core_b	SL2.50ch10	62000000
out_core_b	SL2.50ch11	65000000
out_core_c	SL2.50ch04	5000000

out_core_c	SL2.50ch04	60000000
out_core_c	SL2.50ch07	2000000
out_core_c	SL2.50ch07	59000000
out_core_c	SL2.50ch07	69000000
out_core_c	SL2.50ch11	2000000
out_core_c	SL2.50ch11	50000000
out_core_d	SL2.50ch06	12000000
out_core_d	SL2.50ch10	5000000
out_core_d	SL2.50ch11	2000000
out_core_d	SL2.50ch11	55000000
out_core_f	SL2.50ch08	55000000
out_core_f	SL2.50ch08	65000000
out_core_f	SL2.50ch12	2000000
out_core_f	SL2.50ch12	39000000
out_core_f	SL2.50ch12	65000000
out_core_k	SL2.50ch06	10000000
out_core_k	SL2.50ch11	2000000
out_core_k	SL2.50ch11	25000000
processing	SL2.50ch04	5000000
processing	SL2.50ch04	60000000
processing	SL2.50ch05	65000000
processing	SL2.50ch06	10000000
processing	SL2.50ch06	38000000
processing	SL2.50ch06	49000000
processing	SL2.50ch11	52000000
SLC_Ecu	SL2.50ch03	5000000

SLC_Ecu	SL2.50ch03	59000000
SLC_Ecu	SL2.50ch03	70000000
SLC_Ecu	SL2.50ch06	39000000
SLC_Ecu	SL2.50ch06	50000000
SLC_Ecu	SL2.50ch07	2000000
SLC_Ecu	SL2.50ch07	62000000
SLC_Ecu	SL2.50ch09	2000000
SLC_Ecu	SL2.50ch09	65000000
SLC_Ecu	SL2.50ch10	60000000
SLC_Ecu	SL2.50ch11	6000000
SLC_Ecu	SL2.50ch11	55000000
SLC_or_SLL_mex	SL2.50ch08	58000000
SLC_Peru	SL2.50ch04	2000000
SLC_Peru	SL2.50ch08	65000000
SLC_Peru	SL2.50ch11	40000000
SP_Necu	SL2.50ch05	32000000
SP	SL2.50ch02	25000000
SPxSLL	SL2.50ch11	18000000

Apéndice XII

run_raisd.py

```
from pathlib import Path
import subprocess
from collections import defaultdict
import tempfile as tmf

VCF_FPATH =
Path('/home/marta/devel/tomato/traditom-20180531_onlygbs.tier1.no_bad_samples.marta.v
cf')
SAMP_TO_POP_FPATH = Path('/home/marta/devel/diploSHIC/sample_to_pop.txt')


def get_samples_per_pop(samp_to_pop_fpath):
        samp_to_pop_fhand = samp_to_pop_fpath.open()
        samp_per_pop_dic = defaultdict(list)
        for line in samp_to_pop_fhand:
        samp, pop = line.strip().split()
        samp_per_pop_dic[pop].append(samp)
        return samp_per_pop_dic


def run_raisd(vcf_fpath, samp_per_pop_dic):
        for pop, samples in samp_per_pop_dic.items():
        samp_file_fhand = tmf.NamedTemporaryFile('w')
        for samp in samples:
        samp_file_fhand.write(samp+'\n')
        samp_file_fhand.flush()
        samp_fpath = samp_file_fhand.name
        raisd_args = ['RAiSD', '-n', pop, '-I', str(vcf_fpath),
                '-S', samp_fpath, '-P', '-f']
        raisd_command = ' '.join(raisd_args)
        subprocess.run(raisd_args)


def main():
        samp_to_pop_fpath = SAMP_TO_POP_FPATH
        samp_per_pop_dic = get_samples_per_pop(samp_to_pop_fpath)
        vcf_fpath = VCF_FPATH
        run_raisd(vcf_fpath, samp_per_pop_dic)


if __name__ == '__main__':
        main()
```

Apéndice XIII

regiones que han sufrido selección identificadas por RaisD

POP	CHR	REGION
cherry1	SL2.50ch06	30000000
control3	SL2.50ch02	28000000
GAL	SL2.50ch08	40000000
green	SL2.50ch06	12000000
green	SL2.50ch09	50000000
modern1	SL2.50ch06	18000000
modern1	SL2.50ch08	50000000
out_core_b	SL2.50ch08	25000000
out_core_d	SL2.50ch06	34000000
out_core_f	SL2.50ch10	45000000
out_core_k	SL2.50ch06	39000000
SLC_Ecu	SL2.50ch03	55000000
SLC_Ecu	SL2.50ch06	35000000
SLC_Peru	SL2.50ch04	50000000
SLC_Peru	SL2.50ch08	5000000

Apéndice XIV

vcf_to_bayescan_format.py

```
from pathlib import Path
import subprocess

POP_VCFS_DIR = Path('/home/marta/devel/BayeScan_test/pop_vcfs')
POP_GESTES_DIR = Path('/home/marta/devel/BayeScan_test/pop_gestes')


def run_PGDSpider(pop_vcfs_dir):
	for pop_vcf_fpath in pop_vcfs_dir.iterdir():
	pop_name = str(pop_vcf_fpath).split('/')[-1].split('.')[0]
	pgd_command = ['java', '-Xmx3000m', '-Xms512m',
		'-jar', '/home/marta/devel/PGDSpider_2.1.1.5/PGDSpider2-cli.jar',
		'-inputfile', str(pop_vcf_fpath),
		'-inputformat', 'VCF',
		'-outputfile', pop_name + '.txt',
		'-outputformat', 'GENEPOP',
		'-spid', '/home/marta/devel/PGDSpider_2.1.1.5/vct_to_geste_pop.spid']
	print(' '.join(pgd_command))
	subprocess.run(pgd_command)


def re_write_file(gestes_dir):
	all_pops_geste = open('all_pops.txt', 'w')
	first_iteration = True
	pop_gestes = list(gestes_dir.glob('*'))
	num_total_pops = len(pop_gestes)
	pop_num = 1
	for pop_geste in pop_gestes:
	print(pop_geste)
	first_line, second_line, third_line = True, True, True

	if not first_iteration:
	all_pops_geste.write('\n')

	for line in pop_geste.open():
	line = line.strip()

	if not line:
		continue

	if first_iteration and first_line:
		first_line = False
		all_pops_geste.write(line+'\n'+'\n')
		continue

	if first_iteration and second_line:
		first_iteration = False
		second_line = False
		all_pops_geste.write('[populations]={}\n\n'.format(num_total_pops))
```

```
                continue

        if first_line:
                first_line = False
                continue

        if second_line:
                second_line = False
                continue

        if third_line:
                third_line = False
                all_pops_geste.write('[pop]={}\n'.format(pop_num))
                print('[pop]={}\n'.format(pop_num))
                pop_num +=1
                continue
        all_pops_geste.write(line+'\n')


def main():
        pop_vcfs_dir = POP_VCFS_DIR
        gestes_dir = POP_GESTES_DIR
        #run_PGDSpider(pop_vcfs_dir)
        re_write_file(gestes_dir)

if __name__ == '__main__':
        main()
```

Apéndice XV

vcf2genepop_pgdspider.py

```
from pathlib import Path
import subprocess

POP_VCFS_DIR =
Path('/home/jope/tomato/population_traditom/snps/tfg_Marta/BayeScan_test/pop_vcfs')
POP_GENEPOP_DIR =
Path('/home/jope/tomato/population_traditom/snps/tfg_Marta/lositan_test/genepops')


def run_PGDSpider(pop_vcfs_dir):
        for pop_vcf_fpath in pop_vcfs_dir.iterdir():
        pop_name = str(pop_vcf_fpath).split('/')[-1].split('.')[0]
        pgd_command = ['java', '-Xmx10000m', '-Xms512m',
                '-jar', '/home/jope/soft/PGDSpider_2.1.1.5/PGDSpider2-cli.jar',
                '-inputfile', str(pop_vcf_fpath),
                '-inputformat', 'VCF',
                '-outputfile', pop_name + '.txt',
                '-outputformat', 'GENEPOP',
                '-spid',
'/home/jope/tomato/population_traditom/snps/tfg_Marta/vcf_to_genepop_pop.spid']
        print(' '.join(pgd_command))
        subprocess.run(pgd_command)


def re_write_genepop_file(genepops_dir):
        all_genepops = open('all_genepops.txt', 'w')
        pop_to_num = open('pop_to_num.txt', 'w')
        num = 0
        first_iteration = True
        for genepop_fpath in genepops_dir.iterdir():
        pop_name = str(genepop_fpath).split('/')[-1].split('.')[0]
        print(pop_name)
        pop_to_num.write(str(num)+'\t'+pop_name+'\n')
        num+=1
        first_line = True
        for line in genepop_fpath.open():
        if first_line and first_iteration:
                all_genepops.write(line)
                first_line = False
                continue
        if first_iteration and 'SNP' in line:
                all_genepops.write(line)
                continue
        if first_iteration and 'SNP' not in line:
                first_iteration = False
        if first_line:
                first_line = False
                continue
        if not first_iteration and ('SNP' in line or not line.strip()):
```

```
                continue
        all_genepops.write(line)


def main():
        pop_vcfs_dir = POP_VCFS_DIR
        genepops_dir = POP_GENEPOP_DIR
        run_PGDSpider(pop_vcfs_dir)
        re_write_genepop_file(genepops_dir)

if __name__ == '__main__':
        main()
```

Apéndice XVI

compare_selected_regions.py

```
from pathlib import Path
from csv import DictReader
from collections import defaultdict

SWEED_REGIONS_FPATH =
Path('/home/marta/TFG_bueno/sweed_test/outlier_regions.csv')
DSHIC_REGIONS_FPATH =
Path('/home/marta/TFG_bueno/diploSHIC/outlier_regions_without_stats.csv')
RAISD_REGIONS_FPATH =
Path('/home/marta/TFG_bueno/raisd_test/rais_results/outlier_regions.csv')


def selected_regions_dic(csv_regions_fpath, error):
        selected_regions = defaultdict(dict)
        for selected_region in DictReader(csv_regions_fpath.open()):
        pop,chr,region =
selected_region['POP'],selected_region['CHR'],int(selected_region['REGION'])
        selected_region_win = (region-error, region+error )
        if chr not in selected_regions[pop]:
        selected_regions[pop][chr] = [selected_region_win]
        else:
        selected_regions[pop][chr].append(selected_region_win)
        return selected_regions
        #dictionary with {popA:{chr1:[win,win],chr2:[win]},
        #                 popB:{...}
        #                 }


def region_in_ranges(region,ranges):
#region: one point (int), ranges: list of ranges (tuples) i.e. [(2,5),(12,19)]
        region_in_range = False
        ranges.sort(key=lambda t:t[0])
        for range in ranges:
        if region < range[0]:
        break
        if region > range[1]:
        continue
        region_in_range = True
        return region_in_range


def compare_selected_regions(sweed_regions_fpath, dshic_regions_fpath):
        sweed_range_regions_dic = selected_regions_dic(sweed_regions_fpath, 5000000)
        for selected_region in DictReader(dshic_regions_fpath.open()):
        pop, chr, region = selected_region['POP'],
selected_region['CHR'],int(selected_region['REGION'])
        if chr not in sweed_range_regions_dic[pop].keys():
        continue
```

```
        sweed_ranges = sweed_range_regions_dic[pop][chr]
        selected_in_both = region_in_ranges(region,sweed_ranges)
        if selected_in_both:
        print(pop, chr, region)


def main():
        sweed_regions_fpath = SWEED_REGIONS_FPATH
        dshic_regions_fpath = DSHIC_REGIONS_FPATH
        raisd_regions_fpath = RAISD_REGIONS_FPATH
        compare_selected_regions(dshic_regions_fpath, raisd_regions_fpath)

if __name__ == '__main__':
        main()
```