Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo Fin de Grado

Grado en Ingeniería Informática

Genómica Nutricional y Algoritmos Evolutivos Multiobjetivo: aproximaciones con METCO

Nutritional Genomics and Multiobjective Evolutionary Algorithms: approaches with METCO .

Pamela Jiménez Rebenaque

La Laguna, 30 de junio de 2018

Dra. **Coromoto León Hernández**, con N.I.F. 78.605.216-W profesora Titular de Universidad del área de Lenguajes y Sistemas Informáticos adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora y

Dr. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z profesor Asociado adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

"*Genómica Nutricional y Algoritmos Evolutivos Multiobjetivo: aproximaciones con METCO.*"

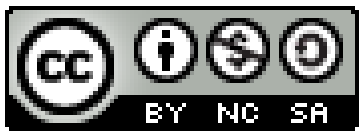
ha sido realizada bajo su dirección por D.ª **Pamela Jiménez Rebenaque**, con N.I.F. 79.153.153-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de junio de 2018

Agradecimientos

A la tutora y cotutor por su gran ayuda durante todo el proceso de desarrollo de este Trabajo Fin de Grado.

Licencia



Resumen

El objetivo de este Trabajo Fin de Grado es diseñar e implementar algoritmos evolutivos multiobjetivo para encontrar patrones en secuencias de proteínas que permitan descubrir y clasificar biomarcadores de enfermedades dietéticas.

Para ello, se ha utlizado la herramienta Metaheuristic-based Extensible Tool for Cooperative Optimisation (METCO), que implementa distintos algoritmos multiobjetivo que permiten resolver el problema de la búsqueda de patrones. Para la utilización de esta herramienta, se ha implementado una clase que modela el problema, y se han diseñado una serie de experimentos con los algoritmos Non-dominated Sorting Genetic Algorithm II (NSGA-II) y Strength Pareto Evolutionary Algorithm 2 (SPEA2). Por último, se ha llevado a cabo un análisis de los resultados obtenidos en los experimentos sin que se puedan extraer conclusiones sólidas sobre qué algoritmo se comporta mejor.

Palabras clave: proteínas, búsqueda, patrones, multiobjetivo, NSGA-II, SPEA2, nutrigenómica, nutrignética, algoritmo, dieta, enfermedad, optimización

Abstract

The main objective of this project is to design and implement multiobjective evolutionary algorithms to find patterns in protein sequences that allow dietary diseases biomarkers to be discovered.

To do this, the Metaheuristic-based Extensible Tool for Cooperative Optimisation (METCO) tool, that implements different multiobjective evolutionary algorithms that are able to solve the Pattern Finding problem, has been used. To use this tool, a class that models the problem has been implemented and several experiments with the algorithms Non-dominated Sorting Genetic Algorithm II (NSGA-II) and Strength Pareto Evolutionary Algorithm 2 (SPEA2), have been designed. Furthermore, the analysis of the results obtained from the experiments has been carried out. Nevertheless, sound conclusions about which of both algorithms provides better performance could not be extracted.

Keywords: *proteins, search, pattern, finding, multiobjective, NSGA-II, SPEA2, nutrigenomics, nutrigenetics, algorithm, diet, disease, optimization*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se detalla la motivación y objetivos de este trabajo, así como la metodología empleada y el plan de trabajo propuesto.

1.1. Motivación

Con el avance de las últimas técnicas en genética, el ácido desoxirribonucleico (ADN) ha tomado un papel relevante en las disciplinas relacionadas con la salud. Una de ellas es la nutrición. La genómica nutricional se encarga de estudiar la relación entre el genoma y la nutrición, así como sus efectos en la salud. Esta disciplina se divide a su vez en dos ramas: la nutrigenómica y la nutrigenética.

La nutrigenómica estudia los efectos de los nutrientes en la salud a través de la modificación que generan en el genoma y su manera de expresarse. Esto es, pretende proporcionar un conocimiento molecular sobre los componentes de la dieta que contribuyen a la salud mediante la alteración de la expresión y constitución genética individual. De esta manera, la nutrigenómica tiene un carácter teórico, buscando patrones y definiendo diversos aspectos de la genómica nutricional.

La nutrigenética estudia los efectos de las variaciones genéticas en la interacción con la dieta. Es decir, investiga cómo las distintas variantes genéticas de las personas influyen en el metabolismo de los nutrientes, la dieta y las enfermedades asociadas a esta. Así, la nutrigenética tiene un carácter más aplicado, tratando de dar recomendaciones referentes a los riesgos y a los beneficios de dietas concretas o de los componentes dietéticos aislados.

En ambos casos, tanto en la nutrigenómica como en la nutrigenética, surgen problemas en los que se debe realizar una búsqueda de patrones.

1.2. Objetivos

El objetivo de este Trabajo Fin de Grado es diseñar e implementar algoritmos evolutivos multi-objetivo para encontrar patrones en secuencias de proteínas que permitan descubrir y clasificar biomarcadores de enfermedades dietéticas.

Para ello, se utilizarán los algoritmos disponibles en la herramienta Metaheuristic-based Extensible Tool for Cooperative Optimisation (METCO), con el fin de determinar qué algoritmo tiene mejor comportamiento para este problema. Por último, se compararán los resultados con los de otros algoritmos y experimentos ya realizados.

1.3. Metodologías y Plan de Trabajo

Durante el desarrollo de este Trabajo Fin de Grado se diferenciará entre la metodología general de investigación y la metodología empleada para la implementación de la solución del problema.

La metodología de investigación seguida será la de una investigación cuantitativa con experimentos computacionales marcada por las siguientes fases:

1. Fase conceptual; subdividida de la siguiente forma:
   a) Formulación y delimitación del problema a tratar.
   b) Revisión de la literatura asociada y del estado actual.
   c) Contrucción del marco teórico.
   d) Formulación de hipótesis.
2. Fase de diseño e implementación. En esta fase se llevará a cabo el diseño e implementación de la solución del problema.
   a) Diseño del individuo.
   b) Implementación del individuo.
3. Fase de formulación y diseño. En esta fase se definen los experimentos a realizar:
   a) Selección de las instancias del problema que se van a estudiar.
   b) Diseño de los experimentos.
4. Fase empírica. Fase en la que se realizan los experimentos y se recogen sus resultados:
   a) Realización de los experimentos diseñados.
   b) Recolección de los resultados de los experimentos.
   c) Preparación de los datos para realizar un análisis.
5. Fase analítica. Comprende los procesos de análisis de los datos obtenidos:
   a) Análisis de los datos.
   b) Comparativa con experimentos anteriores.
   c) Interpretación de los resultados del análisis y comparativa.
6. Fase de difusión. Esta fase está destinada a la redacción y publicación de la investigación realizada.

Por otro lado, la metodología utilizada para el desarrollo e implementación del problema se basará en las directrices de metodologías de desarrollo de software ágiles, los principios *Single responsibility, Open-closed, Liskov substitution, Interface segregation* y *Dependency inversion* (SOLID) de la programación orientada a objetos y la filosofía de código abierto.

Además, en el diseño e implementación del individuo que representa una solución del problema, se tendrá en cuenta la compatibilidad con METCO.

En lo referente al plan de trabajo para este proyecto, se diseñará considerando los objetivos SMART, siglas en inglés para específico, medible, alcanzable, relevante y con tiempo limitado (*Specific-Measurable-Attainable-Relevant-Timely*). Por ello, se definirán las tareas a realizar dentro del espacio de tiempo desde el comienzo del proyecto, hasta la entrega del mismo. Así, el plan de trabajo propuesto es el siguiente:

1. Revisión bibliográfica (1 mes - Febrero a Marzo)
2. Diseño de la solución del problema (1 mes - Marzo a Abril)
3. Implementación de una solución (1 mes - Abril a Mayo)
4. Verificación de los resultados (15 días - Mayo)
5. Difusión (Mayo a Junio)

El contenido de esta memoria está distribuido de la siguiente manera. En el Capítulo 2 se describen los antecedentes y el estado actual del tema. La formulación, descripción e implementación del problema de la búsqueda de patrones se encuentra en el Capítulo 3. El Capítulo 4 contiene el diseño de los experimentos y los datos recogidos. Las conclusiones sobre estos experimentos se encuentran en los Capítulos 5 y 6 (en inglés). Por último, el Capítulo 7 contiene el presupuesto del proyecto.

Capítulo 2

Antecedentes

En este capítulo se hablará sobre los antecedentes y el estado actual del tema, incluyendo una descripción detallada de los artículos que han servido como base de este trabajo.

2.1. Metodología

Se ha seguido el curso de la biblioteca para la búsqueda de información contrastada en las bases de datos de Ingeniería.

Partiendo de .Q [9], se han realizado búsquedas en *Web Of Science* [12], Scopus [10], *Association for Computing Machinery* (ACM) [1], y *Institute of Electrical and Electronics Engineers* (IEEE) Xplore [3] con el fin de encontrar artículos relevantes para este proyecto.

2.2. Estado Actual

Los antecedentes de este trabajo incluyen los siguientes artículos:

Nutrigenomics: goals and strategies. **Michael Muller, Sander Kersten.** [18]

La nutrigenómica permitirá entender cómo influye la nutrición en las rutas metabólicas y el control homeostático, como se perturba esta regulación en las fases tempranas de una enfermedad dietética y cómo contribuye el genotipo a estas enfermedades. Las herramientas de la genómica se pueden aplicar mediante dos estrategias complementarias. La primera consiste en identificar los genes y proteínas cuya expresión puede ser influenciada por los nutrientes, lo que permite, a su vez, identificar las rutas reguladoras a través de las cuales la dieta influye en la homeostasis. La segunda consiste en catalogar las diferentes firmas de genes, proteínas y metabolitos que están asociadas con nutrientes específicos, lo que puede proveer con biomarcadores a nivel molecular que adviertan con antelación de cambios en la homeostasis debidos a nutrientes. En definitiva, la nutrigenómica permitirá aplicar estrategias efectivas de intervención dietética, ya que el objetivo principal de la aplicación de la genómica en la nutrición debería ser la prevención de enfermedades relacionadas con la dieta.

Gene selection heuristic algorithm for nutrigenomics studies. **D. Valour, I. Hue, B. Grimard, B. Valour.** [19]

El objetivo del algoritmo heurístico descrito es la búsqueda de conexiones entre el transcriptoma y el metaboloma. Para ello, extiende el clásico análisis de correlación canónica (CCA) para un número

mayor de variables (sin regularización). Para empezar, se resumen modelos CCA en matrices PageRank, cuyo producto da lugar a una matriz estocástica que continua el camino autoevitante que cubre el algoritmo. Luego, un proceso de Markov homogéneo aplicado a esta matriz converge las probabilidades de interconexión entre genes, dando lugar a una selección de subconjuntos disjuntos de genes. Por último, cada subconjunto de genes es enlazado con el conjunto de datos metabólicos o clínicos que representa el fenotipo biológico de interés.

***Online Tools for Bioinformatics Analyses in Nutrition Sciences*. Sridhar A. Malkaram, Yousef I. Hassan, Janos Zempleni.** [17]

Tras la revolución del Proyecto Genoma Humano, se han creado diversas bases de datos, tanto aquellas de mayor tamaño creadas por consorcios y centros, como otras de menor tamaño creadas por investigadores individuales. A su vez, se han creado un número de herramientas de bioinformática para la minería de esas bases de datos. El objetivo de este artículo es mostrar los recursos de bioinformática (bases de datos de transcriptomas, proteomas, epigenomas, metabolomas, etc.) disponibles en el dominio público, que pueden ser utilizados como punto de partida para las investigaciones en este campo.

***A hybrid MPI/OpenMP parallel implementation of NSGA-II for finding patterns in protein sequences*. David L. González-Álvarez, Miguel A. Vega-Rodríguez, Álvaro Rubio-Largo.** [13]

Descubrimiento de patrones comunes repetidos como un problema de optimización multiobjetivo mediante una aproximación MPI/OpenMP híbrida que paraleliza un algoritmo metaheurístico conocido, el NSGA-II.

En conclusión, el artículo [18] ha sido utilizado para determinar qué es la nutrigenómica, el [17] para recopilar las posibles herramientas a utilizar, y los artículos [19] y [13] para plantear la formulación del problema.

Capítulo 3

Problema búsqueda de patrones

En este capítulo se procederá a explicar de manera detallada el problema de la búsqueda de patrones. Para ello, se comenzará dando una descripción teórica del problema, para luego explicar la implementación que se ha llevado a cabo.

3.1. Formulación del problema

La formulación del problema se basa en la utilizada en el artículo [13]. Se ha formulado el problema de encontrar patrones comunes como un problema de optimización multiobjetivo (MOP, de sus siglas en inglés Multi-objective Optimisation Problem).

Un MOP general se define como minimizar (o maximizar):

$$F(x) = (f_1(x), ..., f_k(x))$$

Sujeto a:

$$g_i(x) \leq 0, i = \{1, \ldots, m\}$$
$$h_j(x) = 0, j = \{1, \ldots, p\}$$

con $x \in \Omega$.

Una solución MOP minimiza (o maximiza) las componentes de un vector $F(x)$, donde x es un vector de variables de decisión n-dimensional de algún universo Ω.

Dicho esto, para resolver el problema de optimización en cuestión se tienen que maximizar simultáneamente tres funciones objetivas en conflicto:

- Longitud del patrón.
- Soporte.
- Similitud.

$$f(x) = maximize\{f_1(x), f_2(x), f_3(x)\}$$

$$f_1(x) = l$$

$$f_2(x) = \sum_{i=1}^{D} threshold(\sum_{j=1}^{l} match(S_{ij}, Con_j), l)$$

$$f_3(x) = \frac{\sum_{j=1}^{l} max_b\{f(b,i)\}}{l}$$

donde:

- $f_1(x)$ es la longitud del patrón y maximiza el número de aminoácidos que componen el patrón.
- $f_2(x)$ es el soporte y maximiza el **número** de secuencias de proteínas usadas para construir la solución final. D es el número de secuencias de proteínas, j es la posición dentro del patrón, l es la longitud, S_{ij} es el caracter j del patrón candidato encontrado en la secuencia de proteínas i, Con_j es el caracter j del patrón de consenso, y $threshold$ y $match$ se definen como:

$$threshold(r, l) = \begin{cases} 0 & \text{si } r < (l * 0{,}50) \\ 1 & \text{si } r \geq (l * 0{,}50) \end{cases}$$

$$match(S_{ij}, Con_j) = \begin{cases} 0 & \text{si } S_{ij} \neq Con_j \\ 1 & \text{si } S_{ij} = Con_j \end{cases}$$

- $f_3(x)$ es la similitud y maximiza la **similitud** de las subsecuencias usadas para formar la solución final. $f(b,i)$ es el índice que indica la frecuencia de ocurrencia de un aminoácido b en la columna i y $max_b\{f(b,i)\}$ es el índice que indica la frecuencia de ocurrencia del aminoácido dominante en la columna i.

Además, la primera función objetivo, la longitud del patrón, está sujeta a la restricción C_1, que indica que su valor ($f_1(x)$, longitud del patrón) debe pertenecer al rango $(7 - 16)$. Este rango de longitudes es comúnmente usado al resolver este problema y, como se verá, también lo usan la mayoría de herramientas de biología.

Para entender por completo la formulación del problema, es necesario definir claramente las variables de decisión que se han de tomar en cuenta. Las variables de decisión se representan por la longitud de patrón y las posiciones de comienzo de cada subpatrón en cada secuencia (sl_x representa la posición de comienzo del subpatrón en la secuencia x).

Este conjunto de valores provee la información mínima que se necesita para extraer los subpatrones y calcular el valor de las tres funciones objetivo.

3.2. Implementación con METCO

Metaheuristic-based Extensible Tool for Cooperative Optimisation (METCO) es una herramienta desarrollada por el Grupo de Algoritmos y Lenguajes Paralelos del Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, que permite resolver problemas con distintos algoritmos multiobjetivo: el *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) [7] y el *Strength Pareto Evolutionary Algorithm 2* (SPEA2) [11] entre otros.

Esta herramienta permite la inclusión de plugins, es decir, permite agregar tanto nuevos algoritmos como nuevos problemas. En este caso, se agregará un nuevo problema a la herramienta: el de la búsqueda de patrones en secuencias.

Como la propia herramienta METCO está escrita en C++, los plugins también deben estar escritos en C++, por lo que se ha elegido este lenguaje para la implementación del problema.

En primer lugar, una instancia del problema está compuesta por un número concreto de secuencias de proteínas contenidas en un fichero en formato FAST-All (FASTA). Este formato se utiliza en ficheros que contienen información relacionada con la biología. Además de poder contener cadenas de ADN o secuencias de proteínas, puede contener información adicional como comentarios [2].

```
>sp|P22259|PCKA_ECOLI Phosphoenolpyruvate carboxykinase (ATP) OS=Escherichia coli (strain K12) OX=83333 GN=pckA PE=1 SV=2
MRVNNGLTPQELEAYGISDVHDIVYNPSYDLLYQEELDPSLTGYERGVLTNLGAVAVDTG
IFTGRSPKDKYIVRDDTTRDTFWWADKGKGKNDNKPLSPETWQHLKGLVTRQLSGKRLFV
VDAFCGANPDTRLSVRFITEVAWQAHFVKNMFIRPSDEELAGFKPDFIVMNGAKCTNPQW
KEQGLNSENFVAFNLTERMQLIGGTWYGGEMKKGMFSMMNYLLPLKGIASMHCSANVGEK
GDVAVFFGLSGTGKTTLSTDPKRRLIGDDEHGWDDDGVFNFEGGCYAKTIKLSKEAEPEI
YNAIRRDALLENVTVREDGTIDFDDGSKTENTRVSYPIYHIDNIVKPVSKAGHATKVIFL
TADAFGVLPPVSRLTADQTQYHFLSGFTAKLAGTERGITEPTPTFSACFGAAFLSLHPTQ
YAEVLVKRMQAAGAQAYLVNTGWNGTGKRISIKDTRAIIDAILNGSLDNAETFTLPMFNL
AIPTELPGVDTKILDPRNTYASPEQWQEKAETLAKLFIDNFDKYTDTPAGAALVAAGPKL
>sp|O09460|PCKA_ANASU Phosphoenolpyruvate carboxykinase (ATP) OS=Anaerobiospirillum succiniciproducens OX=13335 GN=pckA PE=1 SV=1
MSLSESLAKYGITGATNIVHNPSHEELFAAETQASLEGFEKGTVTEMGAVNVMTGVYTGR
SPKDKFIVKNEASKEIWWTSDEFKNDNKPVTEEAWAQLKALAGKELSNKPLYVVDLFCGA
NENTRLKIRFVMEVAWQAHFVTNMFIRPTEEELKGFEPDFVVLNASKAKVENFKELGLNS
ETAVVFNLAEKMQIILNTWYGGEMKKGMFSMMNFYLPLQGIAAMHCSANTDLEGKNTAIF
FGLSGTGKTTLSTDPKRLLIGDDEHGWDDDGVFNFEGGCYAKVINLSKENEPDIWGAIKR
NALLENVTVDANGKVDFADKSVTENTRVSYPIFHIKNIVKPVSKAPAAKRVIFLSADAFG
VLPPVSILSKEQTKYYFLSGFTAKLAGTERGITEPTPTFSSCFGAAFLTLPPTKYAEVLV
KRMEASGAKAYLVNTGWNGTGKRISIKDTRGIIDAILDGSIDTANTATIPYFNFTVPTEL
```

Figura 3.1: Fichero en formato FASTA

Las secuencias de proteínas están formadas por una longitud variable de caracteres que representan los aminoácidos que componen dicha proteína, tal y como se puede observar en la Figura 3.1. Las correspondencias entre el aminoácido y el caracter que lo representa se muestran en la Figura 3.2.

Amino Acid	3 letter code	1 letter code	Amino Acid	3 letter code	1 letter code
Glycine	Gly	G	Threonine	Thr	T
Alanine	Ala	A	Cysteine	Cys	C
Valine	Val	V	Tyrosine	Tyr	Y
Leucine	Leu	L	Asparagine	Asn	N
Isoleucine	Ile	I	Glutamine	Gln	Q
Methionine	Met	M	Aspartic Acid	Asp	D
Proline	Pro	P	Glutamic Acid	Glu	E
Phenyl alanine	Phe	F	Lysine	Lys	K
Tryptophan	Trp	W	Arginine	Arg	R
Serine	Ser	S	Histidine	His	H

Figura 3.2: Correspondencias de aminoácidos

El problema consiste en encontrar subsecuencias en cada una de las secuencias de proteínas que componen una instancia, que tengan un alto grado de similitud entre ellas. Además, también deberán tener una grado de similitud (al que se llamará soporte) con el patrón de consenso, una secuencia de 16 caracteres (la máxima longitud que pueden alcanzar las subsecuencias) que se utiliza como patrón de referencia para guiar la búsqueda hacia el patrón que se quiere encontrar.

Como se trata de algoritmos evolutivos, METCO requiere que cada problema esté compuesto por una única clase, que herede de la clase `Individual`, ya implementada en la herramienta, y que representa una solución del problema a tratar. Además, esta clase debe tener el mismo nombre que la carpeta que lo contiene, que se llamará como el problema en sí. Por lo tanto, se ha creado la clase `Pattern_Finding`, que hereda de la clase `Individual` de METCO.

Las variables de decisión de este problema son las mínimas necesarias para representar una solución del problema de la búsqueda de patrones. Esto es:

- Longitud del patrón.
- Posición de comienzo para cada secuencia.

La longitud del patrón es la longitud de la subsecuencia que se va a analizar. Su valor debe encontrarse, como ya se mencionó en la Sección 3.1, en el rango $(7 - 16)$. Las posiciones de comienzo son la posición de cada secuencia en la que comienza la subsecuencia a analizar. Si la instancia contiene n secuencias, habrán n posiciones de comienzo. Cabe destacar, que el valor mínimo para cada posición es 0, y el máximo la longitud de la secuencia menos la longitud del patrón (para evitar que la subsecuencia se salga fuera de los límites de la secuencia). De esta forma, en la Figura 3.3 se muestra la estructura de un individuo.

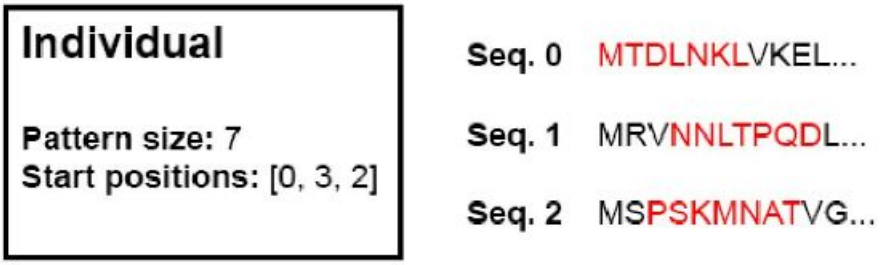


Figura 3.3: Estructura de un individuo

Por lo tanto, el número de variables de decisión para cada instancia es $1 + n$ (longitud del patrón + número de secuencias de la instancia), y sus valores se guardaran en un vector de tipo `double` que la clase `Individual` tiene como atributo.

Para las variables de decisión, es necesario para METCO implementar dos métodos que devuelvan los valores mínimos y máximos posibles, cuyos rangos ya se han mencionado. La variable en la posición 0 será el tamaño del patrón y el resto, las posiciones de comienzo. Véase el código en la Figura 3.4.

```
double Pattern_Finding::getMaximum(const int i) const {
        if (i == 0)
                return getMAX();
        else
                return getSequence(i - 1).size() - getPatternSize();
}

double Pattern_Finding::getMinimum(const int i) const {
        if (i == 0)
                return getMIN();
        else
                return 0;
}
```

Figura 3.4: Métodos `getMinimum` y `getMaximum`

Por otro lado, puesto que METCO no soporta atributos estáticos ni varias clases para un mismo problema, la clase `Pattern_Finding` tendrá los atributos que se pueden ver en la Figura 3.5.

```
class Pattern_Finding : public Individual {

private:

        vector < vector < char > > instance;//!<@brief instance of the problem
        string consensus; //!<@brief consensus pattern

        const int MIN = 7; //!<@brief minimum size for pattern
        const int MAX = 16; //!<@brief maximum size for pattern
```

Figura 3.5: Atributos de `Pattern_Finding`

- El atributo `instance` contiene la instancia del problema en un vector de vectores de tipo `char`. Debido al poco tamaño que ocupa cada instancia, no supone un gran sacrificio de memoria el que cada individuo de la población cuente con su propia copia de la instancia.
- El atributo `consensus` es un `string` que contiene el patrón de consenso utilizado para guiar la búsqueda de las subsecuencias.
- Los atributos `MIN` y `MAX` son constantes que representan el tamaño mínimo y máximo posible para las subsecuencias.

Así, se consigue un individuo de la población que representa una solución del problema con la mínima información necesaria.

METCO necesita que la clase implemente ciertos métodos. Uno de ellos es el método `init` al que se llama desde el constuctor. Véase el código en la Figura 3.6.

```
bool Pattern_Finding::init(const vector<string> &params) {
        if (params.size() != 2) {
                cerr << "Error in Pattern_Finding init: incorrect number of parameters" << end
                exit(-1);
        }
        initInstance(params[0]);
        setConsensus(params[1]);

        setNumberOfVar(1 + getNumberSequences());
        setNumberOfObj(3);

        return true;
}
```

Figura 3.6: Método `init`

En primer lugar, se comprueba que el número de parámetros sea el correcto. En este caso, deben ser dos: el número de la instancia y el patrón de consenso. A continuación, simplemente se llama a un método que lee la instancia desde el fichero correspondiente al número de instancia pasado como parámetro y lo inicializa, y luego se inicializa el patrón de consenso. Por último, se utilizan métodos de la clase `Individual` para establecer el número de variables de decision (1 + número de secuencias o posiciones de comienzo) y el número de funciones objetivo (que son tres: tamaño del patrón, soporte y similitud).

En lo referente a las funciones objetivo del problema, es necesario implementar el método `evaluate`, tal como se muestra en la Figura 3.7. Este método simplemente establece los valores de las tres funciones objetivo llamando a los métodos correspondientes que las calculan. En lo referente a dichos métodos, aparte del tamaño del patrón que se encuentra almacenado en el vector de variables de decisión, se implementarán las fórmulas descritas en la Sección 3.1.

```
void Pattern_Finding::evaluate(void) {
        setObj(0, getPatternSize());
        setObj(1, calculateSupport());
        setObj(2, calculateSimilarity());
}
```

Figura 3.7: Método `evaluate`

Para calcular el soporte, se usa el método definido en la Figura 3.8. En este método se recorren todas las subsecuencias, comparándolas con el patrón de consenso. En `matchCount` se almacena el número de aciertos (caracteres iguales en la misma posición) para la subsecuencia actual, y en `thresholdCount` se guarda el número de subsecuencias que tienen mas de un 50 % de aciertos. De esta manera, se devuelve el porcentaje de subsecuencias que tienen más de un 50 % de similitud con el patrón de consenso.

```
double Pattern_Finding::calculateSupport() const {
        int matchCount = 0;
        int thresholdCount = 0;

        for (int i = 0; i < getNumberSequences(); i++){
                vector<char> currentSeq(getSequence(i));

                int first = getStartPos(i);
                int last = first + getPatternSize();
                int k = 0;

                for (int j = first; j < last; j++ ) {
                        matchCount += isMatch(currentSeq.at(j), getConsensus().at(k));
                        k++;
                }
                thresholdCount += isThreshold (matchCount);
                matchCount = 0;
        }

        return (double)thresholdCount / getNumberSequences();
}
```

Figura 3.8: Método `calculateSupport`

Por otro lado, para calcular la similitud se utiliza el método definifido en la Figura 3.9. En este método se recorren las posiciones de cada subsecuencia (para una misma posición, se comprueba el caracter correspondiente en todas las subsecuencias), guardando el número de apariciones de cada caracter para la misma posición en todas las subsecuencias. Luego, se busca el caracter dominante (aquel con un número mayor de apariciones) y se calcula el porcentaje de veces que aparece respecto al número de subsecuencias. Se van sumando estos porcentajes y, por último, se calcula el porcentaje de apariciones de los caracteres dominantes para todas las subsecuencias. Así se puede averiguar el porcentaje de similitud entre todas las subsecuencias.

```
double Pattern_Finding::calculateSimilarity() const{
        double dominantValues = 0.0;

        for (int i = 0; i < getPatternSize(); i++) {
                map<char, int> ocurrences;
                for (int j = 0; j < getNumberSequences(); j++) {
                        char current = getSequence(j).at(getStartPos(j) + i);
                        if (ocurrences.count(current))
                                ocurrences[current]++;
                        else
                                ocurrences[current] = 1;

                }

                float maxValue = 0;
                map <char, int>::iterator it;
                for (it = ocurrences.begin(); it != ocurrences.end(); it++) {
                        if (it->second > maxValue)
                                maxValue = it->second;
                }

                dominantValues += maxValue / getNumberSequences();
                ocurrences.clear();
        }

        return  dominantValues / getPatternSize();
}
```

Figura 3.9: Método `calculateSimilarity`

Se requiere, además, la implementación de un operador de cruce y otro de mutación, dado que se está trabajando con algoritmos evolutivos. El operador de cruce utilizado es el *Simulated Binary Crossover* (SBX) y el operador de mutación utilizado es la mutación polinomial, ambos implementados en METCO [16] [14]. Tras cualquier mutación o cruce, el individuo generado se autoreparará (véase la Figura 3.10) en caso de que los resultados del cruce o de la mutación den lugar a un individuo erróneo (por ejemplo, si el subpatrón excede el límite de la secuencia en sí). Esto se hará recorriendo todas las posiciones de comienzo y calculando la diferencia entre ellas y el final de la secuencia. Si el menor de estos valores es menor que el tamaño del patrón, el nuevo tamaño del patrón será dicho mínimo.

```
void Pattern_Finding::repair() {
        int min = -1;
        for (int i = 0; i < getNumberSequences(); i++) {
                int dif = (int) getSequence(i).size() - getStartPos(i);
                if ((min == -1) || (min > dif))
                        min = dif;
        }

        if (min < getPatternSize())
                setPatternSize(min);
}
```

Figura 3.10: Método `repair`

Por último, se necesita un método que indique si hay que maximizar o minimizar cada función objetivo. En este caso, hay que maximizar las tres funciones objetivo por lo que el método que aparece en la Figura 3.11.

```
unsigned int Pattern_Finding::getOptDirection(const int i) const {
        return MAXIMIZE;
}
```

Figura 3.11: Métodos `getOptDirection`

Con esto concluye la descripción de la clase `Pattern_Finding`, excluyendo los métodos triviales. El código de esta implementación (comentado y explicado) se encuentra en el repositorio de Github [15] en la carpeta code/METCOPatternFinding junto con todos los archivos adicionales para este proyecto.

Capítulo 4

Resultados computacionales

En este capítulo se describe el diseño de los experimentos a realizar y los resultados obtenidos, para discernir con qué algoritmo se comporta mejor el problema.

4.1. Descripción de los experimentos

En primer lugar, se pretende realizar una comparativa entre dos algoritmos de optimización multiobjetivo disponibles en la herramienta METCO: *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) y *Strength Pareto Evolutionary Algorithm 2* (SPEA2).

Para ello, se emplearán tres instancias del problema de la búsqueda de patrones. Estas instancias han sido obtenidas de PROSITE [8], una base de datos online de secuencias de proteínas y otra información relacionada. En concreto, se pueden encontrar estas instancias en las referencias [4], [5] y [6] respectivamente. Las características de las instancias se describen en la Tabla 4.1. La primera columna muestra el nombre de la instancia. La segunda columna, el número de secuencias de proteínas que compone dicha instancia. La tercera, las longitudes mínima y máxima que llegan a alcanzar las secuencias de la instancia correspondiente. En la cuarta columna se tiene la longitud total de todas las secuencias de la instancia. Por último, en la última columna aparece el patrón de consenso seleccionado para guiar la búsqueda de patrones en la instancia.

Tabla 4.1: Instancias

Instancia	Secuencias	Lomgitud [MIN-MAX]	Tamaño	Patrón de consenso
Inst1	258	[466-671]	137,687	LIGDDEHGWSDNGILN
Inst2	502	[427-598]	246,406	NTDAEGRLNTDAEGRL
Inst3	745	[473-1238]	668,207	IVISTNMAGRGIDISL

Así, para cada algoritmo, se realizarán experimentos con cada instancia, de manera que se pueda comprobar la diferencia entre las instancias de tamaño menor como la primera, y las de tamaño mayor como la tercera. Otro de los parámetros que variará será el de la probabilidad de cruce. Así, mientras la probabilidad de mutación se mantendrá constante en 0.01 (con el fin de simplificar los experimentos y dado que no se detectan mejoras significativas al variarlo), la probabilidad de cruce podrá ser 0.8 o 1. De esta se comprobará la variación en la eficiencia cuando los individuos se cruzan sólo un 80 % de las veces o siempre.

Además, se tendrá en cuenta también el tamaño de la problación. La población de individuos podrá estar compuesta o por 50 individuos o por 100, de modo que se pueda comprobar qué cambios se producen cuando se trabaja con un menor número de individuos (y por lo tanto, una piscina

genética más reducida, donde hay menos diversidad) frente a cuando se trabaja con un mayor número de individuos (y por lo tanto, una mayor piscina y una mayor probabilidad de que haya diversidad entre los individuos).

Como último parámetro a variar, se modificará el número de evaluaciones totales (tras las cuales, el algortimo se detiene). Para este parámetro, se ha decidido variar entre 10000 y 100000 evaluaciones de las funciones objetivo. Así, podrá comprobarse si hay diferencias notables en la mejora de las soluciones si se deja que haya un mayor número de generaciones y por tanto, más probabilidades de que surjan cruces o mutaciones que mejoren la calidad de dichas soluciones.

Todos estos parámetros se combinarán hasta obtener todas las combinaciones posibles, y cada rama (o combinación posible) será ejecutada un total de 10 veces, dado que se está trabajando con algoritmos estocásticos.

Con los datos obtenidos se realizarán las siguientes métricas: para cada rama, se calculará la media de cada función objetivo del frente de pareto de cada ejecución. Después, se obtendrá el valor mínimo, máximo y la media de esas medias para las 10 ejecuciones de una misma rama. Además, para una rama y en las 10 ejecuciones pertinentes, se obtendrá el valor mínimo, máximo y medio de los tiempos de ejecución. De esta forma, se podrá ver la mejoría en cada uno de las funciones objetivo, comprobando y comparando la calidad de las soluciones encontradas por cada algoritmo y con cada parámetro. Finalmente, se analizarán estos datos para obtener conclusiones relevantes y evaluar posibles mejoras en el código o en los algoritmos.

4.2. Tablas de resultados

Los experimentos fueron realizados en un ordenador Debian GNU/Linux (versión 7.6) con cuatro procesadores AMD® Opteron™ (número de modelo 6348 HE) a 1.7 GHz y 64 GB RAM, con g++-4.7 como compilador de C++.

La Tabla 4.2 muestra el significado de las abreviaturas utilizadas en las tablas de resultados. Además, en las columnas de PS, SUP, SIM, TE y FS, se encuentran tres valores para cada una, que corresponden al valor mínimo, el máximo y la media, en ese orden.

Tabla 4.2: Abreviaturas

Abreviatura	Significado
I	Tamaño población
PC	Probabilidad de cruce
EV	Evaluaciones
PS	Tamaño del patrón
SUP	Soporte
SIM	Similitud
TE	Tiempo de ejecución (s)
FS	Tamaño del frente de Pareto

Las Tablas 4.3, 4.4 y 4.5 corresponden a los experimentos realizados con el algoritmo NSGA-II para las tres instancias, mientras que las Tablas 4.6, 4.7 y 4.8 corresponden a los experimentos realizados con el algoritmo SPEA2.

Tras la recogida de los datos, se procederá a la fase de análisis y conclusiones.

Tabla 4.3: Resultados NSGA-II - Instancia 1

I	PC	EV	PS	SUP	SIM	TE	FS
50	0.8	10K	10.500000	0.016957	0.126867	14.767591	15.000000
			12.466667	0.035659	0.142401	17.015566	26.000000
			11.319663	0.023764	0.136681	15.590882	17.100000
50	0.8	100K	9.352941	0.032376	0.190456	141.894558	16.000000
			12.937500	0.090224	0.221785	164.142890	30.000000
			10.945889	0.054038	0.205109	152.417937	21.200000
50	1	10K	9.450000	0.018519	0.126859	14.469494	9.000000
			13.454545	0.025840	0.136791	17.318191	31.000000
			11.435163	0.022660	0.130307	15.938492	19.800000
50	1	100K	9.566667	0.037611	0.161184	146.648993	12.000000
			12.916667	0.146964	0.219495	173.305174	32.000000
			11.394582	0.066104	0.177152	158.307374	23.100000
100	0.8	10K	9.944444	0.018032	0.127905	14.684264	11.000000
			12.533333	0.029951	0.137127	16.826942	23.000000
			11.557386	0.023831	0.132338	15.560430	18.600000
100	0.8	100K	9.941176	0.028424	0.172705	136.278747	16.000000
			12.333333	0.083333	0.208496	162.898145	42.000000
			11.335964	0.048679	0.189866	153.242809	28.500000
100	1	10K	10.500000	0.016078	0.122575	14.708734	11.000000
			14.090909	0.032515	0.130489	16.629297	31.000000
			11.845834	0.022175	0.127120	15.715539	22.500000
100	1	100K	9.250000	0.034703	0.160441	144.725800	14.000000
			11.666667	0.116279	0.191827	174.256646	43.000000
			10.671905	0.055443	0.170571	157.402935	29.200000

Tabla 4.4: Resultados NSGA-II - Instancia 2

I	PC	EV	PS	SUP	SIM	TE	FS
50	0.8	10K	9.074074	0.011952	0.151095	26.445029	10.000000
			13.647059	0.022381	0.163740	37.334751	35.000000
			11.192141	0.016923	0.157602	30.139574	24.000000
50	0.8	100K	9.387097	0.021212	0.193154	269.441752	23.000000
			12.024390	0.052226	0.211356	297.378907	41.000000
			10.329147	0.035444	0.203343	284.619430	33.700000
50	1	10K	10.156250	0.012761	0.143186	28.002773	23.000000
			13.733333	0.019788	0.161548	30.475975	45.000000
			11.615730	0.016148	0.151111	29.632614	33.500000
50	1	100K	9.393939	0.020938	0.168525	270.701975	23.000000
			13.043478	0.044489	0.185406	322.395741	46.000000
			11.065878	0.030067	0.176558	298.185166	37.700000
100	0.8	10K	10.526316	0.012824	0.149216	27.749984	15.000000
			13.708333	0.021046	0.159189	33.656495	32.000000
			11.803201	0.017119	0.152517	30.237738	23.900000
100	0.8	100K	9.833333	0.025564	0.185061	258.677794	32.000000
			12.529412	0.037028	0.200960	309.723551	67.000000
			10.895026	0.030989	0.191266	286.983364	45.500000
100	1	10K	10.775000	0.011759	0.143373	28.846711	19.000000
			13.781250	0.018489	0.152663	33.292731	46.000000
			12.006046	0.014199	0.147550	30.867476	29.700000
100	1	100K	10.829268	0.019307	0.162698	275.979433	34.000000
			11.955224	0.035681	0.180471	332.988619	82.000000
			11.388013	0.026028	0.169496	301.268765	59.200000

Tabla 4.5: Resultados NSGA-II - Instancia 3

I	PC	EV	PS	SUP	SIM	TE	FS
50	0.8	10K	10.806452	0.008483	0.113907	52.410877	23.000000
			12.304348	0.011573	0.120653	56.567960	40.000000
			11.401519	0.010060	0.117448	54.788275	30.000000
50	0.8	100K	9.307692	0.015992	0.143599	497.778014	33.000000
			11.157895	0.027121	0.155959	563.045141	45.000000
			10.239514	0.019290	0.151168	529.621236	36.900000
50	1	10K	9.951220	0.007027	0.109205	51.160591	26.000000
			11.846154	0.012597	0.118821	57.587749	41.000000
			11.191778	0.009623	0.112615	55.007830	34.200000
50	1	100K	10.159091	0.014737	0.119128	534.218950	27.000000
			12.104167	0.022181	0.131131	604.835012	48.000000
			11.393708	0.018580	0.124858	568.595103	43.600000
100	0.8	10K	10.846154	0.008054	0.111546	53.272217	13.000000
			12.200000	0.011255	0.121695	59.409664	36.000000
			11.332510	0.009468	0.114139	56.639537	28.400000
100	0.8	100K	9.612903	0.015056	0.139270	501.859902	31.000000
			11.105263	0.022925	0.148983	573.594238	60.000000
			10.295472	0.019372	0.144153	536.281739	43.700000
100	1	10K	10.531250	0.007785	0.109766	52.756473	24.000000
			12.472222	0.009918	0.117945	59.563503	41.000000
			11.418065	0.008674	0.113773	55.689280	32.600000
100	1	100K	10.361111	0.013226	0.119935	536.341439	29.000000
			12.439024	0.024545	0.126881	619.874573	60.000000
			11.397927	0.018198	0.123299	570.790702	48.100000

Tabla 4.6: Resultados SPEA2 - Instancia 1

I	PC	EV	PS	SUP	SIM	TE	FS
50	0.8	10K	10.100000	0.016796	0.130179	13.846668	10.000000
			14.400000	0.032171	0.149903	15.923890	28.000000
			12.177136	0.023598	0.136545	14.830859	18.600000
50	0.8	100K	10.266667	0.033709	0.179636	131.694934	15.000000
			11.894737	0.071835	0.217736	146.618122	37.000000
			11.161064	0.044283	0.198848	140.405792	26.700000
50	1	10K	10.142857	0.021420	0.125998	13.257917	10.000000
			14.100000	0.029393	0.134321	16.700211	25.000000
			11.468992	0.025039	0.130036	14.540666	20.400000
50	1	100K	10.050000	0.032730	0.156039	132.853463	16.000000
			13.952381	0.135901	0.214444	159.396627	31.000000
			12.205890	0.062018	0.178801	148.013684	23.600000
100	0.8	10K	10.583333	0.016611	0.126439	13.723223	12.000000
			11.875000	0.025194	0.135501	15.333234	32.000000
			11.411397	0.020522	0.131268	14.598960	20.400000
100	0.8	100K	9.435897	0.030760	0.172462	133.512159	26.000000
			12.038462	0.051190	0.199779	151.219994	47.000000
			10.887113	0.037443	0.187432	144.642228	33.400000
100	1	10K	9.733333	0.016346	0.123386	13.898671	12.000000
			13.750000	0.027293	0.132782	16.767575	28.000000
			11.593084	0.021568	0.127977	15.033794	21.800000
100	1	100K	10.666667	0.028365	0.150551	139.216993	26.000000
			13.400000	0.048897	0.170775	170.557511	44.000000
			11.786599	0.037972	0.158928	153.433299	36.700000

Tabla 4.7: Resultados SPEA2 - Instancia 2

I	PC	EV	PS	SUP	SIM	TE	FS
50	0.8	10K	10.888889	0.013318	0.146441	25.273350	12.000000
			13.000000	0.029050	0.157895	28.286682	36.000000
			11.971089	0.018296	0.153912	26.752051	27.400000
50	0.8	100K	10.172414	0.026602	0.191241	254.871847	27.000000
			12.894737	0.042128	0.204332	276.422706	48.000000
			11.412743	0.034919	0.199699	265.413170	34.400000
50	1	10K	10.958333	0.013710	0.144029	25.713449	14.000000
			13.818182	0.022908	0.153721	28.139456	48.000000
			12.297177	0.017393	0.148399	27.258448	27.800000
50	1	100K	10.794872	0.024799	0.167626	261.466562	18.000000
			14.444444	0.042900	0.182993	315.512408	50.000000
			11.896012	0.033559	0.177204	276.568019	37.300000
100	0.8	10K	10.192308	0.010343	0.145817	26.505418	23.000000
			13.925926	0.020068	0.155933	28.938469	33.000000
			12.049148	0.015503	0.149346	27.580129	27.900000
100	0.8	100K	10.111111	0.024868	0.182042	247.581981	31.000000
			13.236842	0.037913	0.195301	302.108799	71.000000
			11.072620	0.029899	0.189565	267.559955	46.900000
100	1	10K	10.142857	0.013375	0.141424	26.021543	18.000000
			12.750000	0.017928	0.153994	28.632247	40.000000
			11.656953	0.015746	0.145849	27.603796	25.700000
100	1	100K	10.125000	0.021863	0.167174	249.449093	29.000000
			12.045455	0.031125	0.179707	282.123525	66.000000
			11.159075	0.027658	0.172466	270.513834	47.400000

Tabla 4.8: Resultados SPEA2 - Instancia 3

I	PC	EV	PS	SUP	SIM	TE	FS
50	0.8	10K	10.406250	0.007659	0.112373	46.659051	18.000000
			12.529412	0.013208	0.119908	50.654424	34.000000
			11.470440	0.009447	0.116250	48.141924	26.600000
50	0.8	100K	9.540541	0.014143	0.143472	464.969417	37.000000
			12.404762	0.019953	0.154438	506.370877	50.000000
			11.208234	0.017352	0.148200	486.169669	44.700000
50	1	10K	9.857143	0.007670	0.111650	46.328431	23.000000
			12.085714	0.010115	0.115991	49.078253	42.000000
			10.899741	0.009120	0.113879	47.674262	33.500000
50	1	100K	10.200000	0.012155	0.124410	449.227437	30.000000
			12.333333	0.028104	0.132532	503.360868	50.000000
			11.162193	0.018981	0.127608	488.777931	39.800000
100	0.8	10K	10.764706	0.007293	0.110822	46.980140	21.000000
			12.205882	0.010968	0.117098	52.052487	38.000000
			11.379015	0.008522	0.114272	49.245861	31.400000
100	0.8	100K	9.875000	0.012796	0.136917	469.188499	38.000000
			11.868421	0.022360	0.144869	505.053513	63.000000
			10.955984	0.016543	0.140395	491.337755	47.800000
100	1	10K	10.156250	0.007287	0.110009	48.963424	28.000000
			12.178571	0.011352	0.113937	51.375706	49.000000
			11.147502	0.008749	0.111622	50.126329	37.000000
100	1	100K	9.413043	0.011425	0.118739	476.796353	43.000000
			12.603448	0.019118	0.129311	528.938735	73.000000
			11.541770	0.014547	0.124124	505.409316	55.900000

Capítulo 5

Conclusiones y líneas futuras

En este capítulo se detallará el análisis realizado y las conclusiones sacadas, así como las líneas futuras de este proyecto.

5.1. Análisis y resultados

Tras un análisis detallado de los datos obtenidos al realizar los experimentos, se ha llegado a las siguientes conclusiones.

Para la instancia 1, que es la que menor número de secuencias tiene (258), el algoritmo NSGA-II da mejores resultados en las funciones objetivo que el algoritmo SPEA2, exceptuando el tamaño del patrón cuando la población es pequeña (50 individuos), donde el algoritmo SPEA2 da mejores resultados. Por otro lado, el algoritmo SPEA2 tiene tiempos de ejecución menores en cualquiera de las combinaciones de variables.

Para la instancia 2, que tiene 502 secuencias, ocurre algo bastante similar: el algoritmo NSGA-II obtiene, por norma general, mejores resultados en las funciones objetivo, exceptuando el tamaño del patrón en la mayoría de casos (no solo cuando la población es pequeña). Además, el algoritmo SPEA2 obtiene mejores resultados en el soporte que el algoritmo NSGA-II en una ligera mayoría de casos. Cabe destacar que el algoritmo SPEA2 sigue obteniendo mejores tiempos de ejecución.

Para la instancia 3, que consta con 745 secuencias (la mayor de las tres), se obtienen resultados similares. Sin embargo, no se perciben mejoras en los resultados del algoritmo SPEA2 con respecto al algoritmo NSGA-II, a excepción de en la función objetivo del tamaño del patrón. Aún así, SPEA2 sigue teniendo mejores tiempos de ejecución.

En definitiva, en términos de maximizar las funciones objetivo, los mejores resultados los obtiene el algoritmo NSGA-II. Sin embargo, los resultados del algoritmo SPEA2 van mejorando con instancias más grandes, por lo que, aunque los datos no son concluyentes, podría determinarse que el algoritmo SPEA2 podría llegar a superar al algoritmo NSGA-II cuando el tamaño de las instancias aumente lo suficiente. Por otro lado, el algoritmo SPEA2 consigue mejores tiempos de ejecución en todos los casos.

En cuanto al tamaño de población, el algoritmo NSGA-II consigue mejores resultados para soporte, similitud y tiempo de ejecución cuando la población es menor, y mejores resultados para el tamaño del patrón cuando la población es mayor. Por otro lado, el algoritmo SPEA2 consigue mejores resultados para todos los ámbitos cuando la población es mayor.

En lo referente a la probabilidad de cruce, se obtienen mejores resultados en el tamaño del patrón cuando la probabilidad de cruce es alta, y mejores resultados en soporte y similitud cuando la probabilidad de cruce es baja (en ambos algoritmos). Además, los tiempos de ejecución son menores cuando la probabilidad de cruce es baja (dado que debe realizar menos operaciones de cruce).

Por último, se observan mejores resultados en las funciones objetivo cuando el número de evaluaciones es mayor (exceptuando el tamaño del patrón que en ocasiones es mejor cuando el número de evaluaciones es menor). Sin embargo, el tiempo de ejecución aumenta exponencialmente con el número de evaluaciones.

En conclusión, ambos algoritmos presentan ciertas ventajas dados ciertos parámetros, por lo que la elección tanto del algoritmo a utilizar como de los parámetros debe hacerse en base a los objetivos planteados.

5.2. Líneas futuras

Basándose en las conclusiones obtenidas, se pueden realizar un gran número de modificaciones en el futuro.

Para empezar, un objetivo claro es el de aumentar el número de evaluaciones disminuyendo el tiempo de ejecución. Esto puede conseguirse paralelizando el problema, lo cual debería dar a lugar a mejores resultados.

Por otro lado, otra modificación posible es la de restringir el valor mínimo permitido de la similitud y el soporte, de manera que aquellas soluciones que no lo superen, sean descartadas. Para evitar que en un principio se descarten todas las soluciones, estas restricciones pueden tener un valor pequeño en las primeras generaciones, e ir aumentando conforme pasen las generaciones, volviéndose más estrictas y obligando a que las soluciones sean cada vez mejores. Esto puede aumentar significativamente el tiempo de ejecución, por lo que también se necesitaría paralelizar el problema.

Otras posibles modificaciones son las de utilizar otros operadores de cruce y mutación y comparar los resultados, o utilizar otros algoritmos para la resolución del problema.

Además, se pueden variar el resto de parámetros, como las probabilidades de cruce y mutación, el tamaño de la población o, incluso, los patrones de consenso (que puede ser el motivo de que los valores de la función objetivo del soporte sean tan bajos).

En definitiva, se pueden realizar una gran cantidad de modificaciones, y este proyecto queda abierto a un gran número de mejoras futuras.

Capítulo 6

Summary and Conclusions

Details about the analysis and the conclusions, as well as the future lines for this project can be found in this chapter.

6.1. Analysis and results

After a detailed analysis of the results obtained from the experiments, the following conclusions have been reached.

For the first instance, which is the one with the least amount of sequences (258), the NSGA-II algorithm gives better results in the objective functions than the SPEA2 algorithm, except for the pattern size when the population is small (50 individuals), where the SPEA2 algorithm gets better results. At the same time, the SPEA2 algorithm has better execution times in all possible combinations.

For the second instance, which has 502 sequences, something similar happens: the NSGA-II algorithm obtains, as a general rule, better results in the objective functions, except for the pattern size in most cases (not only when the population is small). On top of that, the SPEA2 algorithm obtains better results in the support than the NSGA-II algorithm in a slight majority of cases. It should be noted that the SPEA2 algorithm keeps obtaining better execution times.

For the third instance, which has 745 sequences (the biggest one), similar results are obtained. However, there are no better results perceived for the SPEA2 algorithm other than the pattern size, but it still has better execution times.

To conclude, in terms of maximizing the objective functions, the best results are obtained by the NSGA-II algorithm. However, the SPEA2 algorithm's results keep getting better with bigger instances, so, even although the data is not conclusive, it could be determined that the SPEA2 algorithm would overcome the NSGA-II algorithm when the instances are big enough. At the same time, the SPEA2 algorithm gets better execution times in all cases.

In terms of population size, the NSGA-II algorithm gets better results for support, similarity and execution time when the population is smaller, and better results for pattern size when the population is bigger. Apart from that, the SPEA2 algorithm gets better results for all cases when the population is bigger.

With regards to the crossover probability, better results are obtained for the pattern size when the crossover probability is higher, and better results for support and similarity when the crossover probability is lower (in both algorithms). Besides that, the execution times are better when the crossover

probability is lower (since it has to do fewer crossover operations).

Finally, better results are observed in the objective functions when the number of evaluations is higher (except for the pattern size, which sometimes is better when the number of evaluations is lower). However, the execution time increases exponentially with the number of evaluations.

Ultimately, both algorithms show some advantages given certain parameters, which is why choosing the algorithm and the parameters should be done according to the objective.

6.2. Future lines

Based on the obtained conclusions, a large number of modifications can be made in the future.

To begin with, an obvious objective is that of increasing the number of evaluations while decreasing the execution time. This can be done by parallelizing the problem, which would lead to better results.

Besides that, another possible modification is restricting the minimum value allowed for support and similarity, so that those solutions that do not surpass it, are discarded. To avoid discarding every solution at the start, the constraints could start with a small number in the first generations, and keep increasing in later generations, becoming more strict and forcing the solutions to be better every time. This may also increase the execution time significantly, which is why it would be important to parallelize the problem as well.

Other possible modification is using other crossover and mutation operators and compare the results, or using other algorithms to solve the problem.

On top of that, the rest of parameters can also be modified, like the crossover and mutation probabilities, the population size, or even the consensus patterns (since they could be the reason why the support's values are so low).

To conclude, the possible modifications are numerous, and this project remains open to a large number of future improvements.

Capítulo 7

Presupuesto

En este capítulo se detalla el presupuesto estimado de este proyecto.

7.1. Coste de la investigación

En la Tabla 7.1 se muestra el presupuesto que se estima para llevar a cabo la investigación.

Tabla 7.1: Presupuesto de la investigación

Recurso	**Horas**	**Precio Unidad(€)**	**Total(€)**
Estudio sobre la Nutrigenómica y la Nutrigenética	50	11	550
Estudio del problema de la búsqueda de patrones	30	11	330
Implementación del individuo	60	11	660
Estudio de la herramienta METCO	10	11	110
Estudio de los algoritmos utilizados	10	11	110
Diseño y realización de los experimentos	50	11	550
Análisis de resultados y conclusiones	5	11	110
Total	215	-	2420

7.2. Coste del equipo

En cuanto al equipo necesario para realizar los experimentos, se ha presupuestado teniendo en cuenta el precio del núcleo por hora. Dado que el equipo tenía 4 núcleos, el presupuesto es el mostrado en la Tabla 7.2.

Tabla 7.2: Presupuesto del equipo

Recurso	**Horas**	**Precio núcleo/hora(€)**	**Número de núcleos**	**Total(€)**
Equipo	50	2	4	400
Total	-	-	-	400

7.3. Presupuesto total

En la Tabla 7.3 se muestra el importe final del presupuesto.

Tabla 7.3: Presupuesto total

Concepto	**Importe(€)**
Trabajo autónomo	2420
Equipo informático	400
Total	2820

Bibliografía

[1] ACM Digital Library (accedido el 28 de junio de 2018). https://dl.acm.org/.

[2] .FASTA file extension (accedido el 28 de junio de 2018). https://www.reviversoft.com/file-extensions/fasta?ncr=1&lang=en.

[3] IEEE Xplore (accedido el 28 de junio de 2018). https://ieeexplore.ieee.org/Xplore/home.jsp.

[4] Instancia 1 (accedido el 28 de junio de 2018). https://prosite.expasy.org/PS00532.

[5] Instancia 2 (accedido el 28 de junio de 2018). https://prosite.expasy.org/PS00631.

[6] Instancia 3 (accedido el 28 de junio de 2018). https://prosite.expasy.org/PS01312.

[7] NSGAII (accedido el 28 de junio de 2018). https://esa.github.io/pagmo2/docs/cpp/algorithms/nsga2.html.

[8] PROSITE (accedido el 28 de junio de 2018). https://prosite.expasy.org/.

[9] .Q (accedido el 28 de junio de 2018). http://puntoq.greendata.es.accedys2.bbtk.ull.es/primo_library/libweb/action/search.do?vid=ull&reset_config=true&afterPDS=true.

[10] Scopus (accedido el 28 de junio de 2018). https://www.scopus.com/.

[11] SPEA2 (accedido el 28 de junio de 2018). http://www.cleveralgorithms.com/nature-inspired/evolution/spea.html.

[12] WebOfScience (accedido el 28 de junio de 2018). https://www.fecyt.es/es/recurso/web-science.

[13] David L González-Álvarez, Miguel A Vega-Rodríguez, and Álvaro Rubio-Largo. A hybrid mpi/openmp parallel implementation of nsga-ii for finding patterns in protein sequences. *The Journal of Supercomputing*, 73(6):2285–2312, 2017.

[14] Mohammad Hamdan. On the disruption-level of polynomial mutation for evolutionary multi-objective optimisation algorithms. *Computing and Informatics*, 29(5):783–800, 2012.

[15] Pamela Jiménez. Repositorio de este proyecto (accedido el 28 de junio de 2018). https://github.com/coromoto/TFG-BIO/tree/master/code.

[16] Mayank Goyal Kalyanmoy Deb. A combined genetic adapdative search (geneas) for engineering design (accedido el 28 de junio de 2018). https://es.slideshare.net/paskorn/simulated-binary-crossover-presentation.

[17] Sridhar A Malkaram, Yousef I Hassan, and Janos Zempleni. Online tools for bioinformatics analyses in nutrition sciences. *Advances in Nutrition*, 3(5):654–665, 2012.

[18] Michael Müller and Sander Kersten. Nutrigenomics: goals and strategies. *Nature Reviews Genetics*, 4(4):315, 2003.

[19] Damien Valour, Isabelle Hue, Bénédicte Grimard, and Bernard Valour. Gene selection heuristic algorithm for nutrigenomics studies. *Physiological genomics*, 45(14):615–628, 2013.