Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo Fin de Grado

Estimación de la ascendencia local mediante técnicas aproximadas

Local ancestry inference using approximate methods

Daute Rodríguez Rodríguez

La Laguna, 10 de junio de 2019

Dra. D. **Coromoto León**, con N.I.F. 78605216W, Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos del Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna

Dr. D. **Carlos Segura**, con N.I.F. 78704244S, Investigador Titular "A", SNI: Nivel I del Área de Ciencias de la Computación del Centro de Investigación en Matemáticas, A.C. de Guanajuato, México

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Estimación de la ascendencia local mediante técnicas aproximadas*"

ha sido realizada bajo su dirección por D. **Daute Rodríguez Rodríguez**, con N.I.F. 79151574H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2019

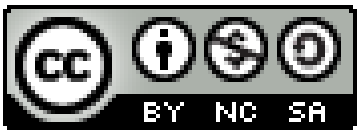
Agradecimientos

A los integrantes del Departamento de Genética de la Universidad de La Laguna por presentarme un problema tan llamativo y apasionante.

A las personas de mi entorno por apoyarme y ayudarme durante el desarrollo del trabajo.

A mis tutores Coromoto León y Carlos Segura, quienes me guiaron, aconsejaron y enseñaron tanto.

Licencia



Resumen

La estimación de la ascendencia local da a conocer aspectos biológicos de gran importancia, como la variación fenotípica o la proclividad hacia determinada enfermedad, que han ido adquiriendo los individuos de una población híbrida a causa de la mezcla genética a lo largo de las generaciones. La estimación consiste en determinar, para cada gen, cuál es el tipo de población de la que proviene. Loter es un paquete software que formula el problema de la estimación de la ascendencia local como un problema de optimización y lo resuelve haciendo uso de una técnica exacta (programación dinámica). El objetivo de este proyecto consiste en desarrollar técnicas aproximadas (computación evolutiva) que resuelvan el problema siguiendo la formulación propuesta en Loter, de manera que se obtengan soluciones de calidad evitando las limitaciones que surgen al usar técnicas exactas. Se han realizado numerosas pruebas para comparar los métodos implementados y determinar cuál de ellos se comporta mejor resolviendo el problema.

Palabras clave: ascendencia local, algoritmos evolutivos, optimización, técnicas aproximadas

Abstract

Local ancestry inference gives information about important biological aspects that individuals of an admixed population acquire during generations such as phenotypic variation and susceptibility to certain diseases. Local ancestry inference consists in, for each gene, establish the type of ancestral source population. Loter is a software package that states the problem of the local ancestry inference as an optimization problem and resolves it using an exact method (dynamic programming). The goal of this project lies in develop aproximate methods (evolutionary computation) to solve the problem following the formulation proposed in Loter and obtaining high quality solutions avoiding the limitations of exact methods. Several tests have been made to compare the different implemented methods and establish which one obtains best solutions.

Keywords: *local ancestry, evolutionary algorithms, optimization, approximate methods*

Índice general

Índice de figuras

Índice de cuadros

Índice de algoritmos

Capítulo 1

Introducción

1.1. Motivación

La **ascendencia local** se define como la ascendencia genética de un individuo en una zona concreta de sus cromosomas (*locus*), donde el individuo puede tener 0, 1 o 2 copias de un *alelo* derivado de una población ancestral [18], siendo un *alelo* las diferentes formas alternativas que puede tener un mismo gen [20]. Estimar la ascendencia local consiste en determinar, dado un *locus* concreto, la cantidad de copias que se originan a partir de cada una de las poblaciones ancestrales de origen.

Conocer la ascendencia local aporta información sobre diversos aspectos biológicos que van adquiriendo los individuos de una población híbrida a causa de la *mezcla genética* a lo largo de las generaciones. La *mezcla genética* se define como el proceso por el cual se origina una nueva población híbrida mediante la mezcla de dos o más poblaciones ancestrales que habían estado previamente aisladas [8].

A más alto nivel, la estimación de la ascendencia local supone identificar la población o poblaciones de origen de un individuo a partir de los datos disponibles sobre las poblaciones que se presupone que han participado en la generación de la población a la que pertenece dicho individuo. En la actualidad, se hace uso de marcadores genéticos que permiten obtener la ascendencia de un individuo de manera fiable y con una alta precisión [13].

Dado que los aspectos o características biológicas que aporta el conocer la ascendencia local de un individuo híbrido son de gran importancia, la estimación de la ascendencia local tiene numerosas aplicaciones. La más importante consiste en **determinar la relación existente entre genes y enfermedades**, identificando aquellos *loci* que son susceptibles a las mismas de entre el material genético de los individuos pertenecientes a las poblaciones que participan en el proceso de la *mezcla genética* [5, 13].

De entre las distintas aplicaciones que se le dan a la estimación de la ascendencia local, cabría destacar las siguientes:

- Farmacogenómica: estudio que analiza la composición genética de un individuo para establecer cómo afectará a su respuesta a los fármacos [3]
- Localización de secuencias en el *genoma humano* [4]
- Estudio de la *variación fenotípica* y la *adaptación biológica* [19]
- Estimación de la selección natural [7]

- Estimación demográfica [9]

Todas estas posibles aplicaciones están relacionadas con ámbitos de investigación sumamente relevantes, que podrían ampliar significativamente nuestro conocimiento sobre la naturaleza y mejorar la forma de vida de la raza humana. En concreto, la aplicación de la estimación de la ascendencia local en la farmacogenómica y en la búsqueda de la relación existente entre los genes y las enfermedades suponen un gran avance dentro del ámbito de la **medicina personalizada**.

1.2. Antecedentes

Hasta el momento, la estimación de la ascendencia local se ha abordado mayoritariamente mediante el uso de modelos estadísticos (Tabla 1 del artículo *Models, methods and tools for ancestry inference and admixture analysis* [21]). A pesar de este hecho, en Junio de 2018 un grupo de investigadores franceses publicó *Loter: A Software Package to Infer Local Ancestry for a Wide Range of Species* [2], artículo en el que se formula el problema de la estimación de la ascendencia local como un **problema de optimización**. Los autores de *Loter* abordaron el problema haciendo uso de técnicas exactas, concretamente, mediante programación dinámica.

Para llevar a cabo la estimación, *Loter* requiere información acerca de los *haplotipos* de los individuos de cada una de las poblaciones implicadas, cada individuo aporta dos haplotipos debido al carácter diploide de las células humanas. Un *haplotipo* es un conjunto de *polimorfismos de nucleótido único* (*Single-nucleotide polymorphism,* ***SNP***) que están estadísticamente asociados y tienden a heredarse juntos.

Un *SNP* es una variación que afecta a una sola base (*Adenina (A), Timina (T), Citosina (C) o Guanina (G)*) de una secuencia del *genoma* (ya sea una pequeña porción o un cromosoma entero, véase la Figura 1.1). Atendiendo a una posición específica, cada variación posible es un alelo del SNP, el alelo más frecuente se denomina *alelo mayor* y el menos frecuente *alelo menor*. Para que una variación se considere SNP ha de estar presente en al menos un 1 % de los individuos de una población.

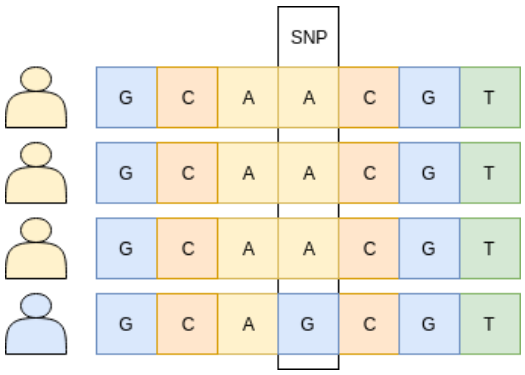


Figura 1.1: En el diagrama se presenta un ejemplo de SNP, tres de los cuatro individuos tienen en la cuarta posición la base A, mientras que el último tiene la base G. Cada variación posible (A o G) es un alelo del SNP, y en este caso el *alelo mayor* sería el relacionado con la base A, pues es más frecuente.

Cada elemento de un haplotipo contiene información acerca de las variaciones (alelos) del SNP que representa. Un haplotipo aporta información acerca de una única secuencia del genoma. Éstos se codifican como secuencias de $\{0, 1\}$, el valor 0 indica que el SNP se corresponde

con el *alelo mayor* y el 1 con el *alelo menor*. Los haplotipos se obtienen generalmente al aplicar algoritmos de estimación o "*phasing*" sobre el *genotipo*.

El *genotipo* de un individuo es la composición genética de una célula que determina una de sus características. Se asocia a los dos haplotipos de un individuo, es decir, representa dos secuencias homólogas del genoma de dicho individuo. Cada elemento del genotipo almacena información acerca de las dos variaciones (alelos) de los SNP correspondientes (uno por cada haplotipo). Se codifica mediante una secuencia de valores $\{0, 1, 2\}$, el valor 0 indica que ambos alelos se corresponden con el *alelo mayor*, el valor 1, que ambos alelos se corresponden con el *alelo menor* y el valor 2 indica que un alelo se corresponde con el *alelo mayor* y el otro con el *alelo menor*.

En la Figura 1.2 se representa un ejemplo para simplificar la relación existente entre el genotipo y los haplotipos así como el significado de los posibles valores que pueden albergar.

Figura 1.2: Ejemplificación de la relación existente entre los haplotipos y el genotipo

Una de la particularidades y puntos fuertes de *Loter* reside en el hecho de no necesitar ningún parámetro de carácter biológico o estadístico para poder llevar a cabo la resolución del problema. Esto supone una gran ventaja frente a otros paquetes software que resuelven el problema de la estimación de la ascendencia local (*HAPMIX*[14], *RFMix*[11] y *LAMP-LD*[1]) y que sí que requieren parámetros de carácter biológico o estadístico, ya que por norma general es costoso obtener dicha información. A pesar de requerir menos parámetros, atendiendo a los experimentos que se presentan en [2], se puede apreciar que *Loter* se comporta mejor en diversos aspectos que los otros paquetes software resolviendo el problema. El hecho de que en *Loter* se haya formulado la estimación de la ascendencia local como un problema de optimización, hace posible que aparezcan **nuevas aproximaciones e intentos de resolución** haciendo uso de otro tipo de técnicas, como en el presente trabajo.

1.3. Objetivo

El objetivo de este Trabajo Fin de Grado (TFG) consiste en llevar a cabo el **desarrollo de técnicas aproximadas** que sean capaces de resolver el problema de la estimación de la ascendencia local obteniendo soluciones de calidad. En concreto, se abordará el problema mediante **computación evolutiva** siguiendo la formulación del problema propuesta en *Loter* [2].

A pesar de que las técnicas exactas devuelven siempre la solución óptima al problema, presentan una serie de inconvenientes poco deseables. Para tamaños de instancia del problema lo

suficientemente grandes, el tiempo y esfuerzo computacional se vuelven intratables. Precisamente en los problemas de optimización relacionados con la genómica el tamaño de los datos de los que se dispone, o equivalentemente, el espacio de soluciones del problema tienen una gran magnitud, por lo que para poder obtener resultados haciendo uso de este tipo de métodos habría que reducir el tamaño de los datos de entrada, pudiendo provocar una alteración del significado o conclusiones que se extraen de las soluciones al problema abordado devueltas por los métodos. Por ello, aportar una **alternativa** al método de programación dinámica propuesto en *Loter* [2] haciendo uso de la formulación del problema que presentan resulta tan prometedor y llamativo.

1.4. Plan de trabajo

A continuación, se presenta el plan de trabajo seguido durante el desarrollo de este proyecto:

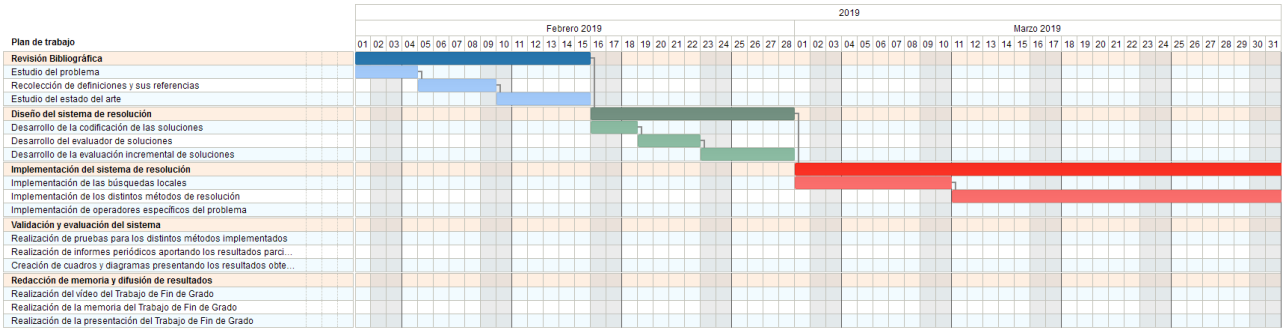


Figura 1.3: Primera parte del plan de trabajo

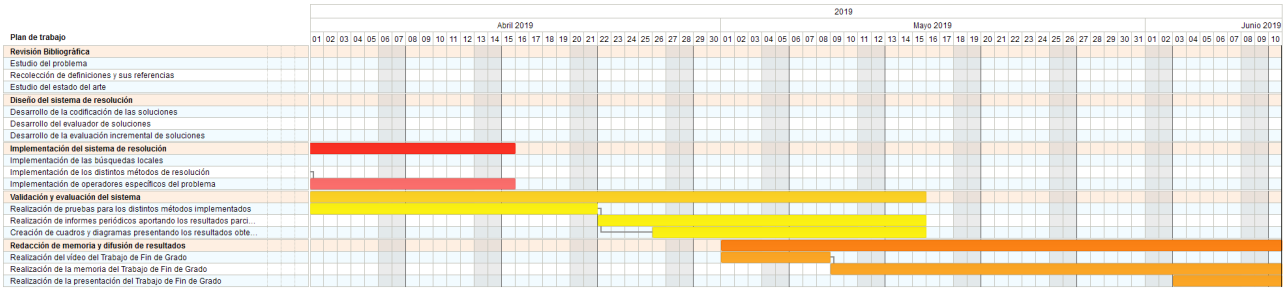


Figura 1.4: Segunda parte del plan de trabajo

1. Revisión bibliográfica: 01/02/2019 - 15/02/2019
    - Estudio del problema
    - Recolección de definiciones y sus referencias
    - Estudio del estado del arte
2. Diseño del sistema de resolución: 16/02/2019 - 28/02/2019
    - Desarrollo de la codificación de las soluciones
    - Desarrollo del evaluador de soluciones
    - Desarrollo de la evaluación incremental de soluciones

3. Implementación del sistema de resolución: 01/03/2019 - 15/04/2019
    - Implementación de las búsquedas locales
    - Implementación de los distintos métodos de resolución
    - Implementación de operadores específicos del problema
4. Validación y evaluación del sistema: 01/04/2019 - 15/05/2019
    - Realización de pruebas para los distintos métodos implementados
    - Realización de informes periódicos aportando los resultados parciales
    - Creación de cuadros y diagramas presentando los resultados obtenidos
5. Redacción de la memoria y difusión de los resultados: 01/05/2019 - 10/06/2019
    - Realización del vídeo del Trabajo de Fin de Grado
    - Realización de la memoria del Trabajo de Fin de Grado
    - Realización de la presentación del Trabajo de Fin de Grado

El resto de los contenidos de esta memoria de TFG se presentan siguiendo la estructura que se define a continuación. En el Capítulo 2 se establece la formulación del problema propuesta, tanto la codificación de las soluciones como la función objetivo. La implementación desarrollada se presenta en el Capítulo 3. En el Capítulo 4 se aglutinan los experimentos llevados a cabo y los resultados obtenidos al abordar el problema haciendo uso de las técnicas desarrolladas. En los Capítulos 5 y 6 se presentan las conclusiones extraídas (en los idiomas español e inglés respectivamente) de los resultados obtenidos en el estudio experimental. Por último, en el Capítulo 7 se presenta un presupuesto estimado para el trabajo llevado a cabo durante el desarrollo de este proyecto.

Capítulo 2

Formulación del problema

Siguiendo la formulación del problema propuesta en *Loter* [2], se espera un conjunto de haplotipos de individuos pertenecientes a las poblaciones ancestrales H y un haplotipo híbrido h, de un individuo perteneciente a la población híbrida, es decir, la población que se asume que se ha originado a partir de la unión de las poblaciones ancestrales. El total de los individuos pertenecientes a las poblaciones ancestrales es n y por consiguiente se dispondrá de $2n$ haplotipos ancestrales. Cabe destacar que tanto los haplotipos ancestrales como el haplotipo híbrido deberán tener el mismo tamaño, es decir, la misma cantidad de SNP.

Loter, y por consiguiente, este trabajo, está basado en el "*copying model*" introducido en la publicación *Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data* [10]. Este modelo asume que los haplotipos híbridos se conforman a partir de porciones de los haplotipos ancestrales, a modo de mosaico. En la Figura 2.1 se muestra un ejemplo.

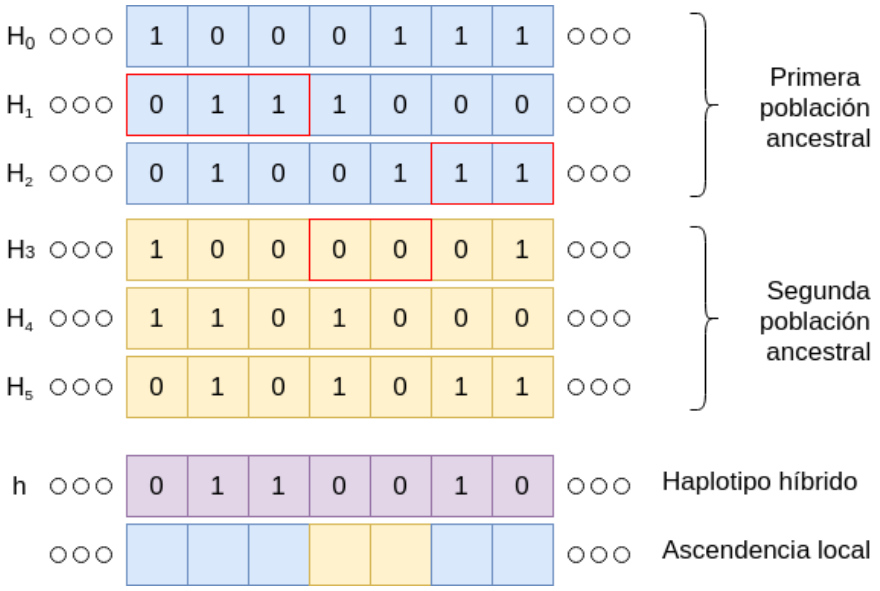


Figura 2.1: Ejemplo de la composición de un haplotipo híbrido a partir de porciones (no exactas) de 3 haplotipos ancestrales. Las porciones que conforman el haplotipo híbrido son aquellas resaltadas en rojo. En la última secuencia del diagrama se puede ver el valor de la ascendencia local para este ejemplo, las posiciones 3 y 4 (cuadros amarillos) se correspondería con la segunda población ancestral mientras que las restantes (cuadros azules) con la primera.

Teniendo en cuenta lo que establece el "*copying model* " se puede concluir que cualquier porción del haplotipo híbrido ha de encontrarse en alguno de los haplotipos ancestrales, incluso en el caso en el que dichas porciones (la del haplotipo híbrido y la del haplotipo ancestral del que supuestamente deriva) no concuerden. A partir de esta afirmación es posible entrever una de las características que hacen que la formulación del problema se corresponda con la formulación propia de un problema de optimización, puesto que habrá que **minimizar la no concordancia**. Aparte de la concordancia, se tendrá en cuenta otro aspecto a la hora de evaluar las soluciones al problema. Como ya se especificó en la sección 1.2, los haplotipos son conjuntos de SNP que están estadísticamente asociados y tienden a heredarse juntos, por ello, se deberá **minimizar la cantidad de cambios o saltos entre haplotipos ancestrales** necesarios para componer o modelar el haplotipo híbrido.

2.1. Codificación de las soluciones

Las soluciones al problema se representan como vectores de tamaño igual al de los haplotipos de los individuos. Cada posición de una solución almacena un índice, dicho índice referencia al haplotipo ancestral del que se origina el SNP de esa misma posición del haplotipo híbrido, teniendo en cuenta la siguiente notación:

- H: Conjunto de haplotipos ancestrales
- h: Haplotipo híbrido
- j: Posición (SNP) arbitraria
- S: Solución arbitraria

La codificación de las soluciones se definiría a partir de la expresión:

$$S_j = k \longleftrightarrow h^j \text{ se origina a partir de } H_k^j$$

Dada esta codificación de las soluciones, el **tamaño del espacio de búsqueda** vendría determinado por la expresión $(2n)^{|S|}$, teniendo en cuenta que n es el número de individuos de las poblaciones ancestrales y $|S|$ el tamaño de la solución, o lo que es lo mismo, el tamaño de los haplotipos. En la Figura 2.2 se presenta un ejemplo de solución al problema siguiendo el mismo ejemplo expuesto en la Figura 2.1.

2.2. Función objetivo

El problema de optimización consiste en minimizar la Ecuación 2.1:

$$C(S_1, S_2, \ldots, S_p) = \sum_{j=1}^{p} |h^j - H_{S_j}^j| + \lambda \sum_{j=0}^{p-1} 1_{S_j \neq S_{j+1}} \tag{2.1}$$

La función objetivo se compone de una función de pérdida y una función de regularización.

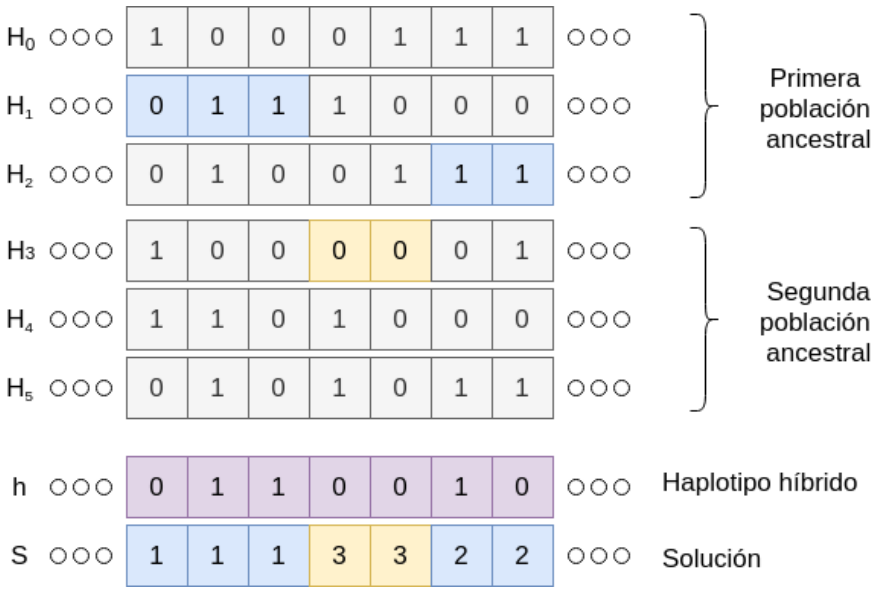


Figura 2.2: Ejemplo de una solución al problema. Atendiendo al diagrama, las 3 primeras posiciones se originan a partir de H_1, las dos siguientes a partir de H_3 y las dos últimas a partir de H_2.

2.2.1. Función de pérdida

La función de pérdida (Ecuación 2.2) contabiliza la cantidad de ocasiones en las que no existe concordancia entre un SNP híbrido h^j y el SNP de origen $H^j_{s_j}$ (Figura 2.3). En adelante esta situación se denominará **fallo**. Por cada fallo presente en la solución se penalizará **1 unidad**.

$$\sum_{j=1}^{p} |h^j - H^j_{S_j}| \tag{2.2}$$

Figura 2.3: Ejemplo de los posibles casos de la función de pérdida. Las situaciones representadas en las posiciones centrales del diagrama se consideran fallos puesto que el SNP híbrido y el SNP de origen no se corresponden.

2.2.2. Función de regularización

La función de regularización (Ecuación 2.3) penaliza los saltos o cambios de haplotipos ancestrales en posiciones adyacentes de la solución (Figura 2.4). λ es el parámetro de regularización que determina la cantidad de unidades con las que se debe penalizar cada uno de los saltos

de haplotipos ancestrales presentes en una solución, es decir, por cada salto se penalizarán λ **unidades** (Figura 2.5).

$$\lambda \sum_{j=0}^{p-1} 1_{S_j \neq S_{j+1}} \tag{2.3}$$

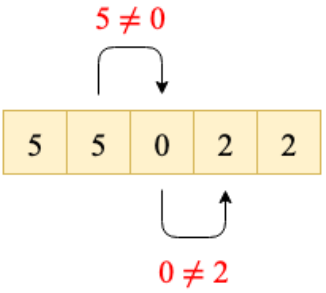


Figura 2.4: Ejemplo del concepto de salto en la evaluación de una solución atendiendo a la función de regularización. Ambos saltos se penalizarán con λ unidades.

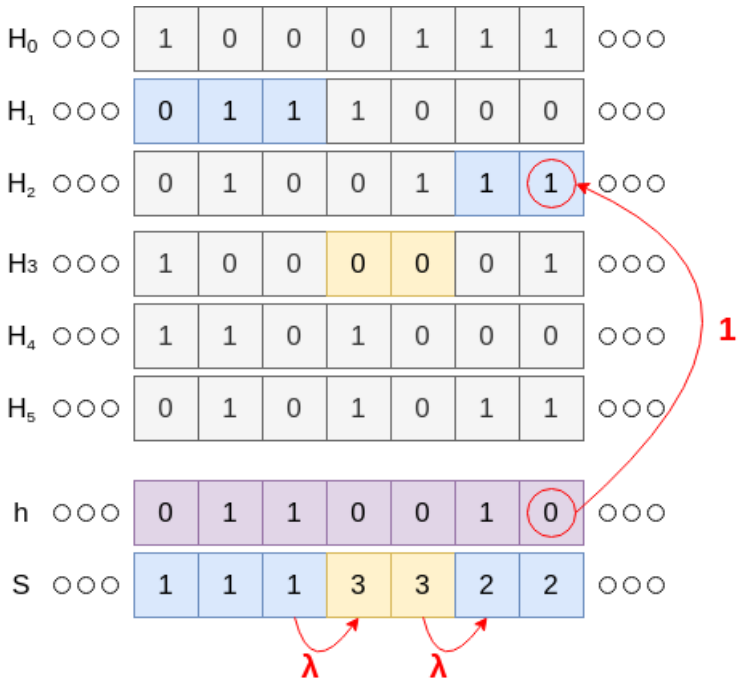


Figura 2.5: Ejemplo de evaluación de una solución. Atendiendo a la función objetivo, a la solución presente en el ejemplo se le asociaría un valor objetivo de $2\lambda + 1$ pues se producen 2 saltos y un fallo.

Capítulo 3

Implementación

En este Capítulo se presentan las técnicas de resolución implementadas, así como algunos métodos de interés, para resolver el problema de la estimación de la ascendencia local.

3.1. Generación de soluciones

Para la generación de soluciones iniciales se implementó un generador aleatorio que sigue una distribución uniforme. El método recibe como parámetros el tamaño de las soluciones $|h|$ (tamaño de los haplotipos) y la cantidad de haplotipos ancestrales $|H|$ con los que se cuenta. El funcionamiento es simple, en cada posición de la solución a generar se introduce un número entero aleatorio desde 0 hasta $|H| - 1$. Generar una solución supone recorrer un vector por lo que la complejidad temporal del método es $\mathcal{O}(n)$. En el Algoritmo 1 se muestra el pseudocódigo del método de generación de soluciones aleatorias.

Algoritmo 1: Generador de soluciones aleatorias

Datos: $|H|$, $|h|$
Resultado: Nueva solución aleatoria

```
1 S = [|h|]
2 for i in {0, 1, ..., |h| − 1} do
3 |   S_i = RandomInteger(0, |H| − 1)
4 end
5 return S
```

Un dato importante a tener en cuenta reside en el hecho de que el generador aleatorio sigue una distribución uniforme para asignar los índices de cada una de las posiciones de la solución, esto implica que sea poco probable ($\frac{1}{|H|^2}$) que se asigne el **mismo índice a dos posiciones adyacentes**. Cabe destacar que precisamente éste es uno de los aspectos que se tienen en cuenta en la evaluación de soluciones y que por ello, la calidad de las soluciones generadas aleatoriamente mediante este método es sumamente baja.

3.2. Evaluación de soluciones

Teniendo en cuenta la codificación de las soluciones presentada en la sección 2.1 y la función objetivo del problema de la estimación de la ascendencia local definida en la sección 2.2, evaluar una solución conlleva únicamente realizar un bucle iterando a través de sus posiciones. Por cada posición i, habrá que comprobar si el SNP h^i es igual al SNP de origen $H^i_{S_i}$, además de si el

haplotipo ancestral de origen de la posición i es el mismo que el de la posición $i+1$. A partir de esto, se puede concluir que la complejidad temporal de la función encargada de llevar a cabo la evaluación de soluciones es $\mathcal{O}(n)$. El Algoritmo 2 muestra el pseudocódigo de una posible implementación para esta función.

Algoritmo 2: Evaluador de soluciones

Datos: H, S, h, λ
Resultado: Valor objetivo de S

```
value = 0
for i in {0, 1, ..., n − 2} do
    if (h^i ≠ H^i_{S_i}) then
        value = value + 1
    end
    if (S_i ≠ S_{i+1}) then
        value = value + λ
    end
end
if (h^{n−1} ≠ H^{n−1}_{S_{n−1}}) then
    value = value + 1
end
return value
```

De cara al ahorro computacional, se llevó a cabo el desarrollo de una función de **evaluación incremental**. Ésta mide la variación que se ocasiona en el valor objetivo de una solución al problema tras haber aplicado un cambio o movimiento.

Un **cambio** C en una solución S queda definido por un par de índices:

- C_0: Posición de la solución en la que se debe aplicar el cambio.
- C_1: Nuevo valor para la posición C_0 de S

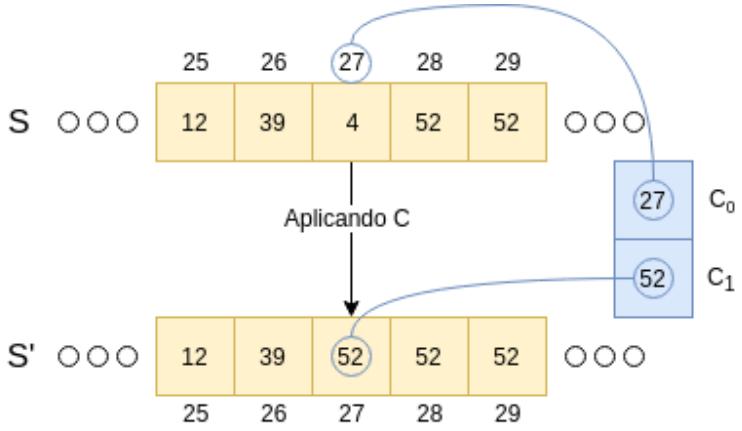


Figura 3.1: Ejemplo de aplicación de un cambio a una solución. El cambio C a aplicar está compuesto por el par de índices $\{27, 52\}$, por lo que habrá que sustituir el valor presente en la posición 27 de S por el índice 52.

Cabe recordar que los valores que se almacenan en una solución hacen referencia al haplotipo ancestral del que deriva el haplotipo híbrido en esa misma posición, por lo que, realizar un cambio C sobre una solución S equivale a establecer que h^{C_0} deriva de $H^{C_0}_{C_1}$ en vez de $H^{C_0}_{S_{C_0}}$. Recalcular el valor objetivo v a partir de un cambio no supone muchas complicaciones, tan solo habrá que despenalizar S, si procediese, por haber albergado el índice S_{C_0} en la posición C_0 y penalizarla, si fuera necesario, por introducir el índice C_1 en la posición C_0, véase la Figura 3.1.

Llevar a cabo las operaciones necesarias para recalcular el valor objetivo requiere un tiempo constante por lo que la complejidad temporal de esta función es $\mathcal{O}(1)$. En los métodos en los que se deben realizar numerosas evaluaciones, contar con con una función de evaluación incremental supone un **gran ahorro computacional**. En el Algoritmo 3 se muestra el pseudocódigo asociado a la función de evaluación incremental implementada.

Algoritmo 3: Evaluador incremental de soluciones

Datos: H, S, h, λ, v, C
Resultado: Valor objetivo de S al aplicar C

```
value = v − (|h^{C0} − H^{C0}_{S_{C0}}|) + (|h^{C0} − H^{C0}_{C1}|)
if (C1 > 0) then
    if (S_{C1−1} ≠ S_{C0}) then
        value = value - λ
    end
    if (S_{C1−1} ≠ S_{C1}) then
        value = value + λ
    end
end
if (C1 < n − 1) then
    if (S_{C1+1} ≠ S_{C0}) then
        value = value - λ
    end
    if (S_{C1+1} ≠ S_{C1}) then
        value = value + λ
    end
end
return value
```

3.3. Método exacto

La implementación de un método exacto que resolviese el problema de la estimación de la ascendencia local resultó necesaria para poder **comparar los resultados** obtenidos por los métodos aproximados desarrollados. Disponer de un método exacto permite llevar a cabo ejecuciones, controlando el tamaño de instancia del problema, con el objetivo de conseguir el óptimo global para dicha instancia del problema. De esa manera, tras ejecutar la misma instancia del problema con el resto de técnicas, es posible averiguar cuan lejos están las soluciones obtenidas de la mejor solución posible.

En concreto, se implementó el método exacto mediante programación dinámica que se propone en *Loter* [2]. Se basa en que una solución de tamaño p, con p SNPs, puede construirse

fácilmente a partir de una solución de tamaño $p-1$. Dada esta situación, pueden ocurrir dos escenarios:

1. El SNP de la posición p de h deriva del mismo haplotipo ancestral que el SNP de la posición $p-1$, es decir, $S_{p-1} = S_p$. Recalcular el valor objetivo de la solución teniendo en cuenta la posición p implicaría comprobar si se produce un **fallo** en dicha posición, tal y como se muestra en la Ecuación 3.1:

$$C(S_1, S_2, \ldots, S_p) = C(S_1, S_2, \ldots, S_{p-1}) + |h^p - H^p_{S_{p-1}}| \tag{3.1}$$

2. El SNP de la posición p de h deriva de un haplotipo ancestral distinto del que deriva el SNP de la posición $p-1$, es decir, $S_{p-1} \neq S_p$. Recalcular el valor objetivo de la solución teniendo en cuenta la posición p implicaría comprobar si se produce un **fallo** en dicha posición y penalizar con λ unidades el salto de haplotipo ancestral producido entre las posiciones $p-1$ y p, tal y como se muestra en la Ecuación 3.2:

$$C(S_1, S_2, \ldots, S_p) = C(S_1, S_2, \ldots, S_{p-1}) + |h^p - H^p_{S_p}| + \lambda \tag{3.2}$$

Teniendo en cuenta lo ya expuesto y tal y como se establece en *Loter* [2], es posible obtener el valor objetivo asociado al óptimo global realizando una **búsqueda del camino de coste mínimo** en el grafo representado en la Figura 3.2. Nótese que n es la cantidad de individuos de las poblaciones ancestrales y p la cantidad de SNPs de los haplotipos (tamaño de la solución).

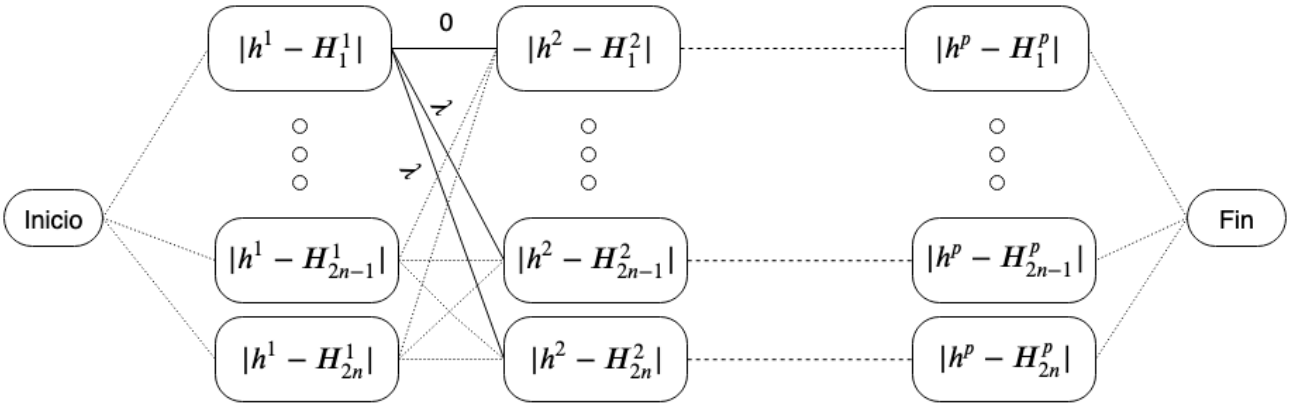


Figura 3.2: Grafo al que se reduce el problema de la estimación de la ascendencia local según la formulación propuesta en *Loter* [2]. El coste asociado al camino de coste mínimo del grafo representa el valor objetivo del óptimo global del problema.

Cada nodo del grafo de la Figura 3.2 tiene un valor $Q(i,j)$ asociado que representa el valor objetivo de la solución óptima para las primeras j posiciones cuando el SNP j del haplotipo híbrido h deriva del SNP del haplotipo ancestral H^j_i. Para obtener el valor objetivo del óptimo global, el método de programación dinámica debe calcular dicho valor en cada uno de los nodos del grafo y obtener el mínimo $Q(i,j)$ para $j = |h|$, es decir, cuando se han evaluado todas las posiciones o SNPs del haplotipo híbrido. Obtener el valor objetivo del óptimo global al problema supone rellenar una tabla de valores, tal y como se ha expuesto, el valor de $Q(i,j)$ se puede calcular a partir de los valores previamente calculados en la columna anterior(tamaño de solución $j-1$) siguiendo la Ecuación 3.3:

$$Q(i,j) = min(Q(i,j-1), min_{i' \in \{1,...,|H|\}}\{Q(i',j-1)\} + \lambda) + |h^j - H_i^j| \tag{3.3}$$

Con el propósito de reducir la complejidad, durante el proceso de completado de la tabla de valores Q, conviene llevar un registro con el mínimo valor de la columna anterior a la que se está rellenando, de esta manera se evita tener que recorrer la columna anterior repetidamente. Incorporando este ajuste, la complejidad temporal y espacial del método resulta $\mathcal{O}(n \cdot p)$, siendo n el número de individuos pertenecientes a las poblaciones ancestrales y p la cantidad de SNPs de los haplotipos. El Algoritmo 4 muestra una posible implementación de este método.

Algoritmo 4: Método de programación dinámica

```
Datos: H, h, λ
Resultado: Valor objetivo del óptimo global
table = [|H|][|h|]
previousColumnMin = ∞
currentColumnMin = ∞
for i in {0, 1, ..., |H| − 1} do
    table[i][0] = |h^0 − H_i^0|
    if (table[i][0] < previousColumnMin) then
        previousColumnMin = table[i][0]
    end
end
for j in {1, ..., |h| − 1} do
    for i in {0, 1, ..., |H| − 1} do
        table[i][j] = table[i][j − 1]
        if (table[i][j − 1] > previousColumMin + λ) then
            table[i][j] = previousColumMin + λ
        end
        table[i][j] = table[i][j] + |h^j − H_i^j|
        if (table[i][j] < currentColumnMin) then
            currentColumnMin = table[i][j]
        end
    end
    previousColumnMin = currentColumnMin
end
return currentColumnMin
```

Durante el desarrollo de las técnicas aproximadas se creyó conveniente conocer la forma, naturaleza y particularidades de alguna de las soluciones óptimas del problema, por esta razón, se llevo a cabo la implementación de un método auxiliar capaz de reconstruir la solución a partir de la tabla de valores dada por el método de programación dinámica. Para ello fue necesario que cada posición de la tabla $Q(i,j)$, además de almacenar el valor objetivo de la solución óptima de tamaño j siendo el haplotipo ancestral H_i el origen del SNP h^j, se guardase el índice del haplotipo ancestral (valor de i) del que derivó el valor $Q(i,j)$. A partir de la información disponible en cada una de las posiciones de la tabla, es posible reconstruir la solución óptima por medio de un método que implemente el Algoritmo 5.

Algoritmo 5: Método de reconstrucción de las soluciones óptimas

Datos: $|H|$, $|h|$, table
Resultado: Solución óptima

```
1  globalOptimum = [|h|]
2  minValue = ∞
3  globalOptimumIndex = -1
4  for i in {0, 1, ..., |H| − 1} do
5  |  if (table[i][|h| − 1] < minValue) then
6  |  |  minValue = table[i][|h| − 1]
7  |  |  globalOptimumIndex = i
8  |  end
9  end
10 globalOptimum[|h| − 1] = globalOptimumIndex
11 for i in {|h| − 1, |h| − 2, ..., 1} do
12 |  globalOptimum[i − 1] = table[globalOptimum[i]][i].index
13 end
14 return globalOptimum
```

Conociendo la forma y peculiaridades de las soluciones óptimas resultó más sencillo diseñar y desarrollar los operadores específicos del problema para el algoritmo evolutivo.

3.4. Búsqueda local

Una búsqueda local consiste en, partiendo de una solución inicial, realizar una serie de iteraciones en las que se van aceptando *vecinos* de la solución actual que mejoran el valor objetivo de la misma (Figura 3.3). Una vez que todos los *vecinos* disponibles son peores que la solución actual, se para la búsqueda, pues se ha alcanzado un **óptimo local** [17].

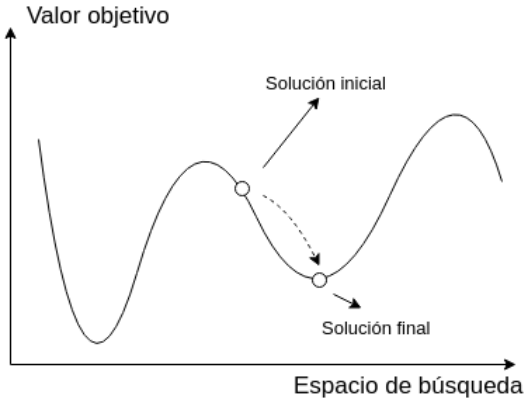


Figura 3.3: Diagrama del proceso de una búsqueda local. Comenzando con una solución inicial y tras explorar la vecindad y aceptar vecinos iterativamente se termina alcanzando un óptimo local.

Un *vecino* de una solución se define como otra solución próxima a ésta que pertenece a su *vecindad*. La *vecindad* de una solución queda definida por un pequeño movimiento o perturbación. La vecindad se compone por todas aquellas posibles soluciones que surgen al aplicar dicho movimiento a la solución en cuestión. El cambio propuesto en la sección 3.2 al introducir el método de evaluación incremental sirve como ejemplo, la Figura 3.1 muestra dos soluciones

vecinas atendiendo a la vecindad asociada al cambio mencionado. El movimiento o perturbación determina el tamaño de la vecindad, y por consiguiente, el tiempo de búsqueda promedio necesario para encontrar un óptimo local en un una búsqueda local. El orden en el que se explora la vecindad de la solución actual y el criterio de aceptación de vecinos también determinan la naturaleza de la búsqueda local. La complejidad depende directamente del tamaño de la vecindad y de como se construye.

Algoritmo 6: Estructura básica de una búsqueda local

```
   Resultado: Óptimo local
 1 currentSolution = GenerateRandomSolution()
 2 currentSolutionValue = EvaluateSolution(currentSolution)
 3 neighbourhood = ConstructNeighbourhood(currentSolution)
 4 neighboursToVisit = |neighbourhood|
 5 while neighboursToVisit > 0 do
 6 |   neighbour = SelectNeighbour(neighbourhood)
 7 |   neighbourValue = EvaluateSolution(neighbour)
 8 |   if neighbourValue < currentSolutionValue) then
 9 |   |   currentSolutionNeighbour = neighbourValue
10 |   |   currentSolution = neighbour
11 |   |   neighbourhood = ConstructNeighbourhood(currentSolution)
12 |   else
13 |   |   neighboursToVisit = neighboursToVisit - 1;
14 |   end
15 end
16 return currentSolution
```

El Algoritmo 6 muestra el pseudocódigo de la estructura seguida al implementar las variantes de búsqueda local para este proyecto. Nótese que existen ciertos aspectos de dicha estructura que deben ser aclarados:

- El método *ConstructNeighbourhood* deberá tener en cuenta la perturbación o movimiento que define la vecindad para poder construir la vecindad.
- El método *SelectNeighbour* deberá ser capaz de discriminar aquellos vecinos que ya se han escogido previamente y no se han aceptado.
- El criterio de aceptación de vecinos se basa en la técnica de escalada, si el vecino evaluado es mejor que la solución actual se acepta inmediatamente y se actualiza la vecindad.

La estructura que se expone en el Algoritmo 6 es un ejemplo simple y poco eficiente. Si atendemos a la implementación concreta llevada a cabo para este proyecto debería mencionarse que la vecindad como tal no está constituida por todas las soluciones vecinas a la solución actual sino por los posibles cambios que se le pueden realizar a la misma. De esta manera, al seleccionar un vecino, realmente se está seleccionando el cambio que transformará la solución actual en ese vecino. Actualizar la vecindad tras haber aceptado un vecino es tan simple como eliminar el cambio aplicado e introducir el inverso. Para evaluar los vecinos no se realiza una evaluación completa de éstos, se lleva a cabo una **evaluación incremental** que es menos costosa computacionalmente como se describe en la sección 3.2.

El orden de selección de vecinos o movimientos viene dado aleatoriamente. Mediante la actualización de un índice se asegura que los vecinos seleccionados no han sido elegidos previamente.

Este índice establece la frontera en la vecindad entre los vecinos evaluados y los no evaluados, cada vez que se rechaza un vecino, éste se intercambia la posición con el último de los vecinos no evaluados y se incrementa el índice. En el caso de que el vecino que se ha seleccionado se acepte porque mejora el valor objetivo de la solución actual, se actualiza la vecindad y se restablece el índice. Se puede ver un ejemplo de este método en la Figura 3.4.

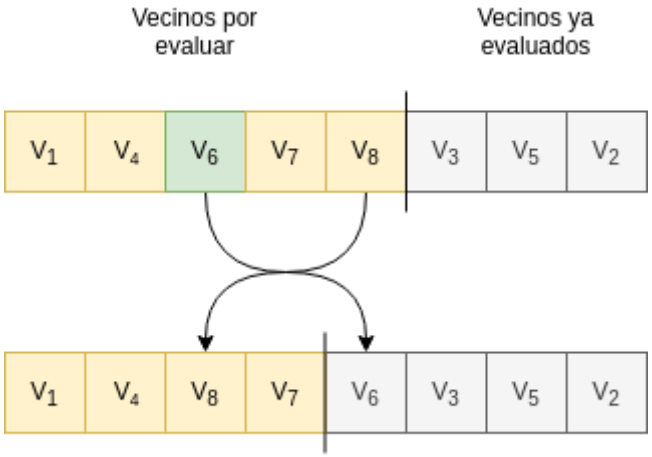


Figura 3.4: Ejemplo del proceso de selección de vecinos de las variantes de búsqueda local. El vecino V_6 es seleccionado para la evaluación. Su valor objetivo es peor que el de la solución actual y por ello se rechaza, se realiza un intercambio entre éste y el último vecino no evaluado ,V_8, y se adelanta una posición el índice que define la frontera entre vecinos evaluados y no evaluados, para que V_6 no se vuelva a seleccionar.

Con el propósito de disponer de varias opciones a la hora de realizar el estudio experimental de este TFG, se implementaron **tres variantes** de la estructura anteriormente expuesta de búsqueda local. Como ya se ha comentado en este Capítulo todas siguen un mismo esquema, las diferencias entre ellas residen en cómo se recorre y construye la vecindad.

3.4.1. Greedy Local Search

Greedy Local Search (GLS) es la primera y más básica de las variantes de búsqueda local implementadas, Algoritmo 7. La vecindad queda definida por el cambio comentado en la sección 3.2 al introducir el método de evaluación incremental. Se construye teniendo en cuenta la siguiente expresión:

$$V_{GLS} = \{i, j\} / \quad \forall i \in \{0, \ldots, |h-1|\} \ \wedge \ \forall j \in \{0, \ldots, |H-1|\} \ / \ S_i \neq j$$

Por tanto, la vecindad la conforman todos aquellos posibles vecinos que surgen al seleccionar una de las posiciones de la solución actual y alterar el índice que contiene. Teniendo en cuenta que los posibles índices que puede albergar cada una de las posiciones de la solución están en el intervalo $[0, |H|-1]$ y que las soluciones tienen un tamaño $|h|$, la cantidad de vecinos de la vecindad en esta búsqueda local queda definida por la expresión:

$$|V_{GLS}| = (|H|-1) \cdot |h|$$

Para tamaños de entrada al problema considerablemente grandes la vecindad adquiere tamaños intratables y, al igual que ocurre con el método de programación dinámica, esta variante

de búsqueda local no es escalable. Si se decidiese utilizar en una técnica aproximada (como un algoritmo evolutivo), ésta tendría la misma desventaja que el método de programación dinámica y además no aseguraría encontrar el óptimo global, por lo que sería una peor técnica de resolución. A pesar de que *GLS* presenta esta característica tan negativa, se creyó conveniente su implementación, tanto de cara al estudio experimental como por considerarse un primer paso a la hora de desarrollar las distintas variantes de búsqueda local.

Algoritmo 7: Construcción de la vecindad de la búsqueda local *Greedy*

Datos: $|H|$, $|h|$, S
Resultado: Vecindad de la búsqueda local *Greedy*

```
N = []
for i in {1, ..., |h| − 1} do
    for j in {0, 1, ..., |H| − 1} do
        if S_i ≠ j then
            N.insert({i, j})
        end
    end
end
return N
```

3.4.2. Promising Neighbours Local Search

Promising Neighbours Local Search (PNLS) es la segunda de las variantes de búsqueda local desarrolladas, Algoritmo 8. El método que sigue para construir la vecindad es exactamente igual al *GLS* por lo que **presenta las mismas desventajas**. La diferencia entre *PNLS* y *GLS* reside en la manera en la que se recorre la vecindad a la hora de evaluar los vecinos:

$$V_{GLS} = V_{PNLS} \qquad |V_{GLS}| = |V_{PNLS}|$$

Tal y como se comenta en la sección 3.4 para llevar el control de los vecinos que se han evaluado dentro de la vecindad se hace uso de un índice que establece la frontera entre éstos y los no evaluados. Siguiendo el procedimento común, cada vez que se acepta un vecino, la vecindad se actualiza y dicho índice es restablecido, pues los vecinos ya visitados lo habían sido desde la solución anterior. Sin embargo, en *PNLS* el índice no se reestablece tras haber aceptado un vecino, pues se considera que los vecinos que aún no han sido evaluados son **más prometedores** que aquellos que ya se han evaluado y rechazado.

Esta manera de recorrer la vecindad requiere de un mecanismo extra, pues no se puede asegurar que tras haber visitado todos los vecinos de la vecindad la solución actual sea un óptimo local. Por ello, tras visitar todos los vecinos el índice de frontera es restablecido y se vuelve a comprobar que ningún vecino es mejor que la solución actual. En el caso de que sí que exista un vecino mejor, se acepta y se restablece el índice, por lo que el proceso comienza de nuevo.

Algoritmo 8: Promising Neighbours Local Search

Datos: $|H|$, $|h|$
Resultado: Óptimo local

```
1  solution = GenerateRandomSolution()
2  solutionValue = EvaluateSolution(solution)
3  N = ConstructNeighbourhood(solution, |H|, |h|)
4  NSize = |N|
5  visited = 0
6  finalCheck = false
7  while visited < NSize do
8  |  index = RandomInteger(0, NSize - 1 - visited)
9  |  swap(N, index, NSize - 1 - visited)
10 |  neighbourValue = IncrementalEvaluation(solution, N[index])
11 |  if (neighbourValue < solutionValue) then
12 |  |  solutionValue = neighbourValue
13 |  |  UpdateSolution(solution, N[index])
14 |  |  UpdateNeighbourhood(solution, N[index], N)
15 |  |  if (finalCheck) then
16 |  |  |  finalCheck = false
17 |  |  |  visited = 0
18 |  |  end
19 |  else
20 |  |  visited = visited + 1;
21 |  end
22 |  if (visited = NSize & ! finalCheck) then
23 |  |  finalCheck = true
24 |  |  visited = 0
25 |  end
26 end
27 return solution
```

3.4.3. Adjacent Positions Local Search

Adjacent Positions Local Search (APLS) es la última de las variantes de búsqueda local implementadas, Algoritmo 9. Se diferencia de las anteriores variantes en la manera en la que se construye la vecindad a partir de una solución. *APLS* se desarrolló como un operador específico considerando las particularidades del problema de la estimación de la ascendencia local.

Como era de esperar teniendo en cuenta los aspectos a los que atiende la función objetivo, **las soluciones óptimas devueltas por el método de programación dinámica presentaban extensas porciones con el mismo índice**. Se decidió aprovechar esta particularidad creando una vecindad conformada únicamente por vecinos que contuviesen los índices de las posiciones adyacentes de la solución para cada una de las posiciones, es decir, la vecindad se construye atendiendo a la siguiente expresión:

$$V_{APLS} = \{0, S_1\} \cup \{|h| - 1, S_{|h|-2}\} \cup \{i, S_{i-1}\} \cup \{i, S_{i+1}\} \quad \forall i \in \{1, \ldots, |h| - 2\}$$

Por tanto, en la vecindad existen únicamente dos vecinos por cada una de las posiciones de la solución, exceptuando los extremos, que aportan un único vecino cada uno, por lo que se ha

reducido drásticamente el tamaño de la vecindad en comparación con las variantes de búsqueda local previamente expuestas:

$$|V_{APLS}| = (|h| - 1) \cdot 2$$

Esta reducción del tamaño de la vecindad implica que esta variante de búsqueda local si es escalable para tamaños de entrada mayores del problema, **pues el tamaño de la vecindad aumenta linealmente** atendiendo al tamaño de la entrada. *APLS* podría utilizarse en el algoritmo evolutivo sin comprometer su rendimiento o complejidad.

Algoritmo 9: Adjacent Positions Local Search

```
   Datos: |H|, |h|
   Resultado: Óptimo local
 1 solution = GenerateRandomSolution()
 2 solutionValue = EvaluateSolution(solution)
 3 N = [{0, 1}, {|h| − 1, |h| − 2}]
 4 for i in {1, ..., |h| − 2} do
 5 |   N.insert({i, i − 1})
 6 |   N.insert({i, i + 1})
 7 end
 8 NSize = |N|
 9 visited = 0
10 while visited < NSize do
11 |   index = RandomInteger(0, NSize - 1 - visited)
12 |   swap(N, index, NSize - 1 - visited)
13 |   neighbour = {N[index][0], solution[N[index][1]]}
14 |   neighbourValue = IncrementalEvaluation(solution, neighbour)
15 |   if (neighbourValue < solutionValue) then
16 |   |   solutionValue = neighbourValue
17 |   |   UpdateSolution(solution, neighbour)
18 |   |   visited = 0
19 |   end
20 |   else
21 |   |   visited = visited + 1;
22 |   end
23 end
24 return solution
```

Nótese que a la hora de construir la vecindad, el par de índices que se asocia a cada uno de los vecinos representan la posición de la solución a alterar y la posición de solución de la que se debe copiar el nuevo valor. Aprovechando este pequeño ajuste, no es necesario actualizar constantemente la vecindad pues los valores que se deben insertar en las posiciones seleccionadas se resuelven dinámicamente, Línea 13 del Algoritmo 9.

3.5. Iterated Local Search

Iterated Local search (ILS) consiste en aplicar iterativamente una fase de **perturbación** y una **búsqueda local** a un óptimo local inicial, Algoritmo 10. El propósito de aplicar la fase

de perturbación reside en intentar escapar de la optimalidad local explorando nuevas y prometedoras regiones del espacio de soluciones. Si la perturbación es lo suficientemente *contundente* en la fase de la búsqueda local se alcanzará un óptimo local distinto al actual. Atendiendo a un criterio de aceptación ese nuevo óptimo local puede convertirse en la solución actual, y por consiguiente, la solución sobre la que se aplique la fase de perturbación de la siguiente iteración [17]. En la Figura 3.5 se ejemplifica el proceso de la búsqueda local iterada.

Algoritmo 10: Estructura básica de una búsqueda local iterada

```
   Datos: S
   Resultado: Óptimo local
1  solution = LocalSearch(S)
2  while stop condition is unmet do
3  |   newSolution = Perturb(solution)
4  |   newSolution = LocalSearch(newSolution)
5  |   if (acceptance condition is met) then
6  |   |   solution = newSolution
7  |   end
8  end
9  return solution
```

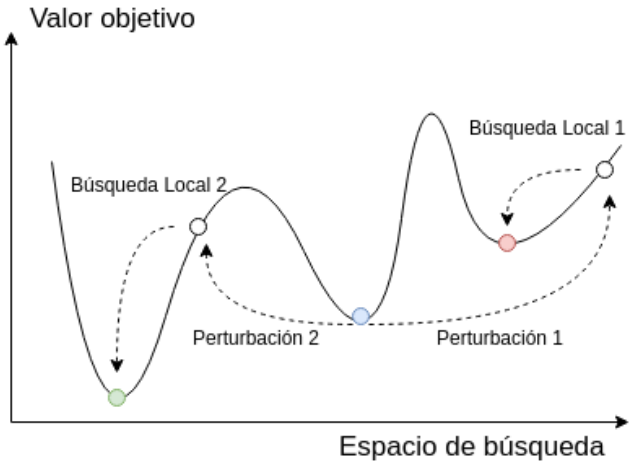


Figura 3.5: Diagrama del proceso de una búsqueda local iterada. En la primera iteración, el nuevo óptimo local (punto rojo) alcanzado no cumple el criterio de aceptación. En la segunda iteración el óptimo local alcanzado (punto verde) sí que cumple el criterio de aceptación y se convierte en la solución actual.

Para la implementación concreta llevada a cabo en este trabajo el criterio de aceptación consiste en **aceptar aquellas soluciones** con mejor valor objetivo que la actual. La fase de perturbación (*Shake*) desarrollada consiste en seleccionar una porción de la solución de tamaño variable (entre el 4 % y el 6 % del tamaño total) y alterar aleatoriamente los índices de cada posición de la porción seleccionada. Cabe destacar que se procuró que la selección fuera cíclica, es decir, que en el caso de que se seleccionase como posición inicial de la región a alterar una de las posiciones finales de la solución, la región continuase al inicio de la solución. El Algoritmo 11 representa el pseudocódigo de la fase de perturbación implementada.

Algoritmo 11: Fase de perturbación de la búsqueda local iterada *Shake*

Datos: S
Resultado: Solución alterada
1 size = RandomFloat(0.04, 0.06) * $|S|$)
2 startIndex = Round(RandomInteger(0, $|h| - 1$))
3 **for** i in {startIndex, ..., startIndex + size} **do**
4 $\quad S_{i\,\%|S|}$ = RandomInteger(0, $|h| - 1$)
5 **end**
6 **return** S

3.6. Evolutionary Algorithm

Los *algoritmos evolutivos* se basan en la evolución de una población de individuos. Cada uno de los individuos es una posible solución al problema, inicializadas aleatoriamente. La evolución de la población se consigue mediante un proceso iterativo, cada iteración de dicho proceso se denomina *generación*. El valor objetivo de cada solución o individuo determina su adaptabilidad al medio o *fitness*, es decir, su capacidad de sobrevivir a la siguiente generación. El proceso terminará cuando se alcance una condición de parada determinada [17].

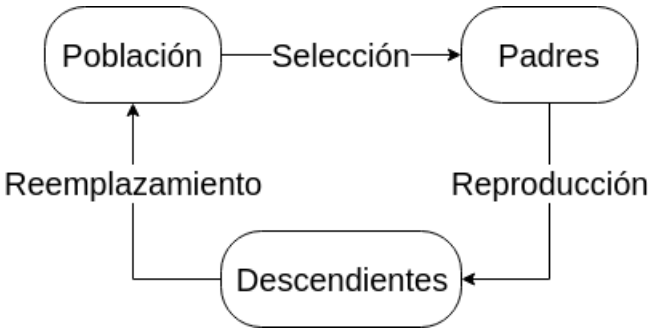


Figura 3.6: Diagrama del proceso de una generación en un algoritmo evolutivo.

Tal y como se muestra en la Figura 3.6, cada generación la componen una serie de fases. La primera de ellas es la *fase de selección*. En ella, se seleccionan los individuos de la población que se convertirán en padres de los nuevos individuos a generar. La siguiente, es la *fase de reproducción*, en ésta, los individuos seleccionados en la primera fase se reproducen y generan nuevos individuos (soluciones al problema). Por último, se lleva a cabo la *fase de reemplazamiento*, en la que se determina que individuos de entre los padres y los descendientes sobrevivirán y conformarán la población de la siguiente generación.

A su vez, la *fase de reproducción* está compuesta por dos operadores. El primero de ellos es el de recombinación o cruce, típicamente se trata de un operador binario. La idea consiste en aplicar el operador sobre los padres seleccionados en la *fase de selección* para generar nuevos individuos que hereden características de ambos. El segundo de los operadores es el de mutación, éste afecta a los nuevos individuos resultantes del operador de cruce, consiste en alterar los individuos de manera aleatoria buscando obtener diversidad en la población. El Algoritmo 12 representa la estructura básica de un algoritmo evolutivo.

Algoritmo 12: Estructura básica de un algoritmo evolutivo

```
   Resultado: Mejor individuo conseguido
 1 population = GeneratePopulation()
 2 bestIndividual = SearchBestIndividual(population)
 3 for each individual in population do
 4 |   individual = LocalSearch(individual)
 5 end
 6 while stop condition is unmet do
 7 |   parents = Selection(population)
 8 |   offsprings = Reproduction(parents)
 9 |   population = Replacement(parents, offsprings)
10 |   bestIndividual = UpdateBestIndividual(population, bestIndividual)
11 end
12 return bestIndividual
```

Es importante mencionar una serie de aspectos característicos del algoritmo evolutivo implementado, teniendo en cuenta que el tamaño de la población especificado es n:

- El tamaño de la población es igual a $n+1$, ya que siempre se mantiene el mejor individuo encontrado hasta el momento.
- El operador de cruce es binario, por lo que en la fase de selección se formarán parejas de individuos.
- El operador de cruce genera dos individuos por lo que se deberán seleccionar $\frac{n}{2}$ parejas de padres en la fase de selección.
- El tamaño de la población es constante durante todo el proceso.
- En el caso de que se aplique un operador de mutación se debe aplicar obligatoriamente a todos y cada uno de los descendientes generados.
- Los individuos de la población han de ser óptimos locales, tras generar la población inicial y tras generar los decendientes en la *fase de reproducción* se ha de ejecutar una búsqueda local sobre cada individuo generado.

Con el objetivo de poder realizar diversas pruebas, la implementación llevada a cabo del algoritmo evolutivo permite utilizar distintas técnicas para cada una de las fases. Desde la condición de parada hasta la búsqueda local, todo está parametrizado para poder realizar ejecuciones probando distintas configuraciones. En los siguientes apartados se expondrán las técnicas desarrolladas para cada una de las fases del algoritmo evolutivo.

Antes de pasar a presentar las técnicas implementadas para cada una de las fases conviene mencionar el concepto de **variedad de la población**. La variedad de la población es un indicativo de lo diversa que es, mantener la variedad alta es importante para evitar la *convergencia prematura* de la población. Se denomina *convergencia prematura* al estado en el que la población es incapaz de generar descendientes de mayor *fitness* que el de los padres. Una vez que la población alcanza el estado de *convergencia prematura* no hay rango de mejora posible, por ello, ha de evitarse. Para conocer la variedad de la población es necesario establecer una medida de distancia que determine cuan diferentes son dos soluciones. Teniendo en cuenta la codificación de las soluciones, se optó por la distancia *Hamming* como medida de distancia entre individuos. La distancia *Hamming* contabiliza la cantidad de diferencias existentes entre dos

soluciones, una diferencia equivale a tener distintos valores en la misma posición de la solución, véase Algoritmo 13. La variedad de un individuo dentro de una población se define como la distancia mínima entre dicho individuo y los demás integrantes de la población. La variedad media de una población la determina la media aritmética de las variedades de cada uno de los individuos.

Algoritmo 13: Distancia Hamming entre dos soluciones

Datos: firstSolution, secondSolution
Resultado: Distancia entre las soluciones

```
1 distance = 0
2 for i in {0, ..., |firstSolution|} do
3 |   if firstSolution_i ≠ secondSolution_i then
4 |   |   distance = distance + 1
5 |   end
6 end
7 return distance
```

3.6.1. Fase de selección

En la fase de selección se escogen aquellos individuos que serán padres en la actual generación. Para este problema, se optó por implementar la *selección por torneo* como método de selección de individuos, y en concreto, se desarrolló el *torneo binario*. En este método se seleccionan cuatro contendientes aleatoriamente de entre los integrantes de la población, tras esto, los contendientes luchan por parejas comparando su *fitness*, el que mejor *fitness* tenga en cada pareja gana y queda elegido como uno de los padres (Figura 3.7).

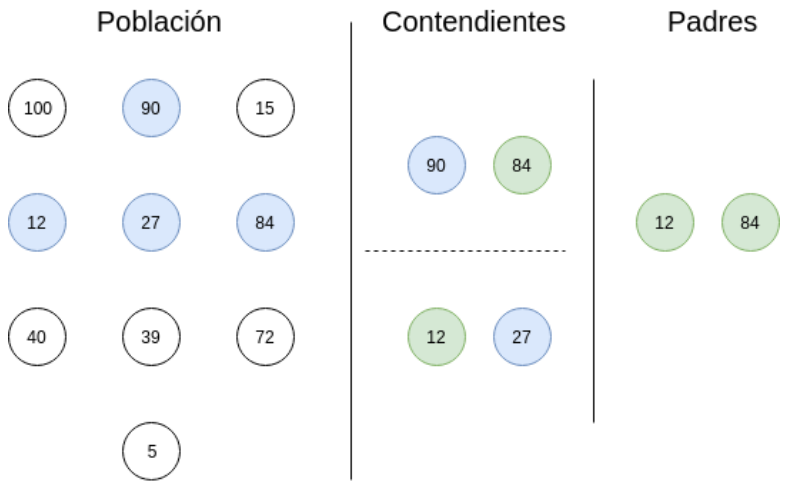


Figura 3.7: Diagrama del método de selección *torneo binario*. Los individuos azules de la población son los contendientes. Los individuos verdes entre los contendientes son los ganadores y padres.

Debido a la naturaleza del operador de cruce implementado, se explicará a continuación, se decidió añadir un pequeño ajuste a este método para que **no se permitiese escoger al mismo individuo** dos veces en la misma pareja de padres, pues esto supondría generar como descendientes dos individuos absolutamente iguales, impactando negativamente en la variedad

de la población. En el Algoritmo 14 se presenta el pseudocódigo del método implementado, el ajuste comentado se controla por medio del flag *allowClones*.

Algoritmo 14: Método de selección *Binary Tournament*

```
Datos: population, allowClones
Resultado: Pareja de padres
first = RandomInteger(0, |population|)
second = RandomInteger(0, |population|)
third = RandomInteger(0, |population|)
fourth = RandomInteger(0, |population|)
firstParent = (first.value < second.value) ? first : second
secondParent = (third.value < fourth.value) ? second : fourth
if (!allowClones) then
    while firstParent = secondParent do
        third = RandomInteger(0, |population|)
        fourth = RandomInteger(0, |population|)
        secondParent = (third.value < fourth.value) ? second : fourth
    end
end
return {firstParent, secondParent}
```

3.6.2. Fase de reproducción

Como se ha mencionado con anterioridad, la *fase de reproducción* está compuesta por dos operadores, el de cruce y el de mutación. La estructura seguida para implementar los métodos de la *fase de reproducción* se muestra en el Algoritmo 15. Tras aplicar los operadores de cruce y mutación se ha de realizar una búsqueda local sobre cada uno de los individuos descendientes para asegurar que la población está compuesta por óptimos locales.

Algoritmo 15: Estructura de un método de la *fase de reproducción*

```
Datos: parents
Resultado: Conjunto de individuos descendientes
offsprings = []
for each couple in parents do
    childs = CrossoverOperator(couple)
    childs[0] = LocalSearch(MutationOperator(childs[0]))
    childs[1] = LocalSearch(MutationOperator(childs[1]))
    offsprings.insert(childs[0])
    offsprings.insert(childs[1])
end
return offsprings
```

Operador de cruce

Es el primer operador de la *fase de reproducción*, se aplica sobre las parejas de individuos seleccionadas durante la *fase de selección* para generar los individuos descendientes de la población. El operador de cruce implementado tiene como nombre *Two Point Crossover*, véase

Algoritmo 16. Consiste en seleccionar aleatoriamente dos posiciones de los individuos conocidas como puntos de recombinación, los descendientes se generan a partir del intercambio de la sección que queda entre los puntos de recombinación de los progenitores. En la Figura 3.8 se puede ver un ejemplo de la aplicación del método.

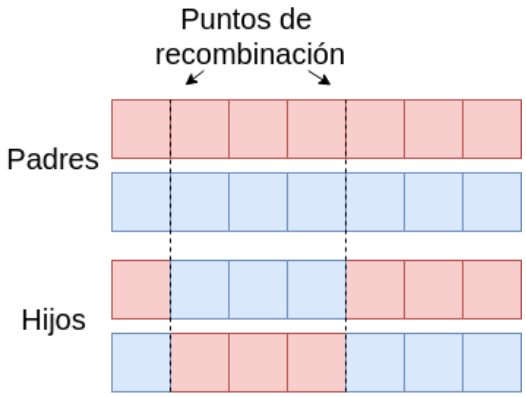


Figura 3.8: Ejemplo de aplicación del operador de cruce *Two Point Crossover*. Los decendientes se conforman a partir de la combinación de las secciones resultantes en los padres tras elegir los puntos de recombinación.

Tal y como se indicó con anterioridad, si se diese el caso en el que ambos progenitores son el mismo individuo, los dos individuos descendientes generados serían idénticos pues no importa que puntos de recombinación se seleccionen ya que las secciones resultantes serían exactamente iguales. Por esto, incluir un mecanismo en la *fase de selección* que evite esta situación es algo a tener en cuenta con el propósito de preservar una buena variedad en la población.

Algoritmo 16: Operador de cruce *Two Point Crossover*

Datos: couple
Resultado: Descendientes
1 firstPoint = RandomInteger(0, |couple[0]|)
2 secondPoint = RandomInteger(0, |couple[0]|)
3 **if** firstPoint > secondPoint **then**
4 | Swap(firstPoint, secondPoint)
5 **end**
6 offsprings = SwapRanges(firstPoint, secondPoint, couple[0], couple[1])
7 **return** offsprings

Operador de mutación

El operador de mutación se encarga de alterar aleatoriamente los nuevos individuos generados con el propósito de obtener diversidad en la población. El operador de mutación desarrollado tiene como nombre *Set Portion*, véase Algoritmo 17. Al igual que la búsqueda local *APLS* (sección 3.4.3) se ha diseñado a medida para este problema, es decir, es un operador específico concebido tras conocer la naturaleza y características propias de algunos de los óptimos globales obtenidos por el método exacto implementado (sección 3.3).

El operador de mutación *Set Portion* consiste en alterar una cantidad variable de posiciones de un individuo estableciéndolas a un mismo valor aleatorio, y al igual que el método imple-

mentado *Shake* (fase de perturbación de la *ILS*) de manera cíclica. En la Figura 3.9 puede verse un ejemplo de aplicación del operador *SetPortion*.

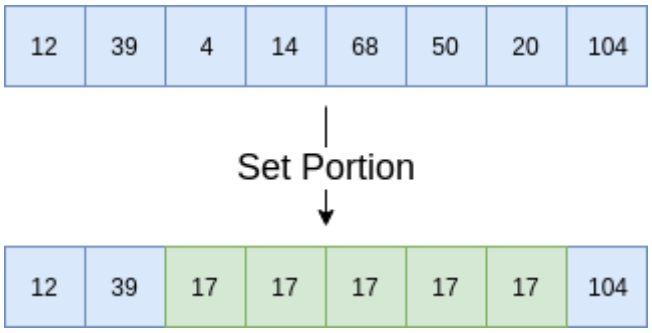


Figura 3.9: Ejemplo de aplicación del operador de mutación *Set Portion*. Tras aplicar el operador de mutación, 5 posiciones resultaron afectadas y en todas se introdujo el mismo valor (17).

Algoritmo 17: Operador de mutación *Set Portion*

Datos: individual, minPositions, maxPositions, $|H|$
Resultado: Individuo mutado

```
size = |individual|
affectedPositions = RandomInteger(minPositions, maxPosition)
startIndex = RandomInteger(0, size)
newValue = RandomInteger(0, |H|)
for i in {startIndex, ..., startIndex + affectedPositions} do
    individual_{i %|S|} = newValue
end
return individual
```

3.6.3. Fase de reemplazamiento

Durante la *fase de reemplazamiento* se seleccionan los individuos que sobrevivirán y conformarán la población de la próxima generación de entre el total de individuos disponibles, es decir, los individuos de la población actual y los descendientes generados durante la *fase de reproducción*. Para este proyecto se desarrollaron dos métodos de reemplazamiento, *New Individuals* y *Best Non Penalized (BNP)*.

El primero en implementarse fue *New Individuals*, Algoritmo 18, consiste en formar la población de la próxima generación a partir de **todos los descendientes** y el mejor individuo de la población actual. Introduciendo el mejor individuo de la población actual se asegura que el mejor individuo encontrado siempre esté en la población.

Tras la implementación del método de reemplazamiento *New Individuals* se desarrolló el método *Best Non Penalized*, Algoritmo 19. Éste, lleva a cabo un control explícito de la variedad a la hora de seleccionar los supervivientes para la próxima generación [15]. El control de la variedad se consigue evitando seleccionar los individuos que estén cerca de los que ya han sido seleccionados (supervivientes). La distancia ,cabe recordar que como medida de distancia se optó por la distancia *Hamming*, que establece la relación de cercanía entre un par de individuos se denomina **distancia de penalización** D.

Algoritmo 18: Método de reemplazamiento *New Individuals*

```
Datos: population, offsprings
Resultado: Población de la siguiente generación
bestIndividual = population[0]
for each individual in population do
    if bestIndividual.fitness > individual.fitness then
        bestIndividual = individual
    end
end
survivors = [bestIndividual]
for each offspring in offsprings do
    survivors.insert(offspring)
end
return survivors
```

Aquellos individuos que estén a una menor distancia de los supervivientes que D estarán **penalizados** y no podrán seleccionarse. La distancia de un individuo al conjunto de individuos supervivientes se establece como la mínima distancia existente entre el individuo y cada uno de los supervivientes. El método escogerá iterativamente el *mejor individuo no penalizado*. Cada vez que un individuo es seleccionado se deberá recalcular la distancia de todos los individuos aún no seleccionados al conjunto de supervivientes, penalizados o no. En el caso en el que todos los individuos no seleccionados estén penalizados, es decir, a una distancia menor que D, se seleccionará al individuo más alejado del conjunto de supervivientes.

Al comienzo del proceso de reemplazamiento se añade al conjunto de supervivientes el mejor individuo disponible, de esta manera se asegura preservar el mejor individuo encontrado en la población. Para permitir que el método explore en las primeras generaciones y a medida que avanza vaya intensificando, la distancia de penalización **disminuye linealmente** con el tiempo de ejecución hasta llegar a 0. El valor inicial de la distancia de penalización D_I se calcula a partir de la variedad media de la primera población M. Para controlar lo estricto que es el método a la hora de penalizar individuos se introduce un factor para la distancia de penalización inicial $f_{D_I}/f_{D_I} \in [0, 1]$. Cuanto menor sea el valor f_{D_I} más permisivo será el método penalizando:

$$D_I = f_{D_I} \cdot M$$

Algoritmo 19: Método de reemplazamiento *Best Non Penalized*

Datos: population, offsprings
Resultado: Población de la siguiente generación

```
1  popSize = |population|
2  individuals = population ∪ offsprings
3  penalized = []
4  bestIndividual = Individual with best fitness in individuals
5  individuals.remove(bestIndividual)
6  survivors = [bestIndividual]
7  D = D_I · (1 − ellapsedTime / totalTime)
8  while |survivors| < popSize do
9      for each ind in individuals do
10         ind.distance = Distance to the closest individual in survivors
11         if ind.distance < D then
12             individuals.remove(ind)
13             penalized.insert(ind)
14         end
15     end
16     if |individuals| = 0 then
17         selected = Individual with highest distance in penalized
18         survivors.insert(selected)
19     else
20         selected = Individual with best fitness in individuals
21         survivors.insert(selected)
22     end
23 end
24 return survivors
```

Capítulo 4

Estudio experimental

En este Capítulo de la memoria del Trabajo de Fin de Grado se expone el estudio experimental realizado para comprobar la calidad y validez de los métodos de resolución presentados en el Capítulo anterior. Además, se explica el origen y naturaleza de los datos con los que se realizaron los experimentos y se comentan las herramientas utilizadas durante el desarrollo del proyecto.

Cabe destacar que en el momento en el que se llevaron a cabo los primeros experimentos no se habían desarrollado todas las técnicas presentadas. Precisamente gracias a los resultados que se iban obteniendo, las características de las ejecuciones y las carencias que se detectaban en los métodos estudiados, era posible conocer con qué métodos y características se podrían mejorar los resultados obtenidos. De esta manera se consiguió ajustar y afinar la configuración de las pruebas hasta dar con las técnicas que mejores resultados obtienen.

Para cada experimento o grupo de pruebas realizado se presentan Cuadros y Figuras en las que se comparan y se ponen de manifiesto las diferencias de las técnicas de resolución del problema bajo estudio. También se realizan una serie de tests estadísticos para determinar cual de dichas técnicas se comporta mejor. Las ejecuciones se llevaron a cabo en tandas de 30.

4.1. Características de los datos

Los datos con los que se realizó el estudio experimental se obtuvieron del repositorio de *Loter* [12], que a su vez, fueron generados a partir de los conjuntos de datos que ofrece el proyecto internacional *HapMap* [6]. Los datos disponibles se correspondían a los haplotipos de los individuos de tres poblaciones distintas:

- Población Europea: 44 individuos, 88 haplotipos.
- Población Africana: 50 individuos, 100 haplotipos.
- Población Mexicana: 23 individuos, 46 haplotipos.

Para generar los haplotipos se utilizaron los primeros 50000 SNP de ambas copias del cromosoma uno de cada individuo. Para el estudio experimental se asumió que la población Mexicana se origina a partir de la *mezcla genética* de las poblaciones Europea y Africana. Teniendo esto en cuenta, podemos deducir que el conjunto de haplotipos ancestrales H estará compuesto por **188 haplotipos** y que el tamaño de los haplotipos, y por consiguiente el de las soluciones, es de **50000** elementos.

4.2. Herramientas utilizadas

Se decidió llevar a cabo las tareas de implementación usando el lenguaje *C++*. Como compilador se utilizó g++ (versión 4.7.2) especificando la versión del estándar como c++11 y con el nivel de optimización O3. La máquina en la que se ejecutaron los experimentos cuenta con 48 núcleos *AMD Opteron(tm) Processor 6348* de 2,8 GHz y 64 GB de memoria RAM.

El procedimiento para llevar a cabo las comparaciones entre técnicas una vez obtenidos los resultados, es decir, tanto el conjunto de tests estadísticos como el orden de ejecución de los mismos, se extrajo de la publicación "*Improving diversity in evolutionary algorithms: New best solutions for frequency assignment*" [16]. Debido al carácter estocástico de los algoritmos ejecutados, cada técnica a comparar se ejecuta al menos 30 veces. Con un nivel de significación del 5 %, se lanzan los tests sobre los valores objetivo de las mejores soluciones obtenidas en cada una de las ejecuciones. En primer lugar se ejecuta el test *Shapiro-Wilk* para confirmar si los valores siguen una distribución Gaussiana, en caso afirmativo, se aplica el test *Levene* para comprobar la homegeneidad de las varianzas. Si las muestras tienen varianzas iguales se ejecuta el test *ANOVA*, y si son distintas el test *Welch*. Para los valores que no sigan una distribución Gaussiana, se lanza el test no paramétrico *Kruskal-Wallis* con el propósito de determinar si las muestras pertenecen a la misma distribución.

4.3. Primer experimento

Para el primer experimento se decidió estudiar el comportamiento de un evolutivo básico y la búsqueda local iterada así como el impacto que ocasiona incluir el ajuste mencionado en la sección 3.6.1 para controlar la selección de padres en la *fase de selección* del evolutivo. Por cada técnica, se realizaron 30 ejecuciones de 6 horas cada una con valor $\lambda = 1$ y sobre un único haplotipo híbrido. En el Cuadro 4.1 puede verse la configuración de las pruebas realizadas.

Nombre	Método	Búsqueda local	Individuos	Selección	Cruce	Mutación / Perturbación	Reemplazamiento
Greedy ILS	Búsqueda local iterada	*GLS*	-	-	-	*Shake*	-
Promising ILS	Búsqueda local iterada	*PNLS*	-	-	-	*Shake*	-
Greedy Evolutionary	Evolutivo	*GLS*	30	*Binary Tournament*	*Two Point Crossover*	-	*New Individuals*
Greedy Evolutionary with clones	Evolutivo	*GLS*	30	*Binary Tournament with clones*	*Two Point Crossover*	-	*New Individuals*
Promising Evolutionary	Evolutivo	*PNLS*	30	*Binary Tournament*	*Two Point Crossover*	-	*New Individuals*
Promising Evolutionary with clones	Evolutivo	*PNLS*	30	*Binary Tournament with clones*	*Two Point Crossover*	-	*New Individuals*

Cuadro 4.1: Configuración de las pruebas del primer experimento.

Atendiendo a la Figura 4.1 y a los valores de los Cuadros 4.2 y 4.3, se puede extraer que para las configuraciones estudiadas el algoritmo evolutivo obtiene mejores resultados que la búsqueda local iterada utilizando la variante de búsqueda local *GLS*. El comportamiento de la variante de búsqueda local *GLS* es superior al de la variante *PNLS*. Esta afirmación se confirma si se atiende a los resultados obtenidos por la búsqueda local iterada con *GLS*, pues son superiores a los obtenidos por los evolutivos con *PNLS* y se presupone que el evolutivo es mejor técnica de resolución.

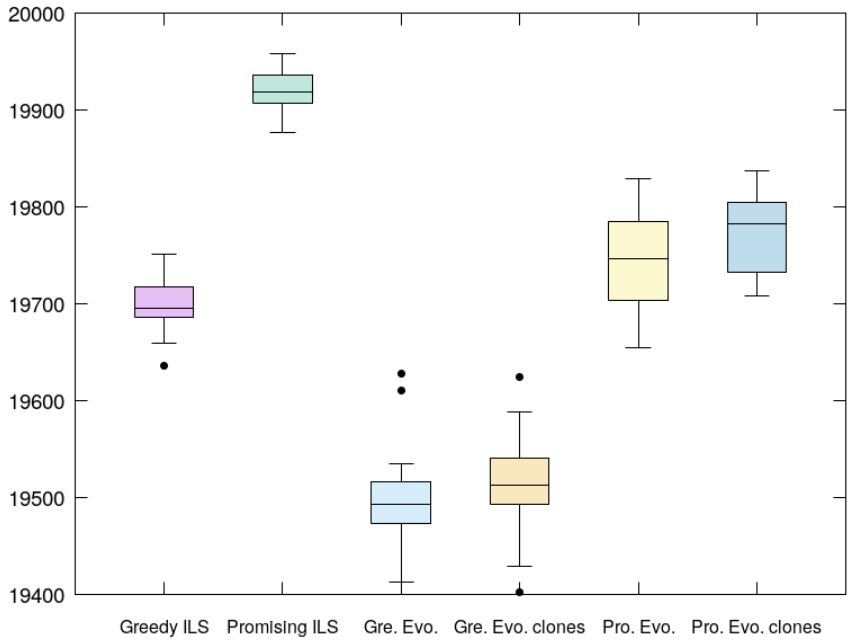


Figura 4.1: Boxplot de las mejores soluciones obtenidas por cada técnica del primer experimento. Los métodos evolutivos con *GLS* obtienen mejores resultados que el resto.

	Mínimo	Máximo	Media	Mediana	Desviación Típica
Greedy ILS	19636	19751	19701	19695	26.80
Promising ILS	19877	19958	19921.4	19919	20.01
Greedy Evolutionary	19413	19628	19495.3	19493	44.97
Greedy Evolutionary with clones	19402	19625	19514	19512	45.57
Promising Evolutionary	19655	19829	19746.1	19747	50.58
Promising Evolutionary with clones	19708	19837	19774	19782	37.40

Cuadro 4.2: Valores estadísticos asociados a las mejores soluciones obtenidas por cada prueba del primer experimento.

	Derrotas	Empates	Victorias	Puntuación
Greedy ILS	3	0	2	-1
Promising ILS	5	0	0	-5
Greedy Evolutionary	0	1	4	4
Greedy Evolutionary with clones	0	1	4	4
Promising Evolutionary	3	0	2	-1
Promising Evolutionary with clones	4	0	1	-3

Cuadro 4.3: Comparación estadística de las pruebas del primer experimento. Las técnicas ganadoras son *Greedy Evolutionary* y *Greedy Evolutionary with clones*.

Estudiando como varían los valores objetivo medios de las poblaciones de los algoritmos evolutivos, Figura 4.2, puede apreciarse una pérdida de mejora drástica después de los 5000 segundos de ejecución aproximadamente. La variedad media de dichas poblaciones, Figura 4.3, es nula tras 6000 segundos, es decir, la población ha alcanzado el estado de convergencia prematura y no puede evolucionar más. Así pues fue necesario incluir algún mecanismo de

control explícito de la variedad en la población del algoritmo evolutivo y por ello se llevó a cabo la implementación del método de reemplazamiento *Best Non Penalized (BNP)*.

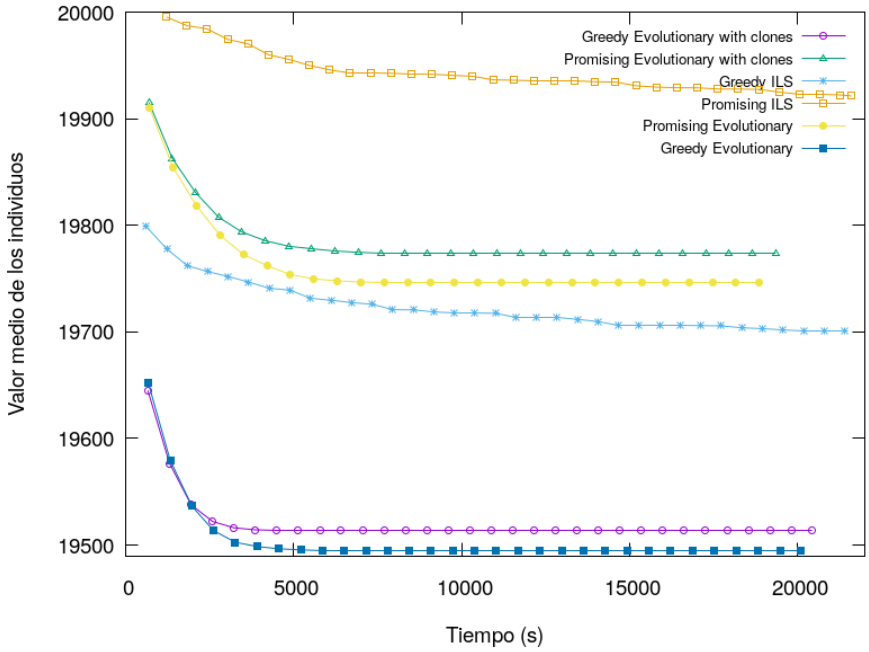


Figura 4.2: Valores medios las poblaciones de las técnicas del primer experimento.

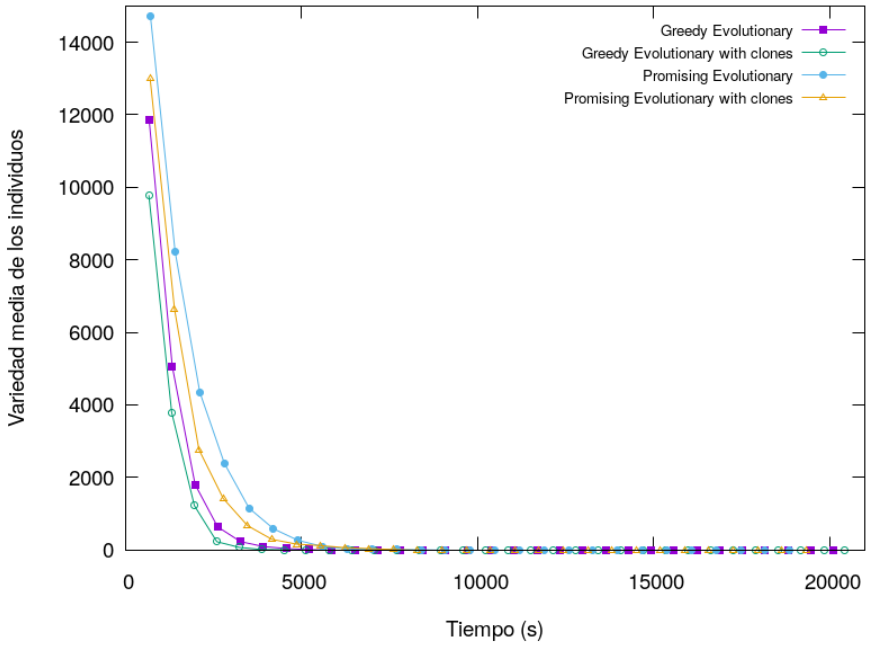


Figura 4.3: Variedad media de los algoritmos evolutivos del primer experimento. Es posible apreciar una rápida pérdida de variedad. Los métodos que permiten seleccionar al mismo individuo 2 veces en la misma pareja de padres pierden aún más rápido la variedad de la población.

A partir de los resultados de la comparación estadística y teniendo en cuenta la similitud de los resultados obtenidos por las técnicas *Greedy Evolutionary* y *Greedy Evolutionary with*

clones, se decidió llevar a cabo un total de 100 ejecuciones para cada uno de estos métodos con el propósito de contar con una muestra más representativa e intentar determinar si existe diferencia estadística entre ellos. El resultado fue negativo, **incluir el ajuste para evitar clones no es un cambio estadísticamente significativo**. A pesar de esto, se decidió incluir en la configuración de las pruebas de los demás experimentos realizados pues a pesar de no ser un cambio estadísticamente significativo, si que consigue mantener durante más tiempo una mayor variedad, Figura 4.3.

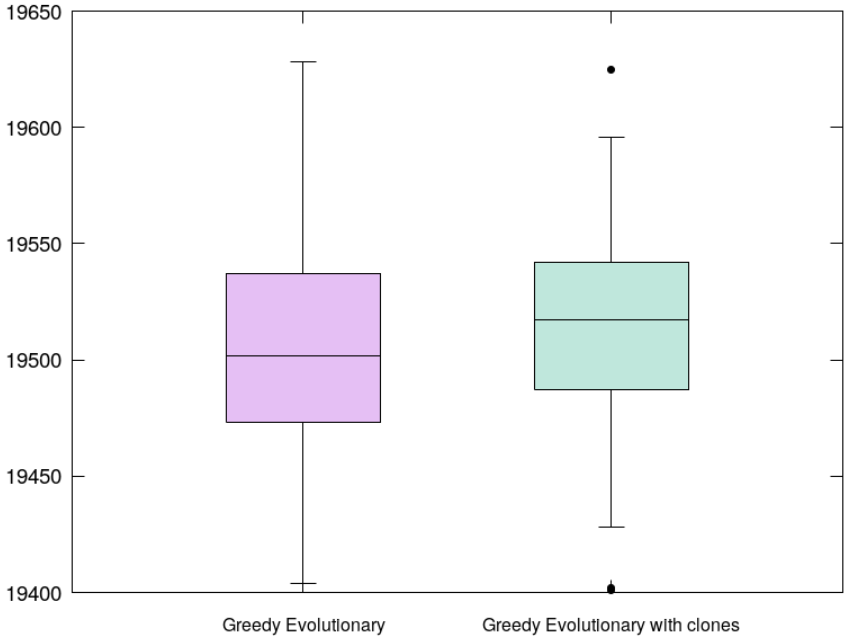


Figura 4.4: Boxplot de las mejores soluciones obtenidas por las técnicas *Greedy Evolutionary* y *Greedy Evolutionary with clones*.

	Mínimo	Máximo	Media	Mediana	Desviación Típica
Greedy Evolutionary	19404	19628	19504.2	19502	45.28
Greedy Evolutionary with clones	19401	19625	19512.9	19517	42.12

Cuadro 4.4: Valores estadísticos asociados a las ejecuciones de los métodos *Greedy Evolutionary* y *Greedy Evolutionary with clones*.

4.4. Segundo experimento

Como ya se ha comentado, tras el primer experimento se desarrolló el método de reemplazamiento *BNP* con el propósito de solucionar el problema de la rápida pérdida de variedad del algoritmo evolutivo. Para el segundo experimento se optó por incorporar el método *BNP* a la mejor técnica del primer experimento y comparar los resultados obtenidos con distintos valores del factor de la distancia de penalización inicial (f_{D_I}). Tras esto, se elegiría el valor que mejor resultados obtuviese para estudiar las consecuencias de incluir el control explícito de la variedad en el algoritmo evolutivo.

Seleccionar el valor de f_{D_I} adecuado es importante ya que dependerá de éste el funcionamiento del método y la calidad del control de la variedad. Cabe recalcar que en un método en el que se no se lleva un control explícito de la variedad, o en el que el **control es deficiente**, múltiples individuos terminarán en los mismos óptimos locales, que además se encuentran cercanos en el espacio de soluciones. Esto puede provocar que la población quede atrapada sin rango de mejora o con uno muy pequeño, debido a que los operadores de cruce o mutación no son lo suficientemente *contundentes* como para generar individuos que escapen de la zona del espacio de soluciones en el que la población se ha estancado. En los métodos con un buen control de la variedad, se asegura que la población esté dispersa por el espacio de soluciones, permitiendo generar individuos prometedores que se encuentren cerca de zonas de mayor calidad inexploradas.

Nombre	Método	Búsqueda local	Individuos	Selección	Cruce	Mutación	Reemplazamiento
Greedy Evolutionary BNP	Evolutivo	*GLS*	30	*Binary Tournament*	*Two Point Crossover*	-	*BNP*

Cuadro 4.5: Configuración de las primeras pruebas del segundo experimento.

Se probó el método *BNP* con la configuración ganadora del experimento previo, Cuadro 4.5, considerando $f_{D_I} = i \cdot 0{,}1 / \forall i \in \{0, \ldots, 10\}$. Por cada valor de f_{D_I} se realizaron 30 ejecuciones de 6 horas cada una con valor $\lambda = 1$ y sobre un único haplotipo híbrido.

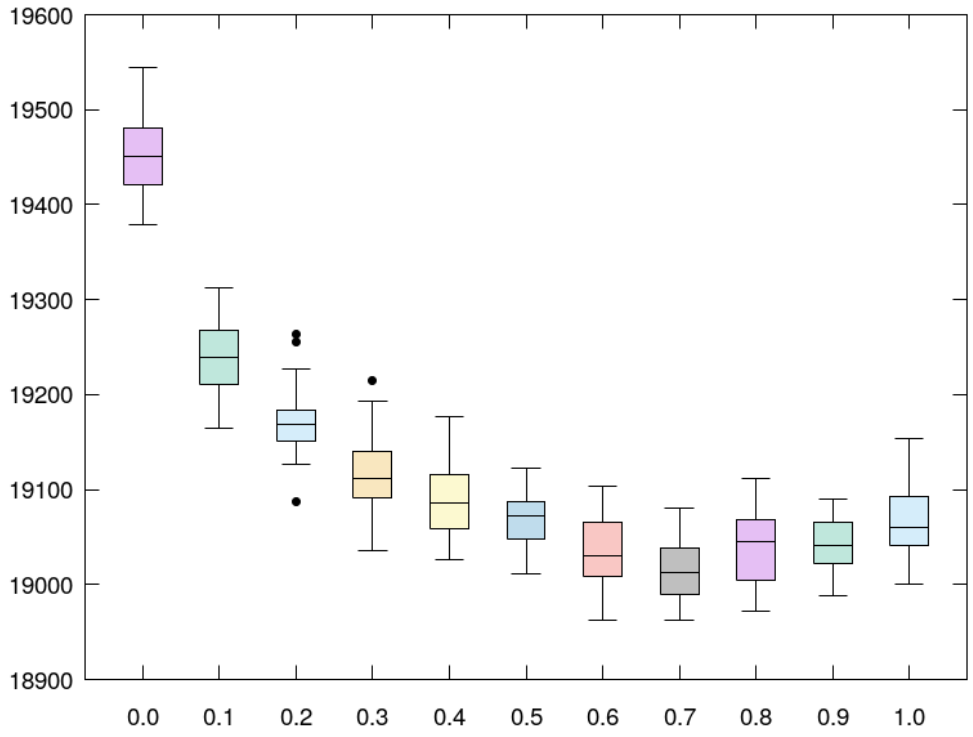


Figura 4.5: Boxplot de los resultados obtenidos por la técnica *Greedy Evolutionary BNP* para cada valor de f_{D_I}.

Como es posible apreciar en la Figura 4.5 y en el Cuadro 4.6, el valor 0.7 para el factor de la distancia de penalización inicial del método *Best Non Penalized* parece ser el mejor de entre

los valores estudiados. Los resultados de la comparación estadística, Cuadro 4.7, respaldan esta afirmación y por consiguiente, **el valor 0.7 es el seleccionado** para comparar los beneficios que se obtienen al incluir el control de la variedad en el algoritmo evolutivo.

	Mínimo	Máximo	Media	Mediana	Desviación Típica
Greedy Evolutionary BNP 0.0	19379	19544	19451.8	19450	36.44
Greedy Evolutionary BNP 0.1	19165	19313	19241.7	19239	37.02
Greedy Evolutionary BNP 0.2	19087	19264	19171.9	19168	35.76
Greedy Evolutionary BNP 0.3	19036	19215	19115.2	19112	39.31
Greedy Evolutionary BNP 0.4	19026	19177	19088.1	19086	34.26
Greedy Evolutionary BNP 0.5	19011	19122	19070.6	19072	30.813
Greedy Evolutionary BNP 0.6	18963	19103	19032.9	19030	34.37
Greedy Evolutionary BNP 0.7	18963	19081	19015.6	19012	29.69
Greedy Evolutionary BNP 0.8	18972	19112	19039.4	19045	42.48
Greedy Evolutionary BNP 0.9	18988	19090	19041.3	19041	27.20
Greedy Evolutionary BNP 1.0	19001	19154	19066.7	19059	36.25

Cuadro 4.6: Valores estadísticos asociados a las mejores soluciones obtenidas por la técnica *Greedy Evolutionary BNP* para cada valor de f_{D_I}.

	Derrotas	Empates	Victorias	Puntuación
Greedy Evolutionary BNP 0.0	10	0	0	-10
Greedy Evolutionary BNP 0.1	9	0	1	-8
Greedy Evolutionary BNP 0.2	8	0	2	-6
Greedy Evolutionary BNP 0.3	7	0	3	-4
Greedy Evolutionary BNP 0.4	6	0	4	-2
Greedy Evolutionary BNP 0.5	4	1	5	1
Greedy Evolutionary BNP 0.6	1	2	7	6
Greedy Evolutionary BNP 0.7	0	0	10	10
Greedy Evolutionary BNP 0.8	1	2	7	6
Greedy Evolutionary BNP 0.9	1	2	7	6
Greedy Evolutionary BNP 1.0	4	1	5	1

Cuadro 4.7: Comparación estadística de las pruebas del método *Greedy Evolutionary BNP* para cada valor de f_{D_I}, 0.7 es el valor óptimo para f_{D_I} de entre los valores estudiados.

En la Figura 4.6 se muestran los valores medios obtenidos durante la ejecución de cada técnica. Se puede ver como el valor de f_{D_I} afecta directamente la velocidad de mejora o evolución de la población. Para tamaños bajos de f_{D_I} la población mejora a un ritmo más lento que para tamaños comprendidos en el rango $[0{,}6, 0{,}9]$. Establecer f_{D_I} a 0 es equivalente a no llevar control sobre la variedad de la población, distancia de penalización igual a 0, y por consiguiente a no penalizar individuos. En este caso el método *BNP* se convierte en un método de reemplazamiento *elitista* que selecciona únicamente a los individuos con mejor *fitness*. A su vez, la Figura 4.7 presenta los valores de variedad media de las poblaciones para cada valor de f_{D_I}.

La Figura 4.8 muestra en detalle la distancia de penalización del método BNP durante la ejecución de las pruebas. Resulta interesante apreciar como las curvas que representan la variedad media de la población se adaptan a la distancia de penalización.

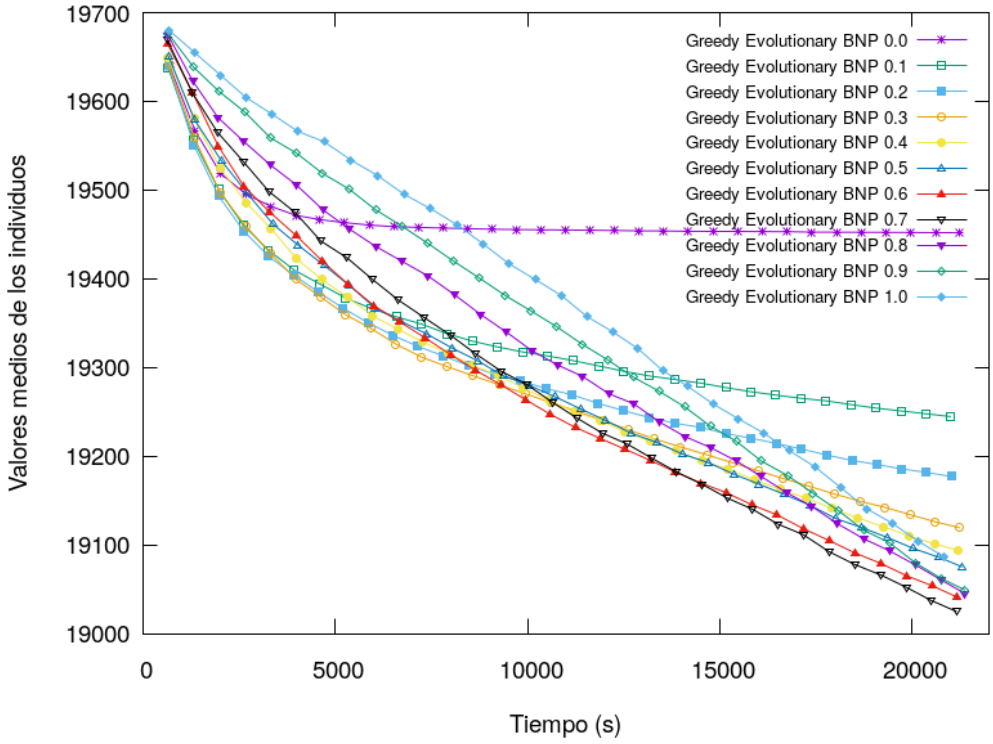


Figura 4.6: Valores medios de las poblaciones de *Greedy Evolutionary BNP* para cada valor de f_{D_I}.

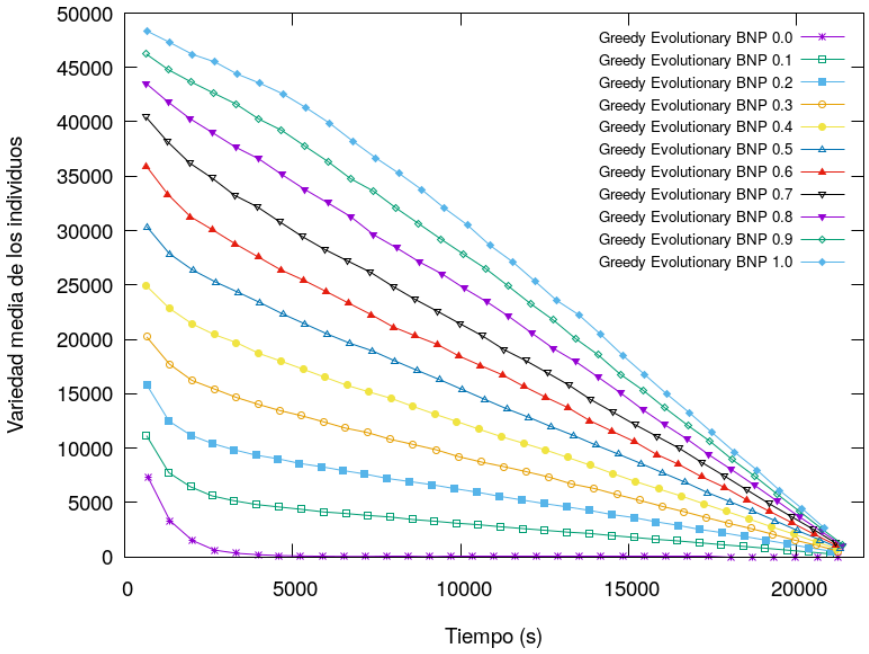


Figura 4.7: Variedad media de *Greedy Evolutionary BNP* para cada valor de f_{D_I}.

Tras conocer el mejor valor para el parámetro f_{D_I}, se propuso llevar a cabo una comparativa de *Greedy Evolutionary BNP* con las técnicas que obtuvieron mejores resultados en el primer experimento. De esta manera, se podrían medir las consecuencias de incluir el control de la variedad en el evolutivo. En el primer experimento las mejores técnicas fueron *Greedy Evolu-*

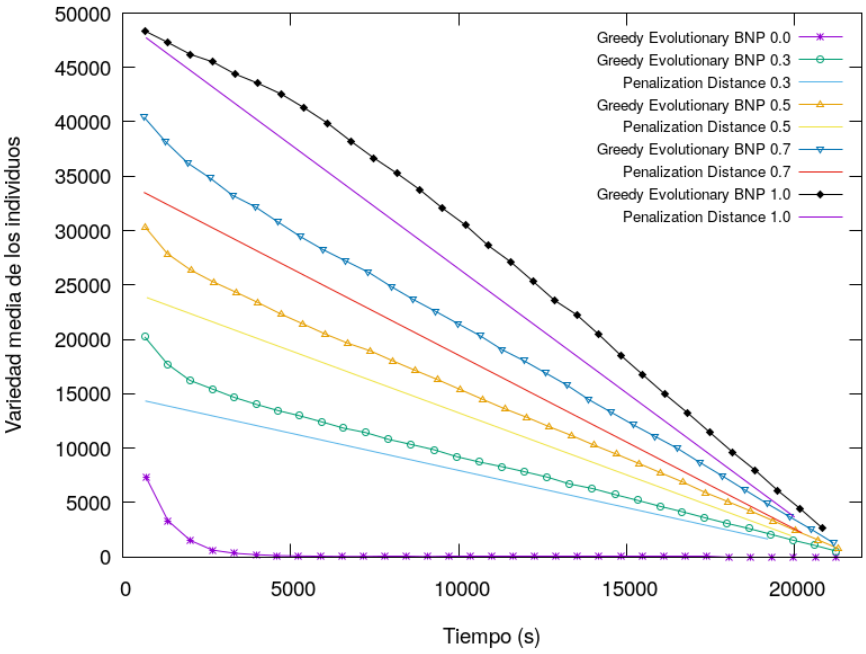


Figura 4.8: Variedad media de *Greedy Evolutionary BNP* con la distancia de penalización

tionary y *Greedy Evolutionary with clones* por lo que las pruebas se configuraron tal y como se muestra en el Cuadro 4.8.

Nombre	Método	Búsqueda local	Individuos	Selección	Cruce	Mutación	Reemplazamiento
Greedy Evolutionary BNP	Evolutivo	*GLS*	30	*Binary Tournament*	*Two Point Crossover*	-	*BNP*
Greedy Evolutionary	Evolutivo	*GLS*	30	*Binary Tournament*	*Two Point Crossover*	-	*New Individuals*
Greedy Evolutionary with clones	Evolutivo	*GLS*	30	*Binary Tournament with clones*	*Two Point Crossover*	-	*New Individuals*

Cuadro 4.8: Configuración de las pruebas realizadas para comparar los benificios de la inclusión del control de la variedad.

	Mínimo	Máximo	Media	Mediana	Desviación Típica
Greedy Evolutionary	19413	19628	19495.3	19493	44.97
Greedy Evolutionary with clones	19402	19625	19514	19512	45.57
Greedy Evolutionary BNP 0.7	18963	19081	19015.6	19012	29.69

Cuadro 4.9: Valores estadísticos asociados a las mejores soluciones obtenidas por las técnicas *Greedy Evolutionary*, *Greedy Evolutionary with clones* y *Greedy Evolutionary BNP* con $f_{D_I} =$ 0,7.

La comparación estadística, Cuadro 4.10, llevada a cabo demuestra que la inclusión de un control explícito de la variedad en el algoritmo evolutivo supone una gran mejora en el comportamiento y funcionamiento de la técnica. Atendiendo conjuntamente a las Figuras 4.10 y 4.11 se puede apreciar que el hecho de mantener y controlar la variedad de la población deriva

	Derrotas	Empates	Victorias	Puntuación
Greedy Evolutionary	1	1	0	-1
Greedy Evolutionary with clones	1	1	0	-1
Greedy Evolutionary BNP 0.7	0	0	2	2

Cuadro 4.10: Comparación estadística de las técnicas *Greedy Evolutionary*, *Greedy Evolutionary with clones* y *Greedy Evolutionary BNP* con $f_{D_I} = 0{,}7$.

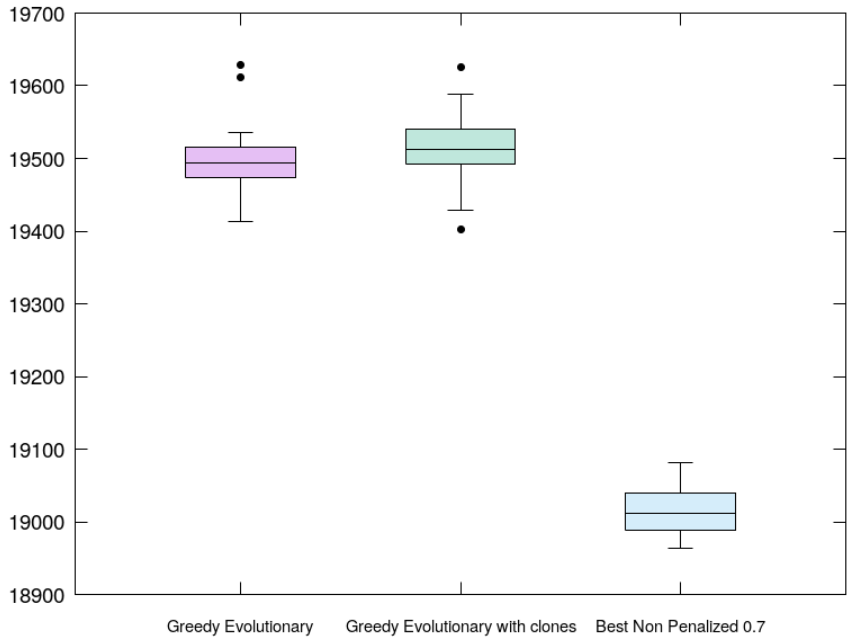


Figura 4.9: Boxplot de los resultados obtenidos por las técnicas comparadas para medir los beneficios del control de la variedad.

directamente en la posibilidad de ésta de seguir evolucionando. Como las poblaciones de las técnicas *Greedy Evolutionary* y *Greedy Evolutionary with clones* alcanzaron el estado de convergencia prematura, no pudieron mejorar más y se desperdició una gran parte del tiempo de ejecución total.

Tras haber completado el segundo experimento se optó por implementar el método exacto visto en la sección 3.3 con el objetivo de conseguir la solución óptima al problema para la entrada sobre la que se habían realizado las pruebas. Sabiendo el valor objetivo de la solución óptima se podrían comparar los resultados obtenidos hasta el momento, de esta manera se determinaría si las soluciones que estaban generando los métodos aproximados implementados eran de buena calidad.

El resultado fue desalentador pues la solución óptima para la instancia del problema sobre la que se habían ejecutado los experimentos tenía como **valor objetivo 2660**, un valor muy lejano al mejor alcanzado 18963, obtenido por la técnica *Greedy Evolutionary BNP* con $f_{D_I} = 0{,}7$ tras 6 horas de ejecución. Habiendo comprendido que las soluciones que se habían obtenido hasta el momento eran de muy baja calidad se decidió diseñar e implementar operadores específicos para el problema de la estimación de la ascendecia local.

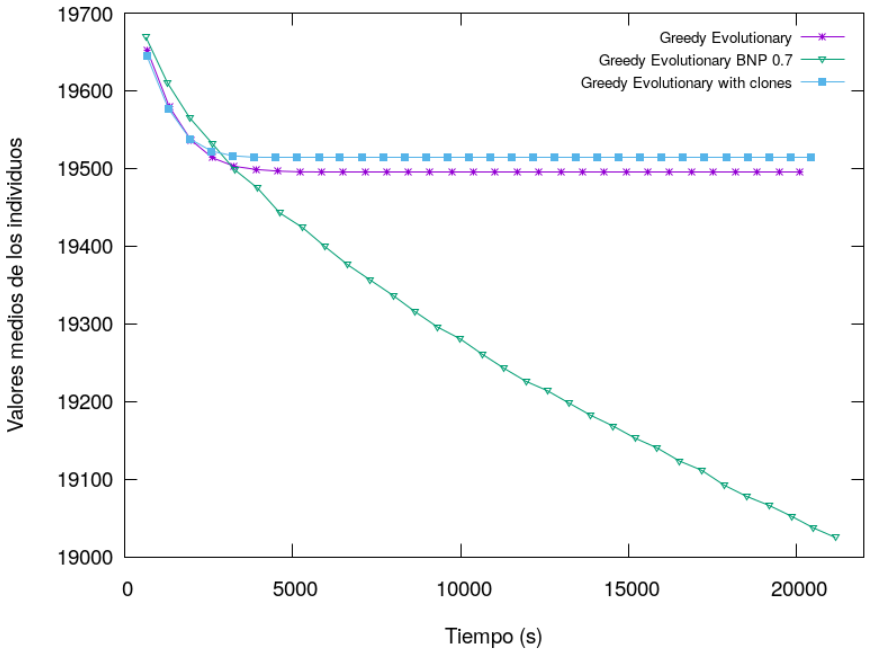


Figura 4.10: Valores medios de las poblaciones *Greedy Evolutionary*, *Greedy Evolutionary with clones* y *Greedy Evolutionary BNP* con $f_{D_I} = 0{,}7$.

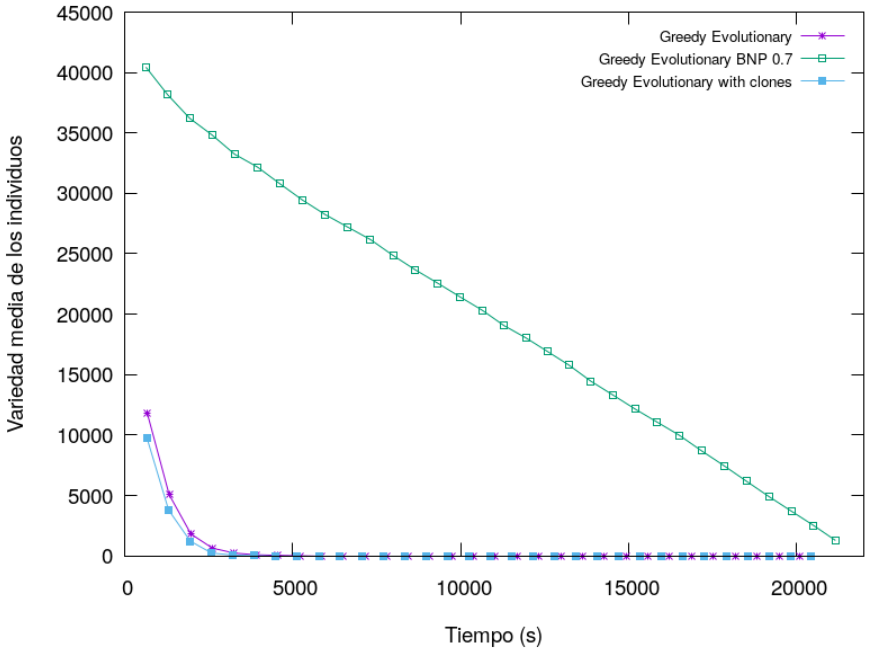


Figura 4.11: Variedad media*Greedy Evolutionary*, *Greedy Evolutionary with clones* y *Greedy Evolutionary BNP* con $f_{D_I} = 0{,}7$.

4.5. Tercer experimento

Tras conocer la baja calidad de las soluciones que estaban generando las técnicas desarrolladas se optó por implementar operadores específicos para el problema. A partir de este punto se desarrollaron la variante de búsqueda local *APLS* y el operador de mutación *Set Portion*. Se

decidió comparar el impacto del operador de mutación incluyéndolo en el evolutivo y también como fase de perturbación de la búsqueda local iterada. También se creyó conveniente observar el comportamiento del evolutivo al reducir el tamaño de la población. En el Cuadro 4.11 puede verse la configuración de las pruebas realizadas. Por cada técnica se realizaron 30 ejecuciones de 6 horas cada una con valor $\lambda = 1$ y sobre un único haplotipo híbrido. El valor de f_{D_I} se estableció a 0.7 por defecto. Las porciones de los individuos afectadas por el operador *Set Portion* se ajustan un tamaño comprendido en el intervalo $[10, 300]$.

Nombre	Método	Búsqueda local	Individuos	Selección	Cruce	Mutación / Perturbación	Reemplazamiento
Greedy Evolutionary BNP 30	Evolutivo	*GLS*	30	*Binary Tournament*	*Two Point Crossover*	*Set Portion*	*BNP*
Greedy Evolutionary BNP 10	Evolutivo	*GLS*	10	*Binary Tournament*	*Two Point Crossover*	*Set Portion*	*BNP*
APLS Evolutionary BNP 30	Evolutivo	*APLS*	30	*Binary Tournament*	*Two Point Crossover*	*Set Portion*	*BNP*
APLS Evolutionary BNP 10	Evolutivo	*APLS*	10	*Binary Tournament*	*Two Point Crossover*	*Set Portion*	*BNP*
APLS ILS	Búsqueda Local Iterada	*APLS*	-	-	-	*Set Portion*	-

Cuadro 4.11: Configuración de las pruebas del tercer experimento.

Atendiendo a los valores que se presentan en el Cuadro 4.13, se puede concluir que la inclusión de operadores específicos resultó ser un éxito. Contando con la misma cantidad de tiempo de ejecución, el valor objetivo de las soluciones obtenidas disminuyó drásticamente, acercándose al valor objetivo del óptimo global. La comparación estadística, Cuadro 4.12, muestra que la mejor de las técnicas estudiadas es, con diferencia, *APLS ILS*.

	Derrotas	Empates	Victorias	Puntuación
Greedy Evolutionary BNP 30	4	0	0	-4
Greedy Evolutionary BNP 10	3	0	1	-2
APLS Evolutionary BNP 30	2	0	2	0
APLS Evolutionary BNP 10	1	0	3	2
APLS ILS	0	0	4	4

Cuadro 4.12: Comparación estadística de las pruebas del tercer experimento. La técnica ganadora es *APLS ILS*.

Como puede apreciarse en la Figura 4.12, el comportamiento de las técnicas que usan *GLS* como variante de búsqueda local no es tan bueno como el de las técnicas que usan *APLS*. La inclusión del operador de mutación *Set Portion* hizo que los resultados obtenidos por el evolutivo con *GLS* y *BNP* mejorasen bastante. También se ha de tener en cuenta la significativa mejora conseguida al reducir el tamaño de la población de 30 individuos a 10. Con menos individuos se pueden llevar a cabo más generaciones en el mismo tiempo por lo que existe un mayor rango de mejora.

4.6. Pruebas finales

La búsqueda local iterada con *APLS* como variante de búsqueda local y *Set Portion* como fase de perturbación fue la mejor de las técnicas estudiadas, consiguiendo valores próximos al

	Mínimo	Máximo	Media	Mediana	Desviación Típica
Greedy Evolutionary BNP 30	14418	15412	14727.1	14716	207.50
Greedy Evolutionary BNP 10	13074	14443	13706.4	13720	249.36
APLS Evolutionary BNP 30	6868	7194	7044.23	7057	74.67
APLS Evolutionary BNP 10	5436	5621	5539.73	5548	52.52
APLS ILS	3686	3792	3737.73	3740	23.24

Cuadro 4.13: Valores estadísticos asociados a las mejores soluciones obtenidas por cada prueba del tercer experimento.

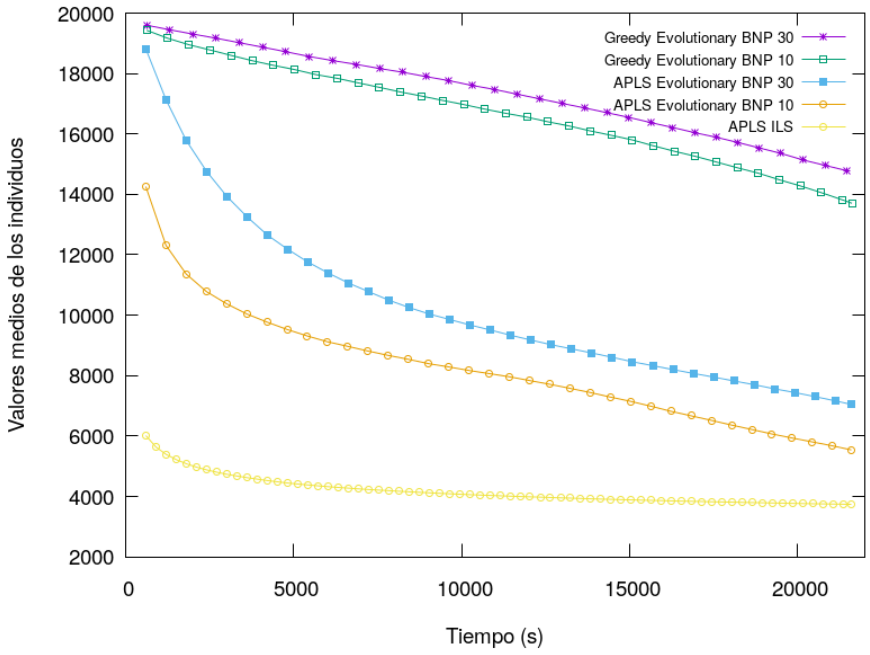


Figura 4.12: Valores medios las poblaciones de las técnicas del tercer experimento.

óptimo global. Por esta razón, se decidió realizar una última tanda de pruebas para distintos valores de λ y distintos haplotipos híbridos.

Lambda	Valor medio alcanzado	Valor óptimo medio	Diferencia media
1	3731,66	2666,66	1065
1.5	3921,66	2703,16	1218,50
2	4734,16	3827,66	906,50
2.5	4844,16	3841	1003,17
3	5407,33	4585	822,33

Cuadro 4.14: Resumen de los resultados obtenidos en las pruebas con distinto valor de λ y distintos haplotipos híbridos

El Cuadro 4.14 muestra los resultados obtenidos. Hay que destacar, que las pruebas se ejecutaron durante 6 horas cada una. Se estudió el comportamiento de la técnica con 6 haplotipos híbridos distintos para $\lambda = 1 + i \cdot 0{,}5 / \forall i \in \{0, \ldots, 4\}$. En cada fila del Cuadro 4.14 se exponen

los valores medios alcanzados, el valor óptimo medio y la diferencia media asociada a cada valor de λ. Se decidió agrupar los resultados por cada valor de λ y no por cada haplotipo híbrido ya que resulta más interesante de cara al estudio realizado. Como puede verse, la técnica bajo estudio obtiene resultados similares independientemente del valor de λ.

Capítulo 5

Conclusiones y líneas futuras

En este Capítulo se presentan las conclusiones alcanzadas, así como las posibles futuras líneas de investigación y experimentación

5.1. Conclusiones

Siguiendo la formulación del problema propuesta en *Loter* [2], se han diseñado e implementado diversas técnicas aproximadas para la resolución del problema de la estimación de la ascendencia local. Para comprobar la calidad y validez de los métodos de resolución desarrollados se llevó a cabo un estudio experimental.

Atendiendo a los resultados obtenidos por los experimentos realizados, la Búsqueda Local Iterada con la variante de búsqueda local *Adjacent Positions Local Search* y la fase de perturbación *Set Portion* ha resultado ser la mejor técnica de resolución de entre las estudiadas. Los resultados obtenidos por esta técnica se aproximan a los valores de los óptimos globales del problema, por lo que se puede deducir que la calidad de las soluciones que genera es alta.

Tanto las variantes de búsqueda local *GLS* y *PNLS* como la fase de perturbación *Shake* resultaron ser malas opciones para abordar el problema de la estimación de la ascendencia local. Las variantes de búsqueda local resultaron ser demasiado exhaustivas debido al tamaño de la vecindad a explorar. El método *Shake* introducía en las soluciones una gran cantidad de valores adyacentes distintos, algo que penaliza la función de evaluación.

Con el método *New Individuals* para la *fase de reemplazamiento* del evolutivo se alcanzaba el estado de convergencia prematura con mucha rapidez. Precisamente por este motivo se incluyó un mecanismo de control explícito de la variedad por medio del método *Best Non Penalized* [15]. Mediante el ajuste del parámetro f_{D_I} se dio con la configuración óptima que conseguía mantener un buen compromiso entre la variedad y la evolución de la población.

La inclusión de operadores diseñados específicamente para el problema marcó un antes y un después en lo relativo a la idoneidad de las técnicas de resolución implementadas. Tanto la variante de búsqueda local *APLS* como el operador de mutación *Set Portion* influyeron en la obtención de resultados muy superiores a los obtenidos previamente, diseñar e incluir estos operadores resultó todo un éxito de cara al estudio experimental. Si bien es cierto que el método que reportó mejores resultados fue la búsqueda local iterada, el algoritmo evolutivo con los operadores específicos también obtuvo buenos resultados.

En conclusión, se ha demostrado mediante el estudio experimental que es viable abordar el problema de la estimación de la ascendencia local usando técnicas aproximadas. Es posible

conseguir soluciones de calidad evitando los problemas de escalabilidad que surgen al aplicar métodos exactos. Existe un amplio margen de mejora en cuanto al estudio, investigación y selección de las técnicas aproximadas a aplicar, en el siguiente apartado se abordan algunas propuestas.

5.2. Líneas futuras

Para continuar con el estudio llevado a cabo en este proyecto existen varias propuestas con las que, a priori, se obtendrían mejores resultados. Realizar ejecuciones más largas de las técnicas que mejor comportamiento mostraron durante el estudio experimental debería reportar soluciones de mayor calidad.

Otro aspecto a valorar supone experimentar combinando el uso de técnicas aproximadas para ver el comportamiento general, por ejemplo, se podría ejecutar una búsqueda local iterada de pocas iteraciones con cada descendiente generado en la *fase de reproducción* del evolutivo. Paralelizar las búsquedas locales (o si se diese el caso, las búsquedas locales iteradas) que se aplican a los descendientes recién generados en la *fase de reproducción* supondría un gran ahorro computacional por cada generación del algoritmo evolutivo. Otra posibilidad a tener en cuenta es la de seguir diseñando e implementando nuevos operadores específicos del problema para compararlos con los ya desarrollados y ver cual de ellos obtiene mejores resultados.

Intentar resolver el problema de la estimación de la ascendencia local haciendo uso de otro tipo de técnicas aproximadas también debería ser objeto de estudio. A pesar de las numerosas alternativas que existen para continuar el estudio presentado en esta memoria, los experimentos realizados durante el desarrollo de este proyecto suponen un punto de partida para futuras investigaciones.

Capítulo 6

Summary and Conclusions

This Chapter presents the conclusions, as well as some research and experimentation proposals to take into account in the future.

6.1. Conclusions

The formulation of the problem proposed in *Loter* [2], several approximate techniques have been designed and implemented to solve the local ancestry problem. To verify the quality of the resolution methods developed, an experimental study was carried out.

According to the results obtained in the experimental study, Iterated Local Search with *Adjacent Positions Local Search* as local search variant and *Set Portion* as perturbation method is the best resolution method within the studied techniques. The results obtained by this technique are close to the values of the global optimum of the problem, therefore, the quality of the solutions it generates is high.

GLS and *PNLS* local search variants as well as *Shake* perturbation method showed low performance level solving local ancestry inference problem. *GLS* and *PNLS* are too exhaustive local searchs due to the large size of the neighbourhood that have to be explored. *Shake* method insert into the solutions too much unequal adjacent values that are penalized by the objective function.

Using *New Individuals* method as replacement phase of the evolutionary algorithm was causing premature convergence quickly. To avoid this issue, a mechanism to explicitly control the mean variety of the population was introduced. *Best Non Penalized* method was the solution to this inconvenient. Adjusting the value of f_{D_I} a balance between the control of the variety and the evolution of the population was reached.

The addition of operators specifically designed to solve the local ancestry problem caused a big impact. The quality of the solutions obtained by the techniques highly improved when *APLS* local search variant and *Set Portion* mutation operator were introduced. Design and develop this operators was a huge success. Even though iterated local search was the better technique, evolutionary algorithm with specific operators also obtained high quality solutions.

In conclusion, the experimental study carried has proven the viability of solving local ancestry inference problem using approximate methods. It is possible to obtain high quality solutions avoiding scalability problems that arise when an exact method is applied. Several improvements could be made in the study, research and selection of approximate methods to solve the problem, the next section presents some proposals.

6.2. Future work

In order to continue the study of this Final Degree Project there are several proposals to take into consideration to improve the results. Performing longer executions of techniques that showed better behaviour during experimental study should obtain higher quality solutions.

Another aspect to take into account involves combining tecniques within the same experiments to observe the global behaviour, for example, instead of a local search, an iterated local search could be run after the generation of each offspring during the reproduction phase of the evolutionary algorithm. Parallelising the local searches or iterated local searches carried after the generation of the offsprings could lead to a huge computational effort saving on each generation. Designing and implementing new specific operators of the problem could be considered as another possibility.

Attempting to solve the local ancestry inference problem with other kind of approximate techniques should be considered. Even though the numerous improvements and alternatives that could be taken to continue this study, the experiments carried out during the development of this project are a starting point for future research.

Capítulo 7

Presupuesto

En este Capítulo se expone una estimación del presupuesto del proyecto. El Cuadro 7.1 muestra el coste de cada una de las actividades de investigación, implementación y análisis llevadas a cabo. En el Cuadro 7.2 se exponen los costes asociados a la realización de las pruebas de cada uno de los experimentos realizados en el estudio experimental. Por último, el Cuadro 7.3 presenta el presupuesto total estimado.

Actividad	Precio / Hora	Horas invertidas	Coste
Investigación del problema	12 €	20	240 €
Estudio del estado del arte	12 €	30	360 €
Selección de técnicas a desarrollar	12 €	20	240 €
Implementación de las técnicas	16 €	50	800 €
Preparación de experimentos	14 €	20	280 €
Tratamiento de los resultados obtenidos	12 €	30	360 €
Análisis de los resultados	14 €	30	420 €
Total	-	200	2700 €

Cuadro 7.1: Presupuesto estimado de cada actividad.

Actividad	Precio / Hora (32 núcleos)	Horas invertidas	Coste
Primer experimento	1.29 €	72	92.88 €
Segundo experimento	1.29 €	66	85.14 €
Tercer experimento	1.29 €	30	38.7 €
Pruebas finales	1.29 €	6	7.74 €
Total	-	174	224,46 €

Cuadro 7.2: Presupuesto del estudio experimental. Precio de la máquina Google Cloud n1-highcpu-32.

Presupuesto total	2924.46 €

Cuadro 7.3: Presupuesto total del proyecto.

Bibliografía

[1] Yael Baran, Bogdan Pasaniuc, Sriram Sankararaman, Dara G Torgerson, Christopher Gignoux, Celeste Eng, William Rodriguez-Cintron, Rocio Chapela, Jean G Ford, Pedro C Avila, et al. Fast and accurate inference of local ancestry in latino populations. *Bioinformatics*, 28(10):1359–1367, 2012.

[2] Thomas Dias-Alves, Julien Mairal, and Michael GB Blum. Loter: A software package to infer local ancestry for a wide range of species. *Molecular biology and evolution*, 35(9):2318–2326, 2018.

[3] Gennady Ermak. *Emerging medical technologies*. World Scientific Publishing Company, 2015.

[4] Giulio Genovese, Robert E Handsaker, Heng Li, Nicolas Altemose, Amelia M Lindgren, Kimberly Chambert, Bogdan Pasaniuc, Alkes L Price, David Reich, Cynthia C Morton, et al. Using population admixture to help complete maps of the human genome. *Nature genetics*, 45(4):406, 2013.

[5] Clive J Hoggart, MARK D Shriver, Rick A Kittles, David G Clayton, and Paul M McKeigue. Design and analysis of admixture mapping studies. *The American Journal of Human Genetics*, 74(5):965–978, 2004.

[6] National Human Genome Research Institute. Proyecto *HapMap*. `https://www.genome.gov/10001688/international-hapmap-project`, 2002. [Online; accessed 7-June-2019].

[7] Wenfei Jin, Shuhua Xu, Haifeng Wang, Yongguo Yu, Yiping Shen, Bailin Wu, and Li Jin. Genome-wide detection of natural selection in african americans pre-and post-admixture. *Genome research*, 22(3):519–527, 2012.

[8] Mark Jobling, Matthew Hurles, and Chris Tyler-Smith. *Human evolutionary genetics: origins, peoples & disease*. Garland Science, 2013.

[9] Jeffrey M Kidd, Simon Gravel, Jake Byrnes, Andres Moreno-Estrada, Shaila Musharoff, Katarzyna Bryc, Jeremiah D Degenhardt, Abra Brisbin, Vrunda Sheth, Rong Chen, et al. Population genetic inference from personal genome data: impact of ancestry and admixture on human genomic variation. *The American Journal of Human Genetics*, 91(4):660–671, 2012.

[10] Na Li and Matthew Stephens. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165(4):2213–2233, 2003.

[11] Brian K Maples, Simon Gravel, Eimear E Kenny, and Carlos D Bustamante. Rfmix: a discriminative modeling approach for rapid and robust local-ancestry inference. *The American Journal of Human Genetics*, 93(2):278–288, 2013.

[12] Mathematical and Computational Biology Research Group at Université Grenoble Alpes (France). *Loter* github repository. `https://github.com/bcm-uga/Loter`, 2018. [Online; accessed 7-June-2019].

[13] Badri Padhukasahasram. Inferring ancestry from population genomic data and its applications. *Frontiers in genetics*, 5:204, 2014.

[14] Alkes L Price, Arti Tandon, Nick Patterson, Kathleen C Barnes, Nicholas Rafaels, Ingo Ruczinski, Terri H Beaty, Rasika Mathias, David Reich, and Simon Myers. Sensitive detection of chromosomal segments of distinct ancestry in admixed populations. *PLoS genetics*, 5(6):e1000519, 2009.

[15] Emmanuel Romero Ruiz and Carlos Segura. Memetic algorithm with hungarian matching based crossover and diversity preservation. *Computación y Sistemas*, 22(2), 2018.

[16] Carlos Segura, Arturo Hernández-Aguirre, Francisco Luna, and Enrique Alba. Improving diversity in evolutionary algorithms: New best solutions for frequency assignment. *IEEE Transactions on Evolutionary Computation*, 21(4):539–553, 2016.

[17] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[18] Timothy A Thornton and Justo Lorenzo Bermejo. Local and global ancestry inference and applications to genetic association analysis for admixed populations. *Genetic epidemiology*, 38(S1):S5–S12, 2014.

[19] Bridgett M vonHoldt, Roland Kays, John P Pollinger, and Robert K Wayne. Admixture mapping identifies introgressed genomic regions in north american canids. *Molecular ecology*, 25(11):2443–2453, 2016.

[20] EJ Wood. The encyclopedia of molecular biology: Editor in chief, sir john kendrew, executive editor, eleanor lawrence. pp 1165. blackwell science, oxford. 1994.£ 99.50. *Biochemical Education*, 23(2):105–105, 1995.

[21] Kai Yuan, Ying Zhou, Xumin Ni, Yuchen Wang, Chang Liu, and Shuhua Xu. Models, methods and tools for ancestry inference and admixture analysis. *Quantitative Biology*, 5(3):236–250, 2017.