CECAL
Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

PROYECTO DE GRADO INGENIERÍA EN COMPUTACIÓN

Resolución del problema de clustering utilizando algoritmos evolutivos

Lucía Carozzi, María Eugenia Curi
lucia.carozzi@gmail.com, meugenia.curi@gmail.com

Noviembre de 2016

Tutor del proyecto:

Sergio Nesmachnow, Universidad de la República.
Franco Robledo, Universidad de la República.

Resolución del problema de clustering utilizando algoritmos evolutivos

Carozzi, Lucía
Curi, María Eugenia

Proyecto de Grado

CECAL
Instituto de Computación - Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay, Noviembre de 2016

Resolución del problema de clustering utilizando algoritmos evolutivos

Resumen

Este proyecto estudia el problema de clustering, donde se busca encontrar una agrupación óptima de elementos. Se analizan dos variantes del problema: una variante monoobjetivo donde se busca maximizar la sumatoria de las similitudes de los elementos al centro del grupo al que pertenecen y una variante multiobjetivo del problema, donde además de maximizar la sumatoria de las similitudes, se plantea minimizar simultáneamente la cantidad de grupos.

Se implementan doce algoritmos evolutivos diferentes para la variante monoobjetivo del problema y un algoritmo evolutivo para la variante multiobjetivo del problema. Los algoritmos evolutivos implementados son evaluados utilizando un conjunto de instancias, dos de ellas generadas por colegas de la Universidad de Luxemburgo y once obtenidas de un repositorio público con instancias de pruebas específicas para el problema de clustering. Los resultados experimentales muestran que para la variante monoobjetivo del problema, los algoritmos evolutivos implementados alcanzan mejoras de hasta un 156.2 % respecto a una estrategia ávida que resuelve el mismo problema. En relación al algoritmo de referencia de la literatura relacionada, se alcanzan mejoras de hasta un 9.5 %. Para la variante multiobjetivo del problema, los resultados experimentales muestran que el algoritmo evolutivo implementado logra mejorar hasta un 31.4 % el mejor resultado del algoritmo de referencia para igual cantidad de grupos. El algoritmo evolutivo que resuelve la variante monoobjetivo obtiene resultados un 2 % en promedio por encima de los resultados obtenidos por el algoritmo evolutivo en su variante multiobjetivo, considerando que el primero es un algoritmo específico para encontrar una agrupación optima dada una cantidad de grupos.

Palabras clave: problema de clustering, optimización, cluster, algoritmos evolutivos

Clustering problem resolution using evolutionary algorithms

Abstract

This project studies the clustering problem, where the goal is to organize information by grouping individuals. Two different variants of the problem are studied: a single objective variant with the goal of maximizing the sum of the similarities between the elements and the center of the group to which they belong to and a multiobjective variant, which proposes the simultaneous minimization of the number of groups.

Twelve different evolutionary algorithms are implemented for the single objective variant of the problem and one evolutionary algorithm is implemented for the multiobjective problem variant. The evolutionary algorithms implemented are evaluated using a set of instances, two of them generated by colleagues from the University of Luxembourg and eleven obtained from a public repository with specific instances for the clustering problem. The experimental results show that the evolutionary algorithms developed for the single objective variant of the problem are able to improve up to 156.2 % upon the results reached using a greedy algorithm. While regarding a reference algorithm from the literature, improvements of up to 9.5 % are achieved. Regarding the multiobjective variant of the problem, the experimental results show that the evolutionary algorithms are able to obtain solutions that improve up to 31.42 % over the best result of the reference algorithm for the same amount of groups. The evolutionary algorithm that solves the single objective variant of the problem obtained results 2 % on average above the results obtained by the evolutionary algorithm in its multi objective variant, where the first one is a specific algorithm aimed at finding an optimal grouping when a number of groups are given.

Keywords: clustering problem, optimization, cluster, evolutionary algorithms

Índice general

Índice de figuras vii

Índice de tablas ix

1. Introducción 1

2. Definición del problema 3
2.1. Versión monoobjetivo del problema de clustering . . . 3
2.1.1. Introducción . . . 3
2.1.2. Formulación matemática . . . 5
2.1.3. Caso de ejemplo . . . 6
2.2. Versión multiobjetivo del problema de clustering . . . 7
2.2.1. Introducción . . . 7
2.2.2. Formulación matemática . . . 7
2.2.3. Caso de ejemplo variante multiobjetivo . . . 7
2.3. Complejidad del problema . . . 8

3. Algoritmos evolutivos 11
3.1. Algoritmos evolutivos . . . 11
3.1.1. Introducción . . . 11
3.1.2. Representación . . . 12
3.1.3. Función de Fitness . . . 13
3.1.4. Operadores evolutivos . . . 14
3.2. Algoritmos evolutivos para problemas mulitobjetivo . . . 14
3.2.1. Problemas de optimización multiobjetivo . . . 14
3.2.2. Algoritmo evolutivo para optimización multiobjetivo: NSGA-II . . 16

4. Trabajos relacionados 19
4.1. Algoritmos basados en trayectoria para el problema del clustering . . . 19
4.2. Algoritmos evolutivos para el problema de clustering . . . 22
4.3. Resumen . . . 25
4.4. Algoritmos de la literatura para la resolución del problema de clustering . 26
4.4.1. KMedoid . . . 26
4.4.2. Local Search . . . 27
4.4.3. Algoritmo Ávido . . . 27
4.4.4. Algoritmo linkage . . . 27
4.4.5. Híbrido . . . 28

5. Implementación **31**
5.1. Bibliotecas de desarrollo ECJ 31
5.2. Algoritmo evolutivo versión monoobjetivo 31
5.2.1. Procedimiento para el cálculo de fitness 32
5.2.2. Operadores para representación binaria 33
5.2.3. Operadores para representación entera 37
5.3. Algoritmo evolutivo versión multiobjetivo 41

6. Evaluación experimental **43**
6.1. Obtención de instancias de prueba 43
6.2. Metodología 44
6.3. Variante monoobjetivo del problema de clustering 44
6.3.1. Parámetros 45
6.3.2. Resultados numéricos 46
6.4. Variante multiobjetivo del problema de clustering 55
6.4.1. Operadores evolutivos y parámetros 55
6.4.2. Hipervolumen relativo como métrica para optimización multiobjetivo 55
6.4.3. Resultados numéricos 56

7. Conclusiones y trabajo a futuro **61**
7.1. Conclusiones 61
7.2. Trabajo futuro 62

Bibliography **62**

8. Bibliografía **63**

Índice de figuras

3.1. Ejemplo representación binaria para el problema del Clustering 13
3.2. Ejemplo representación entera 13
3.3. Soluciones dominadas, no dominadas y frente de Pareto en un MOP con dos objetivos 15
3.4. Fitness "degradado"para la función de *sharing* 16
3.5. Rangos en frentes de pareto NSGA II 17

4.1. Ejemplo de aplicación del algoritmo KMedoid 26

5.1. Procedimiento para seleccionar centros y calcular el fitness de una solución 33
5.2. Cruzamiento de un punto en representación binaria 34
5.3. Cruzamiento de dos puntos en representación binaria 34
5.4. Procedimiento para la mutación de un gen 35
5.5. Función correctiva de soluciones no factibles en representación binaria . . 36
5.6. Procedimiento para el cruzamiento de un punto con representación entera 37
5.7. Procedimiento para el cruzamiento GenC&S 39
5.8. Procedimiento para evitar individuos no factibles en el AE que utiliza representación entera 41

6.1. Porcentajes de mejora del AE monoobjetivo sobre los otros algoritmos . . 54
6.2. AE Monoobjetivo vs. AE Multiobjetivo vs. KMedoid - comparación mejores resultados para instancias chicas 57
6.3. AE Monoobjetivo vs. AE Multiobjetivo vs. KMedoid - comparación mejores resultados para instancias medianas 58
6.4. AE Monoobjetivo vs. AE Multiobjetivo vs. KMedoid - comparación mejores resultados para instancias grandes 59

Índice de tablas

2.1. Cálculo de ST para las mejores agrupaciones con dos centros posibles. . . 6
2.2. Soluciones óptimas para diferentes valores de cantidad de grupos. 8

4.1. Trabajos relacionados al problema de clustering. 25

6.1. Tabla comparativa de los AE con representación entera para las instancias chicas 47
6.2. Tabla comparativa de los AE con representación entera para las instancias medianas 47
6.3. Tabla comparativa de los AE con representación entera para las instancias grandes 48
6.4. Tabla comparativa de los AE con representación binaria para las instancias chicas 49
6.5. Tabla comparativa de los AE con representación binaria para las instancias medianas 50
6.6. Tabla comparativa de los AE con representación binaria para las instancias grandes 50
6.7. Tabla comparativa de los AE: *One-Oneoc* vs. *GenC&S-Oneoc* vs. *One-Flip* vs. *One-Delete* vs. *Two-Delete*. 51
6.8. Tabla comparativa AE monoobjtivo vs. algoritmos de la literatura para las instancias chicas 52
6.9. Tabla comparativa AE monoobjtivo vs. algoritmos de la literatura para las instancias medianas 53
6.10. Tabla comparativa AE monoobjtivo vs. algoritmos de la literatura para las instancias grandes 53
6.11. Promedio del hipervolumen relativo para la variante multiobjetivo del AE 56
6.12. Porcentaje de mejora (%) del AE en su variante monoobjetivo sobre la variante multiobjetivo y KMedoid 60

Lista de algoritmos

1. NSGA-II 17
2. Greedy 27
3. Fitness Lucasius 32
4. Cruzamiento de un punto 33
5. Cruzamiento de dos puntos 34
6. Bit Flip Mutation 35
7. Add Mutation 36
8. Delete Mutation 36
9. Función correctiva de soluciones no factibles 37
10. Cruzamiento de un punto con representación entera 38
11. Cruzamiento GenC&S para la representación entera 38
12. Cruzamiento híbrido para la representación entera 39
13. Mutación de un gen para la representación entera 40
14. Mutación de un gen adaptada 40
15. Función correctiva para el algoritmo evolutivo multiobjetivo 42
16. Cálculo de las funciones objetivo para el algoritmo evolutivo multiobjetivo 42

Capítulo 1

Introducción

El campo de la investigación biomédica experimenta un desarrollo dramático y rápido, generando constantemente nueva información ampliando así su volumen y detalle. Dicha información es almacenada en conjuntos de datos y bases de conocimiento de tipos muy diversos, con interfaces a menudo poco intuitivas. Esto se traduce a que el conocimiento bioinformático, en general, sea representado mediante grafos complejos de gran tamaño. Para poder realizar una exploración e interpretación significativa de estos grafos es necesario poder visualizarlos y manejarlos correctamente. Esta inquietud es la que expresan colegas de la Universidad de Luxemburgo y proponen a las metaheurísticas como métodos para resolver el problema de organizar redes biológicas de información. Esta investigación forma parte de un a proyecto más amplio que pretende comprender y organizar la información relevante a la enfermedad de Parkinson [1].

Bajo este paradigma, es necesario encontrar una agrupación óptima que permita la búsqueda eficiente de elementos dentro de un grafo. El problema de encontrar una agrupación óptima (problema de clustering) es ampliamente conocido, siendo posible encontrar en la literatura distintos algoritmos que resuelven este problema. Por esta razón, el problema de clustering es utilizado para organizar un conjunto de datos de gran tamaño, con el objetivo de comprender la información que contiene de manera más eficiente. Al agrupar el conocimiento, es posible describir cada grupo utilizando una etiqueta y de esta manera acceder a la información eficientemente. Uno de los usos del problema de clustering es el estudio de la segmentación del mercado, donde se agrupa el público objetivo según sus preferencias de consumo. El problema de agrupar un conjunto de datos es interesante para la búsqueda de material relevante en la web [13]. Además de las actividades descritas anteriormente, el problema de clustering es utilizado para la categorización de documentos, procesamiento de imágenes, entre otros [33].

Si bien el problema de clustering consiste en encontrar una mejor agrupación de un conjunto de datos, es posible considerar adicionalmente la cantidad de grupos que se deben formar previamente o explorar el espacio de cantidad de grupos en busca de las mejores agrupaciones. En el primer caso, el problema tiene un solo objetivo que es hallar la mejor agrupación dada una cantidad de grupos. En el segundo caso, el problema tiene dos objetivos que se quieren cumplir, obtener la mejor agrupación de elementos con la menor cantidad de grupos.

El proyecto que se describe en este informe tiene como objetivo estudiar el problema de clustering, tanto en su variante monoobjetivo como multiobjetivo, así como diseñar e

[1] `http://wwwen.uni.lu/lcsb/research/parkinson_s_disease_map`

implementar algoritmos que lo resuelvan eficientemente. Las principales contribuciones del trabajo son las siguientes:

1. El análisis de la literatura relacionada al problema de clustering, específicamente al problema de clustering en su variante particional, exclusivo y completo.
2. La definición y formulación matemática de dos variantes del problema de clustering:
   a) una variante monoobjetivo del problema donde, dada una cantidad fija de grupos, se busca maximizar la sumatoria de las similitudes de cada elemento al centro de su grupo.
   b) una variante multiobjetivo del problema, donde además de maximizar la sumatoria de las similitudes, se plantea minimizar simultáneamente la cantidad de grupos de la solución.
3. La implementación de doce algoritmos evolutivos que resuelven el problema de clustering en su variante monoobjetivo, y la implementación de un algoritmo evolutivo para la variante multiobjetivo del problema.
4. La obtención de un conjunto de instancias realistas de prueba para el problema de clustering utilizando información real de repositorios públicos: un conjunto de doce instancias para la variante monoobjetivo y una instancia adicional a las anteriores para la variante multiobjetivo del problema.
5. La evaluación experimental de los algoritmos propuestos, comparándolos entre sí y contra otros algoritmos diseñados en base a las ideas presentadas en trabajos de la literatura relacionada, siguiendo una estrategia intuitiva para resolver el problema de clustering.

El resto del documento se estructura del modo que se describe a continuación. En el Capítulo 2 se define el problema de clustering, en su variante monoobjetivo y en su variante multiobjetivo. Se presenta la formulación matemática de cada variante, un caso de ejemplo y por último se discute la complejidad del problema. El Capítulo 3 introduce la técnica utilizada para resolver el problema: los algoritmos evolutivos. Se presentan los conceptos generales asociados a esta técnica, así como los detalles específicos al aplicarlos para resolver problemas de optimización multiobjetivo. El Capítulo 4 presenta una reseña de los principales trabajos relacionados con el problema de clustering, así como la descripción de los algoritmos que serán utilizados para la comparación con los algoritmos evolutivos implementados. Los algoritmos evolutivos implementados para resolver el problema de clustering, en sus diferentes versiones, se presentan en el Capítulo 5. En el Capítulo 6 se describe la evaluación experimental realizada sobre los algoritmos implementados y se discuten los resultados alcanzados. Por último, las conclusiones del proyecto y las principales líneas de trabajo futuro se presentan en el Capítulo 7.

Capítulo 2

Definición del problema

En este capítulo se describe el problema abordado en el marco del proyecto, que consiste en agrupar objetos según la similitud entre los mismos. Este problema es conocido como el problema de clustering. En la Sección 2.1 se presenta el problema en su formulación monoobjetivo, considerando la sumatoria de similitudes entre los objetos y el centro de cada grupo como único objetivo a optimizar. La Sección 2.2 describe una variante multiobjetivo del problema, donde además de maximizar la similitud entre los objetos del grupo y su centro, se considera minimizar simultáneamente la cantidad de grupos generados. Por último, en la Sección 2.3 se estudia la complejidad computacional del problema.

2.1. Versión monoobjetivo del problema de clustering

En esta sección se presenta el problema de clustering en su versión monoobjetivo, su formulación matemática y un caso de ejemplo.

2.1.1. Introducción

Everitt *et al.* [13] describen a la tarea de agrupar objetos similares como una de las habilidades más básicas de los seres vivos. La idea de clasificar objetos similares en categorías es una idea primitiva. Por ejemplo, el hombre debía ser capaz de reconocer características similares en alimentos para reconocer si éstos eran comestibles o venenosos. Según Everitt, la clasificación en su sentido más amplio es necesaria para el desarrollo del lenguaje, que consiste en palabras que ayudan a reconocer y distinguir diferentes tipos de eventos, objetos y personas.

Las primeras técnicas de agrupamiento fueron desarrolladas para la biología y la zoología, como mencionan Jain y Dubes [19], con el fin de agrupar animales y plantas similares construyendo así taxonomías. Everitt utiliza como ejemplo a Aristóteles, quien elaboró un sistema para clasificar a las especies del reino animal, comenzando por dividir a los animales en dos grupos, vertebrados e invertebrados.

Jain y Dubes sostienen que la necesidad de organizar grandes cantidades de datos en grupos, categorías, particiones o clases significativas en varias disciplinas científicas ha hecho del clustering una herramienta valiosa en el análisis de datos. Otras aplicaciones del clustering son enumeradas por Everitt, por ejemplo: la estrategia de marketing de dividir clientes en grupos homogéneos, la clasificación de estrellas según cierto criterio estadístico

para la astronomía, la categorización de diagnósticos para la psiquiatría, la clasificación del clima. Mecca *et al.* [28] también utilizan una técnica de clustering aplicada al *Web mining*, clasificando los resultados de una búsqueda en la Web. Nisha *et al.* [30] afirman que agrupar es una fase importante en *data mining* y definen al análisis cluster como un método para detectar el número de grupos en un conjunto de datos dado.

Tan *et al.* [35] clasifican a los distintos tipos de agrupamiento en tres criterios de categorización:

Jerárquico o particional. El clustering particional sugiere una división simple de conjuntos de datos en subconjuntos no solapados, donde un objeto se encuentra en exactamente un subconjunto. Por el contrario, si se permite que los grupos tengan subgrupos, se obtiene un clustering jerárquico, que es un conjunto de grupos anidados organizados como un árbol.

Exclusivo, solapado o difuso. Si cada objeto es asignado a un solo grupo el agrupamiento es exclusivo. En el caso de que un objeto pueda pertenecer simultáneamente a más de un grupo el agrupamiento es solapado o no exclusivo. Para el agrupamiento difuso se define un "membership weight" (mw) entre 0 (no pertenece) y 1 (pertenece). Para la categoría difusa, todos los objetos pertenecen a todos los conjuntos con un mw. En general se impone que la suma de mw para un elemento con todos los grupos es 1.

Completo y parcial. Un agrupamiento completo, a diferencia de un agrupamiento parcial, asigna todos los objetos a un mismo grupo.

Tan *et al.* [35] también plantean diferentes formas de agrupar los datos,

- *Well-Separated*: La similitud entre objetos de un grupo es mayor que la similitud entre objetos de grupos distintos.
- *Prototype-Based*: La similitud entre un objeto del grupo y su centro es mayor que la similitud de ese objeto con el centro de otro grupo.
- *Graph-Based*: Si los datos se representan como un grafo, donde los nodos son objetos y las aristas representan las conexiones entre los objetos, el grupo se representa como los objetos que están conectados a otros.
- *Density-Based*: El grupo es la región densa de objetos rodeada de una región de baja densidad.
- *Shared-Property*: El grupo contiene a los objetos que comparten una propiedad.

Siguiendo la clasificación anterior, en el caso de estudio la agrupación es particional, exclusiva y completa, donde los datos se agrupan de forma "Prototype-Based".

Kaufman y Rousseeuw [21] diferencian dos estructuras de entrada para el problema del clustering. La primer estructura representa a cada objeto por medio de valores de p propiedades. Siendo n la cantidad de elementos a agrupar, esta estructura de entrada se puede representar como una matriz de dimensiones $n \times p$, donde las filas corresponden a los elementos y las columnas a los atributos. Según la terminología de Tucker [37], este tipo de entrada se denomina *two-mode*. La estructura de entrada anterior se ejemplifica en la matriz de 3×3 (2.1), donde el valor $p_{i_{nj}}$ representa la propiedad i del elemento j.

$$\begin{pmatrix} p_{1_{n1}} & p_{2_{n1}} & p_{3_{n1}} \\ p_{1_{n2}} & p_{2_{n2}} & p_{3_{n2}} \\ p_{1_{n3}} & p_{2_{n3}} & p_{3_{n3}} \end{pmatrix} \tag{2.1}$$

La segunda estructura de entrada propuesta por Kaufman y Rousseeuw representa la proximidad entre pares de objetos. Si n es la cantidad de objetos a agrupar, esta estructura consiste en una matriz de dimensiones $n \times n$, denominada *one-mode*, según la terminología de Tucker. En la matriz (2.2) se ejemplifica la estructura descrita anteriormente, donde el valor $s(n_i, n_j)$ representa la proximidad entre el objeto n_i y n_j.

$$\begin{pmatrix} s(n_1,n_1) & s(n_1,n_2) & s(n_1,n_3) \\ s(n_2,n_1) & s(n_2,n_2) & s(n_2,n_3) \\ s(n_3,n_1) & s(n_3,n_2) & s(n_3,n_3) \end{pmatrix} \tag{2.2}$$

Se deben considerar dos tipos de proximidad, la disimilitud (cuán lejos se encuentran los objetos entre sí) y la similitud (cuánto se asemejan los objetos entre sí). Los valores de la matriz de similitudes cumplen las siguientes condiciones:

- $0 \leq s(i,j) \leq 1$
- $s(i,i) = 1$
- $s(i,j) = s(j,i)$

Para el caso de estudio se utiliza la estructura *one-mode* y la similitud entre objetos como tipo de proximidad.

Las siguientes consideraciones deben tenerse en cuenta en el modelo del problema:

- Cada cluster debe tener al menos un elemento.
- Cada elemento debe pertenecer a un cluster.
- Debe existir al menos un cluster.
- La cantidad de clusters debe ser menor que la cantidad de elementos a agrupar.

2.1.2. Formulación matemática

A continuación se presenta la formulación matemática para el problema de clustering. Dados los siguientes elementos:

- Un conjunto $O = \{o_1, o_2, ..., o_n\}$ de objetos a agrupar.
- Una función $s : O \times O \rightarrow [0, 1]$, donde $s(o_i, o_j)$ indica la similitud entre o_i y o_j. Se cumple que $\forall o_i, o_j, s(o_i, o_j) = s(o_j, o_i)$ y $s(o_i, o_i) = 1$.
- Un entero $k, k > 0$, que indica la cantidad de grupos a formar.

El problema consiste en asignar los objetos de O en un conjunto de grupos $G = \{G_1...G_k\}$, donde $G_i = \{c_i\} \cup \{o_m / s(o_m, c_i) \leq s(o_m, c_j) \forall o_m \in O, c_j, c_i \in C, i \neq j\}$, donde $C \subseteq O$, $|C| = k$ es el conjunto de centros que maximiza el valor de ST dado por la Ecuación 2.3 y se cumple que $\forall i, j, i \neq j, i, j \geq 1, i, j \leq k, G_i \cap G_j = \emptyset$.

$$ST = \sum_{o_i \in O} max \sum_{c_i \in C} s(o_i, c_i) \tag{2.3}$$

La formulación presentada no determina la cantidad de objetos que pertenecen a un grupo, sin embargo establece que como mínimo debe haber un elemento por grupo.

2.1.3. Caso de ejemplo

A continuación se describe una instancia de ejemplo del problema de clustering.

Considérese la matriz M (2.4), en la cual se representan las similitudes de los objetos a agrupar, $n = 5$ la cantidad de objetos a agrupar y $k = 2$ la cantidad de grupos.

$$M = \begin{pmatrix} 1 & 0,8 & 0,4 & 0,3 & 0,9 \\ 0,8 & 1 & 0,2 & 0,7 & 0,5 \\ 0,4 & 0,2 & 1 & 0,1 & 0,4 \\ 0,3 & 0,7 & 0,1 & 1 & 0,8 \\ 0,9 & 0,5 & 0,4 & 0,8 & 1 \end{pmatrix} \tag{2.4}$$

La solución al problema consiste en encontrar dos grupos, G_1 y G_2, tal que se maximice la similitud entre los objetos pertenecientes a cada grupo. Una posible agrupación consta de $G_1 = \{1, 3, 5\}$ y $G_2 = \{2, 4\}$, con el valor de $F = 4,0$. Sin embargo, existe una mejor solución cuyo valor de $F = 4,2$, con la agrupación $G_1 = \{1, 2, 4, 5\}$ y $G_2 = \{3\}$, siendo los centros $g_1 = 5$ y $g_2 = 3$. La Tabla 2.1 muestra para todos los posibles pares de centros la agrupación que tiene el mayor valor de ST.

Tabla 2.1: Cálculo de ST para las mejores agrupaciones con dos centros posibles.

Centros	Grupos	Valor de ST
(1,2)	G_1={1,3,5} G_2={2,4}	1+1+0,4+0,7+0,9=4,0
(1,3)	G_1={1,2,4,5} G_2={3}	1+0,8+1+0,3+0,9=4,0
(1,4)	G_1={1,2,3,5} G_2={4}	1+0,8+0,4+1+0,9=4,1
(1,5)	G_1={1,2,3} G_2={4,5}	1+0,8+0,4+0,4+1=4,0
(2,3)	G_1={1,2,4,5} G_2={3}	0,8+1+1+0,7+0,5=4,0
(2,4)	G_1={1,2,3} G_2={4,5}	0,8+1+0,2+1+0,8=3,8
(2,5)	G_1={2} G_2={1,3,4,5}	0,9+1+0,4+0,8+1=4,1
(3,4)	G_1={1,3} G_2={2,4,5}	0,4+0,7+1+1+0,8=3,9
(3,5)	G_1={3} G_2={1,2,4,5}	**0,9+0,5+1+0,8+1=4,2**
(4,5)	G_1={2,4} G_2={1,3,5}	0,9+0,7+0,4+1+1=4,0

En la Tabla 2.1 se observa que el mejor par de centros es (3,5) dado que el valor de ST es el máximo. Resulta evidente que al aumentar la cantidad de elementos de la instancia, se incrementa rápidamente la cantidad de posibilidades que se deben evaluar para cubrir todo el espectro de agrupaciones.

2.2. Versión multiobjetivo del problema de clustering

Esta sección presenta el problema de clustering en su variante multiobjetivo, donde se busca minimizar la similitud entre los elementos del grupo y su centro, y minimizar la cantidad de grupos.

2.2.1. Introducción

Como se vio en el capitulo anterior, maximizar la similitud entre los elementos de un grupo es un claro objetivo en el problema de clustering. En la variante multiobjetivo del problema no se conoce a priori el número óptimo de grupos a formar. Además, se da al usuario la posibilidad de comparar soluciones con diferente número de grupos y diferentes valores de similitud, encontrando diferentes niveles de compromiso entre ambos objetivos.

Para el problema de clustering en el que se enfoca este trabajo, se busca minimizar la cantidad de grupos. Al aumentar la cantidad de grupos, éstos tienen una mayor probabilidad de sufrir de *"empty clustering"* [20], esto es un grupo cuyo único elemento es el centro. Una solución con estas características es de mala calidad debido a que no se está realizando ninguna agrupación.

2.2.2. Formulación matemática

A continuación se presenta la formulación matemática para el problema de clustering en su variante multiobjetivo. Dados los siguientes elementos:

- Un conjunto $O = \{o_1, o_2, ..., o_n\}$ de objetos a agrupar.
- Una función $s : O \times O \rightarrow [0, 1]$, donde $s(o_i, o_j)$ indica la similitud entre o_i y o_j. Se cumple que $\forall o_i, o_j, s(o_i, o_j) = s(o_j, o_i)$ y $s(o_i, o_i) = 1$.

El problema consiste en asignar los objetos de O en un conjunto de grupos $G = \{G_1...G_m\}$, $G_i = \{c_i\} \cup \{o_k / s(o_k, c_i) \leq s(o_k, c_j) \forall o_k \in O, c_j, c_i \in C, i \neq j\}$, donde $C \subseteq O$, $|C| = m$ es el conjunto de centros. Se busca simultáneamente minimizar $|C|$ y maximizar el valor de ST dado por la Ecuación 2.3. Se cumple que $\forall i, j, i \neq j, i, j \geq 1, i, j \leq k, G_i \cap G_j = \emptyset$.

2.2.3. Caso de ejemplo variante multiobjetivo

Para el caso de ejemplo de la variante multiobjetivo del problema de clustering, se utiliza la matriz M definida en la Sección 2.1.3. La Tabla 2.2 presenta para cada valor de cantidad de grupos, la mejor agrupación encontrada tal que maximice el valor de ST. Cada una de las filas de la tabla se obtiene mediante el cálculo presentado en la sección anterior.

Tabla 2.2: Soluciones óptimas para diferentes valores de cantidad de grupos.

Cantidad de grupos	Centros	Grupos	Valor de ST
1	(5)	G_1={1,2,3,4,5}	0,9+0,5+0,4+0,8+1=3,9
2	(3,5)	G_1={3} G_2={1,2,4,5}	0,9+0,5+1+0,8+1=4,2
3	(1,3,4)	G_1={1,2,5} G_2={3} G_3={4}	1+0,8+1+1+0,9=4,7
4	(1,2,3,4)	G_1={1,5} G_2={2} G_3={3} G_4={4}	1+1+1+1+0,9=4,9
5	(1,2,3,4,5)	G_1={1} G_2={2} G_3={3} G_4={4} G_5={5}	1+1+1+1+1=5

La Tabla 2.2 muestra que el hecho de maximizar el valor de F se contrapone con minimizar la cantidad de grupos. Por lo tanto, no existe una única solución óptima, sino un conjunto de soluciones con distintos niveles de compromiso entre la similitud y la cantidad de grupos.

2.3. Complejidad del problema

En esta sección se justifica que el problema de clustering es de clase NP-difícil. Los problemas NP-difícil son una clase de problemas que son al menos tan complejos como el más difícil de los problemas en NP, siendo NP la clase formada por aquellos problemas para los cuales una solución puede ser verificada en tiempo polinomial por una máquina de Turing determinística.

Dasgupta *et al.* [7] prueban que el problema donde, dado un conjunto finito S de puntos en R^m y un entero $k = 2$, se busca encontrar los centros que minimicen la distancia Euclidiana entre cada punto de S con su centro k es un problema NP-difícil. Mahajana *et al.* [26] amplían el trabajo de Dasgupta *et al.* demostrando que, siendo k un entero y $k \geq 1$, el problema sigue siendo NP-difícil. Aloise [4] generaliza aun más la definición y demuestra que los problemas de clustering son de la clase NP-difícil.

Theodoridis y Koutroumbas [36] realiza un análisis sobre la posibilidad de resolver el problema del clustering seleccionando la mejor agrupación de todas las agrupaciones posibles. Siendo $S(n, m)$ todas las posibles agrupaciones de n elementos en m clusters, se cumple:

- $S(n, 1) = 1$
- $S(n, n) = 1$
- $S(n, m) = 0$ para $m > n$

Sea una lista L_{n-1}^k que contiene las posibles agrupaciones para $n-1$ objetos en k clusters, el n-ésimo objeto puede:

- Ser agregado a uno de los grupos ya existentes en L_{n-1}^k.
- O formar un nuevo grupo.

De esta manera, la cantidad de agrupaciones se podría escribir como:

$$S(n,m) = m(S(n-1,m)) + S(n-1,m-1) \tag{2.5}$$

Para visualizarlo en un ejemplo, sea $X = \{x_1, x_2, x_3\}$ y $m = 2$, se deduce que

$$L_2^1 = \{\{x_1, x_2\}\} \text{ y } L_2^2 = \{\{x_1\}, \{x_2\}\} \tag{2.6}$$

Utilizando la Ecuación 2.5 se calcula que $S(3,2) = 2(1) + 1 = 3$, por lo tanto

$$L_3^2 = \{\{\{x_1, x_3\}, \{x_2\}\}, \{\{x_1, x_2\}, \{x_3\}\}, \{\{x_2, x_3\}, \{x_1\}\}\} \tag{2.7}$$

Los autores concluyen que particularmente para $m = 2$, $S(n,2) = 2^{n-2} - 1$. Si se utiliza como ejemplo la agrupación de 100 elementos en 5 clusters, y la evaluación de un agrupamiento demora 10^{-12} segundos, entonces evaluar todas las agrupaciones posibles se realizaría en aproximadamente 10^{48} años. Esto demuestra la limitación de probar todo el espectro de soluciones al aumentar el tamaño de la instancia del problema.

Tomando en cuenta la complejidad del problema, cuando se utilizan instancias de tamaños realistas, los algoritmos exactos tradicionales no resultan útiles para un agrupamiento eficiente. Por este motivo, se propone utilizar heurísticas y metaheurísticas que permitan calcular soluciones de calidad aceptable en tiempos razonables.

Capítulo 3

Algoritmos evolutivos

En este capítulo se describe la técnica utilizada para resolver el problema de clustering en sus variantes monoobjetivo y multiobjetivo. En la Sección 3.1 se detallan los conceptos principales de los algoritmos evolutivos. En la Sección 3.2 se introducen los algoritmos evolutivos multiobjetivos como solución a los problemas de optimización multiobjetivo.

3.1. Algoritmos evolutivos

En esta sección se presentan los algoritmos evolutivos como técnica de resolución a los problemas de optimización. En la Sección 3.1.1 se introduce la historia de los AE y en las restantes secciones se describen los detalles generales correspondientes a los AE.

3.1.1. Introducción

Los métodos de resolución exactos no son capaces de resolver eficientemente instancias de dimensiones grandes para los problemas del tipo NP-difícil, debido a que la complejidad del algoritmo que lo resuelve crece de forma superpolinomial a medida que crece la entrada. Glover *et al.* [15] afirman que muchos problemas de optimización son NP-difícil y las heurísticas y metaheurísticas son alternativas para su resolución eficiente. Dado que los algoritmos evolutivos en particular son un ejemplo de metaheurísticas, se utilizaron AE para resolver el problema presentado.

Los AE son técnicas iterativas donde se aplican operadores estocásticos sobre un conjunto de individuos. Cada individuo de la población codifica una solución posible del problema y tiene asociado un valor de fitness, resultado de una función de evaluación que determina su adecuación para resolver el problema. Las iteraciones dentro del proceso son denominadas generaciones y el conjunto de individuos es la población. La población inicial es generada a través de un procedimiento aleatorio o utilizando una heurística específica para el problema a resolver. La finalidad del AE es mejorar el fitness de los individuos en la población mediante la aplicación iterativa de operadores evolutivos. Los operadores evolutivos son aplicados a individuos seleccionados según su fitness. De esta manera, se guía la búsqueda hacia soluciones tentativas de mayor calidad. Además, los AE se diferencian de los algoritmos tradicionales según Goldberg [16] en los siguientes puntos:

1. Los AE buscan la resolución en poblaciones de soluciones en lugar de soluciones individuales, buscando evitar estancarse en óptimos locales al problema.

2. Mientras otras técnicas requieren de información auxiliar, los AE solo utilizan la función objetivo.

3. Los AE utilizan reglas de transición basadas en probabilidades, no deterministas.

Como introduce Mitchell [29], en el período abarcado entre 1950 y 1960 se estudiaron los sistemas evolutivos con el objetivo de utilizar una analogía a la evolución natural de las especies como una herramienta de optimización para los problemas de ingeniería. La idea subyacente es "evolucionar" una población de soluciones candidatas con operadores inspirados en la selección natural.

Fue Holland [18] quien en 1975 formalizó la teoría de la computación evolutiva. Holland afirma que todos los organismos son una amalgama de características determinadas por los genes en sus cromosomas. Además, expresa que la gran cantidad de estructuras posibles (genotipos) para una determinada especie es un indicador de la complejidad de dichos sistemas.

Goldberg [16] sostiene que los algoritmos evolutivos simples se componen de tres operadores: reproducción, cruzamiento y mutación. En la clasificación anterior, la operación de selección es una versión artificial de la selección natural de Darwin [6]. El cruzamiento involucra el intercambio de genes entre dos individuos. Tanto la reproducción como el cruzamiento involucran generación aleatoria de números, clonación e intercambio parcial entre individuos. La mutación es la alteración aleatoria del valor de un gen. Goldberg considera a la selección y al cruzamiento como los operadores más relevantes, dejando a la mutación como un operador secundario. Sin embargo, la mutación es necesaria porque la reproducción y el cruzamiento pueden perder información genética útil y la mutación provee diversidad, evitando la convergencia a soluciones sub-óptimas. El procedimiento estándar de un AE se detalla a continuación. Cada paso de este proceso se denomina generación, se considera P_i a la población en la generación i.

Paso 1. Inicializar P_0

Paso 2. Mientras no se cumpla el criterio de parada repetir de *a)* hasta *d)*

   a) Evaluar población P_i
   b) Seleccionar los padres
   c) Obtener los hijos a partir de aplicar los operadores en los padres
   d) Obtener nueva población P_{i+1} reemplazando los padres por los hijos de la población actual P_i

Paso 3. Retornar el mejor individuo encontrado

Existen varias opciones para determinar el criterio de parada. Entre ellas se encuentra el umbral del valor del fitness, convergencia del valor del fitness, convergencia de la población y límite de la cantidad de generaciones alcanzada. El último es el que se utiliza en este proyecto para la resolución del problema de clustering.

3.1.2. Representación

Resulta útil conocer la terminología biológica básica para comprender las posibles representaciones que se pueden utilizar en los algoritmos evolutivos. El término *cromosoma* hace referencia a la codificación de una solución del problema. El cromosoma está

compuesto de *genes* que codifican las características particulares de la solución. Los diferentes valores que puede adoptar un gen se denominan *alelos*. El *genotipo* refiere a un conjunto particular de genes contenidos en un individuo, es decir que dos individuos son idénticos si tienen el mismo genotipo. El genotipo da lugar al *fenotipo*, este último está compuesto por las características específicas de un individuo.

La representación codifica una solución al problema. La longitud de esta representación dependerá de las características del problema y de la codificación de la solución. Las representaciones estudiadas para el problema de clustering son la representación binaria y la entera. Ambas se describen a continuación.

Representación binaria. En la representación binaria cada solución es simbolizada como un vector binario. Cada gen debe adoptar un valor del conjunto {0,1}. El largo de la tupla depende del problema a resolver. Por ejemplo, una representación binaria posible para el problema de clustering es un vector binario de largo N, donde N es la cantidad de elementos a agrupar. Para representar que el elemento i es centro de un grupo, el vector contiene un 1 en el lugar i. Se fija la cantidad de grupos (K) en la solución con la cantidad de unos del vector. Por ejemplo, para agrupar nueve elementos en tres grupos, una representación posible se presenta en la Figura 3.1.

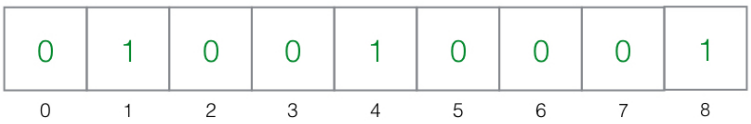


Figura 3.1: Ejemplo representación binaria para el problema del Clustering

Representación entera. En la representación entera cada solución es simbolizada como un vector de números enteros. Por ejemplo, para el problema de clustering una posible representación es un vector de tamaño K, donde se almacenan los centros que resuelven el problema. Por ejemplo, para agrupar treinta elementos en siete grupos, una representación posible se encuentra en la Figura 3.2.

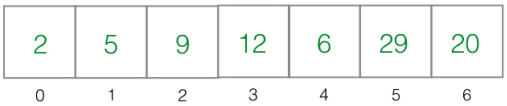


Figura 3.2: Ejemplo representación entera

3.1.3. Función de Fitness

Los AE requieren una función de fitness que asigne un valor a cada cromosoma en una población, dependiendo de que tan bien resuelva el problema. Mitchell define el fitness de un individuo como la probabilidad de que ese individuo viva para reproducirse (viabilidad) o, como una función del número de descendientes que tiene ese organismo (fertilidad). Dependiendo del problema que se quiere resolver esta función podrá minimizar o maximizar la función objetivo.

Dado que las soluciones están codificadas, no se garantiza que todos los genotipos tengan un fenotipo factible en el dominio del problema asociado. Si se encuentra un

genotipo cuya representación no es válida para el problema se pueden tomar tres acciones:

1. Corregir al individuo mediante una transformación que devuelva una solución válida.

2. Penalizar al individuo, asignándole un peor valor de fitness, de forma que la probabilidad de sobrevivir al proceso evolutivo sea mínima.

3. Descartar al individuo.

3.1.4. Operadores evolutivos

Según Holland, los sistemas deben continuamente probar e incorporar propiedades estructurales asociadas a la mejora del rendimiento. El objetivo de los operadores evolutivos es encontrar nuevas estructuras que tengan una alta probabilidad de mejorar el rendimiento. Mitchell afirma que el algoritmo genético más simple involucra tres operadores: selección, cruzamiento y mutación.

Selección. El operador de selección, como su nombre lo indica, es el encargado de seleccionar individuos de la población para la reproducción. Como se menciona anteriormente, la viabilidad de un individuo (es decir, la probabilidad de ser seleccionado) depende de su valor de fitness. Si su aptitud es mayor, aumenta la probabilidad de ser seleccionado.

Cruzamiento. Según Mitchell, el operador de cruzamiento imita a la recombinación biológica entre dos cromosomas. Holland también hace referencia a los sistemas biológicos al describir al operador de cruzamiento, describiendo al cruzamiento como el proceso de recombinación de alelos a través del intercambio de segmentos entre pares de cromosomas.

Mutación. Holland afirma que en el proceso de mutación genética, un alelo de un gen es reemplazado aleatoriamente por otro, retornando una nueva estructura.

3.2. Algoritmos evolutivos para problemas mulitobjetivo

En esta sección se describen los algoritmos evolutivos multiobjetivo como método para resolver problemas de optimización con dos o más objetivos. Además, se presenta la técnica utilizada para resolver el problema del clustering con dos objetivos.

3.2.1. Problemas de optimización multiobjetivo

En los problemas de optimización multiobjetivo (*Multiobjective optimization problems*, MOPs) se opera sobre un espacio multidimensional de funciones vectoriales. Deb [9], utilizando como ejemplo dos objetivos, afirma que no es posible encontrar una única solución óptima a ambas funciones. Además, sugiere que la diferencia entre los problemas monoobjetivo y multiobjetivo radica en la cantidad de soluciones: mientras el primero tiene una solución óptima, el segundo tiene un conjunto de soluciones optimas. Deb enfatiza la importancia de la compensación en los objetivos de las soluciones, es decir que una solución no es mejor que otra en tanto no sea mejor en los dos objetivos. A su vez, Goldberg diferencia los problemas monoobjetivo de los problemas multiobjetivo en base a su definición de óptimo. Mientras que en los primeros se busca maximizar o minimizar la función que se considera objetivo, en los segundos se debe llegar a una definición de óptimo que considere cada uno de los diferentes objetivos.

Para un caso genérico de optimización, la formulación de un problema multiobjetivo se muestra en la Ecuación 3.1.

$$\begin{array}{ll} Max, Min & \vec{F}(x) = (f_1(x), f_2(x), ..f_n(x)) \\ \text{sujeto a} & \vec{G}(x) = (g_1(x), g_2(x), ..g_s(x)) \geq 0 \\ & \vec{H}(x) = (h_1(x), h_2(x), ..h_r(x)) = 0 \end{array} \tag{3.1}$$

siendo $x = (x_1, x_2, ...x_m)$ las variables de decisión del problema
y siendo $x \in \Omega$ el espacio de soluciones factibles

Concepto de dominancia. En los algoritmos de optimización multiobjetivo, dos soluciones son comparadas en base a si una solución domina a la otra. Según Deb [9], una solución x domina a otra solución y si se cumplen las siguientes condiciones:

1. x no es peor que y en ningún objetivo.

2. x es estrictamente mejor que y en al menos un objetivo.

El conjunto de soluciones no dominadas, perteneciente al conjunto de soluciones posibles, cuenta con la propiedad de que para dos soluciones no dominadas la ganancia en un objetivo solo es posible si se sacrifica al menos otro objetivo. Este conjunto de soluciones no dominadas constituye un frente de soluciones no dominadas denominado *conjunto óptimo de Pareto*. La Figura 3.3 ejemplifica los conceptos de soluciones no dominadas y frente de Pareto (FP) para un MOP con dos objetivos, donde se busca maximizar el objetivo 1 y minimizar el objetivo 2.

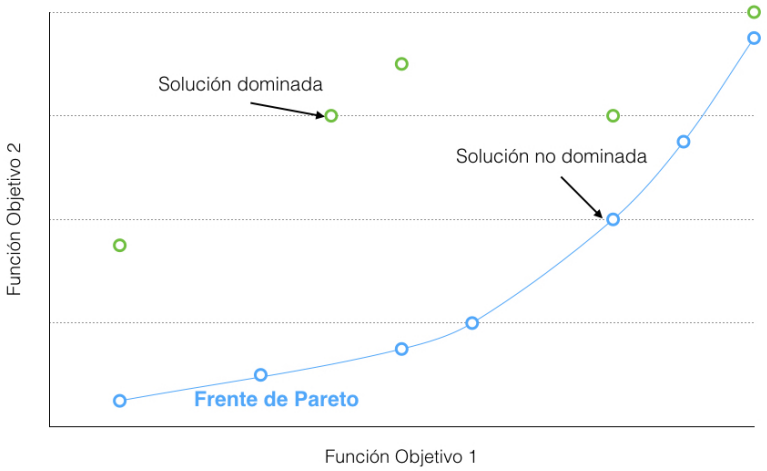


Figura 3.3: Soluciones dominadas, no dominadas y frente de Pareto en un MOP con dos objetivos

Por ejemplo, para el problema de optimización estudiado se cuenta con dos objetivos, maximizar la similitud entre el centro del grupo y sus elementos y minimizar la cantidad de grupos. En el alcance del problema de clustering, se busca que la solución sea heterogénea entre grupos y a su vez homogénea intra grupo.

3.2.2. Algoritmo evolutivo para optimización multiobjetivo: NSGA-II

Según Deb [10], en los MOP la búsqueda del óptimo no puede aplicarse a un solo objetivo, cuando los otros objetivos son también importantes. Deb sugiere entonces dos propósitos para los MOP, convergencia y diversidad.

- *Convergencia*: encontrar un conjunto de soluciones lo más cercanas posibles al FP real del problema.
- *Diversidad*: encontrar un conjunto de soluciones que sea lo suficientemente diverso para representar el rango entero del frente de Pareto.

Como los algoritmos evolutivos multiobjetivo (*Multiobjective evolutionary algorithms*, MOEA) utilizan heurísticas, estos procedimientos no garantizan encontrar los puntos exactos del frente de Pareto. Sin embargo, los MOEA contienen operadores que buscan, desde la perspectiva de la convergencia y la diversidad, mejorar los puntos no dominados para aproximarse al frente óptimo de Pareto.

El algoritmo genético de búsqueda no dominada (*Nondominated Sorting Genetic Algorithm, NSGA*) fue propuesto por Deb y Srinivas [34] en el año 1994. NSGA se diferencia del algoritmo genético simple en el operador de selección. Antes de realizar la selección se clasifica a la población en base a la dominancia. Los individuos no dominados son identificados dentro de la población actual. Estos individuos no dominados constituyen el primer frente no dominado de la población y se les asigna un fitness ficticio alto. Para mantener la diversidad se utiliza un método de *sharing*. Este método se alcanza utilizando en la selección valores de fitness "degradados", obtenidos de dividir el valor del fitness original entre la cantidad de individuos que se encuentran alrededor de ese valor. Un ejemplo de la función de *sharing* puede encontrarse en la Figura 3.4.

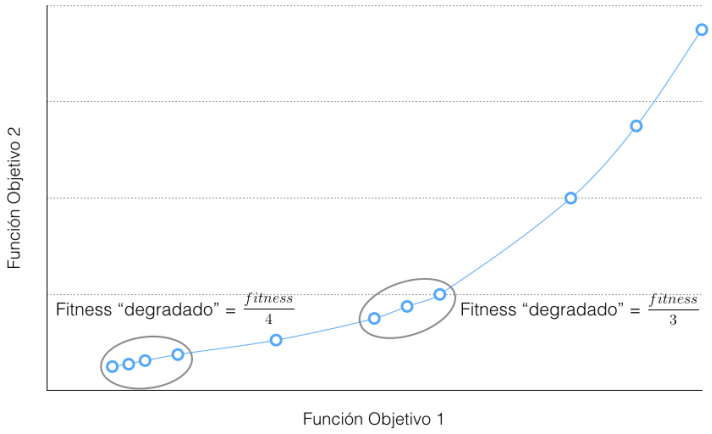


Figura 3.4: Fitness "degradado"para la función de *sharing*

NSGA fue criticado por tres razones:

1. Tiene un alto costo computacional al utilizar poblaciones grandes.
2. No es elitista, por lo tanto no preserva las mejores soluciones durante la selección.
3. Requiere la especificación de parámetros compartidos para asegurar la diversidad en las poblaciones.

Fue por estas críticas que en el año 2002, Deb *et al.* [11] proponen una mejora a NSGA que denominan NSGA-II. El nuevo algoritmo NSGA-II cambia la técnica de *sharing* por una técnica de *crowding*. El operador comparativo de *crowding* guía al proceso de selección a través de las diferentes etapas del algoritmo hacia un frente de Pareto óptimo uniforme. El indicador de *crowding* modela a la densidad de las soluciones. De esta manera se buscan soluciones dispersas y cuyo rango sea el menor. La Figura 3.5 muestra el mecanismo de asignación de rangos en soluciones no dominadas.

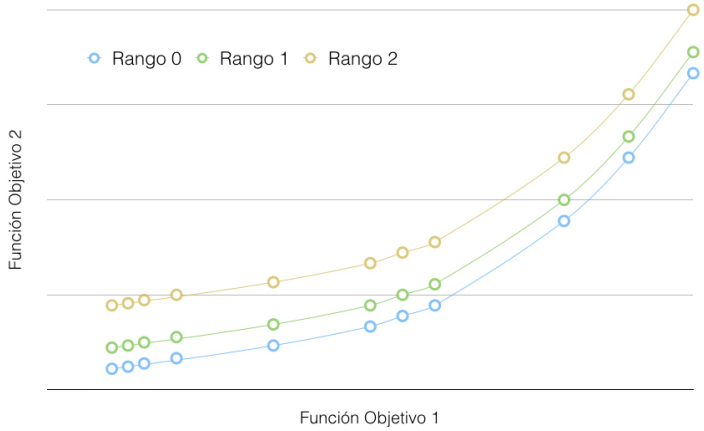


Figura 3.5: Rangos en frentes de pareto NSGA II

El algoritmo NSGA-II es el seleccionado para el caso de estudio ya que ha sido aplicado en diversos problemas, obteniendo muy buenos resultados en la práctica. El Algoritmo 1 muestra el pseudocódigo para el algoritmo genético de búsqueda no dominada versión 2.

Algoritmo 1 NSGA-II

```
1: inicializar(P_0)
2: Evaluar(P_0)
3: mientras no criterio_de_parada hacer
4:    R= Padres ∪ Hijos
5:    Frentes = ordenamiento_no_dominado(R)
6:    nueva_pob = Ø
7:    i=1
8:    mientras |nueva_pob| + |Frentes(i)| ≤ tamaño_población hacer
9:       Distancia_de_crowding(Frentes(i))
10:      nueva_pob = nueva_pob ∪ Frentes(i)
11:      i++
12:   fin mientras
13:   ordenamiento_por_distancia(Frentes(i))
14:   nueva_pob = nueva_pob ∪ Frentes(i)[1:(tamaño_población -|nueva_pob|)]
15:   Hijos = Selección_y_reproducción(nueva_pob)
16:   gen++
17:   P_gen =nueva_pob
18: fin mientras
```

Capítulo 4

Trabajos relacionados

En este capítulo se describen los principales trabajos relacionados al problema de clustering abordado en este proyecto. La Sección 4.1 presenta trabajos que se enfocan en resolver el problema utilizando algoritmos de búsqueda basados en trayectoria, donde en cada iteración se cuenta únicamente con una solución tentativa al problema. La Sección 4.2 describe los principales estudios que utilizan búsquedas basadas en una población de soluciones candidatas, en particular utilizando algoritmos evolutivos. La Sección 4.3 presenta un breve resumen del análisis de la literatura relacionada al problema abordado en este proyecto. Finalmente, en la Sección 4.4 se presentan las diferentes implementaciones, basadas en los trabajos relacionados, que se realizaron en el proyecto para la comparación con los AE, que se describen en el Capítulo 5.

4.1. Algoritmos basados en trayectoria para el problema del clustering

Existen algoritmos que resuelven el problema de clustering, la principal desventaja que tienen estos algoritmos es que están limitados al tamaño de la instancia a resolver.

Deng y Bard [12] proponen utilizar el algoritmo *Greedy Randomized Adaptive Search Procedure* (GRASP) para resolver el problema de clustering con una capacidad limitada (*Capacitated Clustering Problem*, CCP) en un tiempo preestablecido. El CCP tiene como objetivo agrupar elementos en clusters, donde cada cluster tiene una capacidad mínima y máxima de elementos. El algoritmo que plantean los autores se divide en cuatro fases:

1. *Inicialización.* Se utilizan dos estrategias de inicialización con la motivación de identificar aquellos nodos que, para una partición óptima, no pertenecen al mismo grupo.

2. *Local Search.* Se utiliza una búsqueda local para encontrar una solución óptima local.

3. *Selección de un subconjunto de soluciones.* A partir de los óptimos locales se selecciona un subconjunto denominado *Elite Pool (EP).*

4. *Path Relinking (PR).* El objetivo de PR es construir posibles mejores soluciones a partir de pares de soluciones del EP.

Para probar GRASP los autores utilizaron tres conjuntos de datos:

- 30 nodos agrupados en 5 clusters, donde cada cluster debía tener entre 5 a 8 elementos. Para esta instancia se realizaron 150 iteraciones.
- 40 nodos agrupados en 5 clusters, donde cada cluster debía tener entre 5 a 9 elementos. Para esta instancia se realizaron 200 iteraciones.
- 50 nodos agrupados en 5 clusters, donde cada cluster debía tener entre 5 a 12 elementos. Para esta instancia se realizaron 250 iteraciones.

GRASP se comparó con el algoritmo exacto CPLEX. CPLEX [1] es una herramienta para resolver problemas de optimización lineal elaborada por IBM. En todas las instancias de testing GRASP obtuvo la mismas o mejores soluciones que CPLEX en el tiempo establecido. Al incrementar la cantidad de nodos, aristas y grupos aumenta la dificultad de encontrar el óptimo. En el tercer conjunto de datos, donde la cantidad de nodos asciende a 50, CPLEX no logró la convergencia a la solución óptima en el límite de tiempo establecido (1 hora). Los autores llegaron a la conclusión que, para instancias pequeñas, PR no era necesario ya que se alcanzaba el óptimo sin éste y, en instancias grandes, PR obtenía mejoras insignificantes.

Kaufman y Rousseeuw [21] presentan el algoritmo *Partitioning Around Methods* (PAM) para resolver el problema de clustering. En el algoritmo PAM se utilizan los elementos de la instancia como centros y la distancia entre los elementos como medida para crear los grupos. El procedimiento que los autores plantean se divide en dos fases, *BUILD* y *SWAP*. La fase BUILD se encarga de seleccionar los centros de la siguiente manera:

1. Seleccionar el primer centro de forma tal que la suma de la distancia a todos los objetos de la instancia restantes sea la menor.
2. Agregar centros a la solución de la siguiente manera, hasta alcanzar la cantidad de centros establecida:
   a) Considerar el elemento i no seleccionado.
   b) Considerar un elemento j no seleccionado, $j \neq i$
   c) Calcular la distancia entre el centro más similar elegido previamente y j, denominada D_j.
   d) Calcular la distancia entre i y j, denominada $d(i, j)$.
   e) Obtener C_{ij} tal que $C_{ij} = \text{máx}\{D_j - d(i, j), 0\}$.
   f) Calcular $\sum_j C_{ij}$.
   g) Elegir i no seleccionado tal que $\sum_j C_{ij}$ sea máximo.

La segunda fase (SWAP) tiene como objetivo perfeccionar la solución, mediante el intercambio de pares de objetos (i, h), donde i es centro y h no lo es. Para calcular el impacto del intercambio entre dos elementos i y h se realiza el siguiente procedimiento:

1. Considerar el elemento j no seleccionado y calcular su contribución C_{jih} de la siguiente manera:
   a) Si j es más distante de i y h que de otro centro, entonces $C_{jih} = 0$.

b) Si j es más cercano a i que a otro centro, entonces:

1) Si j es más cercano a h que a el siguiente centro más cercano, $C_{jih} = d(j,h) - d(j,i)$
2) Si $d(j,h) \geq E_j$, donde E_j es la distancia entre j y el segundo centro más cercano, entonces $C_{jih} = E_j - D_j$.

c) Si j es más distante de i que de al menos otro centro, pero más cercano de h que cualquier otro centro, entonces $C_{jih} = d(j,h) - D_j$.

2. Calcular el resultado total del intercambio como $T_{ih} = \sum_j C_{jih}$.
3. Seleccionar el par (i,h) tal que T_{ih} sea mínimo.
4. Si el menor T_{ih} es negativo, se realiza el intercambio y se retorna al punto 1. El algoritmo concluye cuando el menor $T_{ih} \geq 0$.

Park y Jun [31] proponen un nuevo algoritmo para el problema de clustering. El algoritmo presentado calcula la matriz de distancias y la utiliza para encontrar nuevos centros en cada iteración. Los centros de los grupos no están representados por elementos de la instancia, sino que son coordenadas en el espacio. El algoritmo se basa en *KMeans* y compara cinco diferentes tipos de inicialización. La secuencia de pasos que sigue el procedimiento es la siguiente:

1. Inicializar.
2. Actualizar los centros de cada grupo, reemplazándolos por los objetos que minimicen la distancia total a los otros elementos en el grupo.
3. Asignar cada objeto al centro más cercano.
4. Calcular la suma de las distancias desde todos los objetos al centro correspondiente. Si la suma es igual que la obtenida en la iteración anterior, se finaliza el algoritmo. Si no, volver al paso 2.

A continuación se describen los cinco tipos de inicialización que los autores implementaron.

- Método 1: Seleccionar k objetos de forma aleatoria.
- Método 2: Ordenar los objetos según una variable del problema, dividir en k intervalos iguales la instancia y seleccionar un objeto de forma aleatoria de cada intervalo.
- Método 3: Tomar 10 % de los elementos de la muestra y sobre ellos ejecutar el algoritmo propuesto. Los k centros obtenidos son utilizados como los centros iniciales.
- Método 4: Seleccionar los k elementos más lejanos.
- Método 5:

1. Calcular la distancia entre cada par de elementos.
2. Para el objeto j calcular $v_j = \sum_{i=0}^{n} \frac{d_{ij}}{\sum_{l=1}^{n} d_{il}}$.

3. Ordenar v_j en orden ascendente. Seleccionar los primeros k elementos como centros.
4. Asignar cada objeto al centro más cercano.
5. Calcular la suma de la distancia entre todos los objetos a los centros correspondientes.

Para determinar el mejor método de inicialización los autores utilizaron instancias de prueba con $n = 3000$ elementos. Los resultados obtenidos indicaron que el Método 5 fue superior a los demás, seguido por el Método 3. Park y Jun compararon su algoritmo con el algoritmo PAM de Kaufman y Rousseeuw, con instancias no mayores a 360 elementos y 3 grupos. Los autores concluyen que a pesar de obtener los mismos resultados, el método propuesto es más eficiente computacionalmente que el algoritmo PAM.

Un algoritmo que es ampliamente utilizado en la literatura para resolver el problema de clustering, es el propuesto por MacQueen [25], denominado *KMeans*. El autor considera que el procedimiento KMeans es fácilmente programable y computacionalmente económico. Los pasos que sigue el algoritmo KMeans se describen a continuación:

1. Seleccionar aleatoriamente k grupos, cada uno constituido por un elemento.
2. Agregar un elemento al grupo cuya media sea la más cercana.
3. Recalcular la media del grupo considerando el último elemento agregado.
4. Repetir el paso 2. hasta agrupar los n elementos de la instancia.

Karimov y Ozbayoglu [20] consideran que si bien KMeans no es el algoritmo con el mejor rendimiento, afirman que es el algoritmo más utilizado debido a su simplicidad, escalabilidad y velocidad de convergencia. Sin embargo, KMeans puede estancarse en óptimos locales, debido a que no permite utilizar una solución intermedia que disminuya el criterio de optimización. Además, la solución del algoritmo KMeans está fuertemente ligada a la selección inicial.

4.2. Algoritmos evolutivos para el problema de clustering

En la literatura se encuentran trabajos relacionados que implementan algoritmos evolutivos para resolver el problema de clustering.

Bokan *et al.* [5] presentan el algoritmo *Intelligent Evolutionary Kmeans Algorithm* (IEKA) para resolver el problema de clustering. Las principales características del algoritmo propuesto son las siguientes:

- Utiliza como centros coordenadas en el espacio y es por esto que utiliza una representación real. Para medir la distancia entre objetos se utiliza la distancia euclidiana.
- La inicialización que se emplea es el resultado de la ejecución del algoritmo *KMeans*.
- Método de selección proporcional: Asigna a cada individuo una probabilidad de ser seleccionado proporcional a su valor de fitness.

- Utiliza el operador de cruzamiento de dos puntos, descrito en el capítulo anterior, y el operador de mutación uniforme. La mutación uniforme asigna al gen del individuo un valor aleatorio entre el mínimo y el máximo posibles.
- Funciones de aptitud:
  - Indice de Dunn: Esta técnica consiste en verificar que los conjuntos de clusters sean compactos y bien separados.
  - Indice de Davies-Bouldin: Es una función de la proporción entre la suma de la dispersión dentro del cluster y la separación entre clusters.

Las pruebas se realizaron sobre dos conjuntos de datos con un máximo de 300 elementos agrupados en dos y tres grupos. IEKA generó una clusterización sin solapamientos, cuyos clusters se encontraron compactos y separados.

Sheng y Liu [33] plantean tres algoritmos: *KMedoid*, *Local Search* y un algoritmo híbrido que denominan *Hybrid KMeans Algorithm* (HKA). Los primeros dos son algoritmos de búsqueda basados en trayectoria, mientras que el tercero es un algoritmo evolutivo. A continuación se describe la serie de pasos que sigue el algoritmo KMedoid, donde los centros están representados por coordenadas en el espacio.

1. Posicionar k puntos en el espacio representado por los objetos que están siendo agrupados. Estos puntos representan a los centroides de los grupos iniciales.
2. Asignar cada objeto al grupo que tiene el centro más cercano.
3. Recalcular la posición de los k centroides.
4. Repetir los pasos 2 y 3 hasta que los centroides se mantengan incambiados.

El algoritmo *Local Search* se compone principalmente de dos bucles, denominados interior y exterior. A partir de los k centros seleccionados aleatoriamente, el bucle exterior asigna cada objeto en primer lugar al grupo correspondiente con el centro más cercano. El bucle interior actualiza los k centros buscando minimizar el costo. Este procedimiento concluye cuando se alcanza la convergencia. El bucle interior utiliza una cantidad de "vecinos más cercanos" p para conseguir un subconjunto de elementos del grupo alrededor del centro e identificar aquel elemento dentro del subconjunto que minimice el costo total. A continuación, se describen los pasos que aplica el algoritmo *Local Search*.

1. Establecer la cantidad de grupos k y la cantidad de vecinos más cercanos p.
2. Seleccionar aleatoriamente los primeros k centros.
3. Asignar cada elemento al grupo C_j cuyo centro sea el más cercano, utilizando la distancia euclidiana.
4. Actualizar los k centros.
5. Para los k grupos, repetir:
   a) Seleccionar un subconjunto C_{subset} que corresponde al centro m_j y sus p vecinos más cercanos.

b) Calcular el nuevo centro como

$$q = \underset{x_k \in C_{subset}}{argmin} \sum_{x_i \in c_j} d(x_k, x_i) \tag{4.1}$$

c) Repetir *a*) y *b*) hasta que los centros se mantengan incambiados.

6. Repetir los pasos 3 y 4 hasta que los centros se mantengan incambiados.

El algoritmo evolutivo planteado en el trabajo de Sheng y Liu tiene las siguientes características:

- La representación utilizada es un vector de enteros de tamaño fijo k.
- La inicialización utilizada es del tipo aletoria sobre los k centros.
- La selección utilizada es una selección por torneo de tamaño 2.
- El operador de cruzamiento utilizado es *Mix Subset Recombination Crossover*, que sigue los pasos descritos a continuación:
  1. Concatenar ambos padres para obtener X_{mix}
  2. Reordenar aleatoriamente los elementos de X_{mix}
  3. Obtener ambos hijos de la división de X_{mix} en dos partes de igual tamaño.
- El operador de mutación utilizado es el denominado *Bit Flip*, que realiza la inversión de un bit.
- Antes de evaluar a la población se realiza un paso del algoritmo *Local Search*.
- Criterios de parada:
  - El valor del fitness se mantiene incambiado por n generaciones.
  - Se alcanza una determinada cantidad de generaciones.

Los autores evaluaron los algoritmos con dos conjuntos de datos, el primero de 517 elementos en 10 grupos mientras que el segundo contiene 2945 elementos divididos en 30 grupos. El algoritmo HKA obtuvo mejores resultados en las instancias de prueba de gran tamaño, mientras que para las pequeñas fue menos eficiente que KMedoid y Local Search.

Krishna y Narasimha [22] proponen el algoritmo *Genetic K-Means Algorithm* (GKA) con la motivación de evitar operadores costosos. GKA es un algoritmo evolutivo con las siguientes características:

- La representación utilizada es un vector de enteros de tamaño n, donde cada alelo toma un valor entre $\{1, 2, .., k\}$.
- La inicialización es aletoria.
- El método utilizado es la selección proporcional.

- Operador de cruzamiento denominado *KMeans operator* (KMO), donde se realiza un paso del algoritmo KMeans. Según la asignación se calculan los centros de cada grupo, luego se reasigna la agrupación de manera que cada elemento pertenezca al grupo con el centro más cercano.
- Operador de mutación denominado *Mutation Distance Based*, donde se mutan aquellos elementos que son cercanos al centro.
- El criterio de parada se establece en alcanzar una cantidad establecida de generaciones.

Krishna y Narasimha en su estudio compararon al algoritmo GKA con el algoritmo KMeans en dos conjuntos de datos. El primer conjunto de datos consiste de 50 elementos agrupados en 10 grupos, mientras que el segundo esta compuesto por 59 elementos divididos en 10 grupos. Los autores concluyen que el algoritmo GKA converge al óptimo global más rápido que el algoritmo KMeans.

4.3. Resumen

La Tabla 4.1 presenta un resumen de los trabajos relacionados relevados, siguiendo el orden cronológico de publicación. Se indica el autor, el año y una breve descripción con la principal contribución del trabajo.

Tabla 4.1: Trabajos relacionados al problema de clustering.

Autores	Año	Comentario
MacQueen [25]	1967	Resuelve el problema de clustering con el algoritmo KMeans
Kaufman et al. [21]	1990	Presentan el algoritmo PAM para resolver el problema de clustering
Krishna et al. [22]	1999	Proponen el algoritmo evolutivo GKA con el operador de cruzamiento basado en el KMeans
Sheng et al. [33]	2004	Proponen tres algoritmos para resolver el problema de clustering: KMedoid, Local Search y HKA.
Park et al. [31]	2009	Resuelven el problema de clustering con una heurística evaluando cinco distintos tipos de inicialización
Deng et al. [12]	2010	Resuelven el CCP con un algoritmo GRASP
Bokan et al. [5]	2011	Plantean el algoritmo evolutivo IEKA para resolver el problema de clustering

Como se puede observar en la revisión de la literatura realizada, existen técnicas variadas para resolver el problema de clustering. El método que más se destaca, ya sea por utilizarse para la comparación con otros algoritmos o incluyéndolo en los algoritmos planteados, es KMeans. Tanto este método como las otras técnicas presentadas en este capítulo, son utilizados como base para la implementación de algoritmos con los que se comparan los AE propuestos en el Capítulo 5.

4.4. Algoritmos de la literatura para la resolución del problema de clustering

Esta sección describe los algoritmos de la literatura que se implementaron para comparar los resultados obtenidos con los algoritmos evolutivos desarrollados. Para efectuar la comparación se mide el valor del fitness de la solución obtenida por cada algoritmo.

4.4.1. KMedoid

El algoritmo KMedoid es un algoritmo estocástico que busca encontrar los mejores centros de cada agrupación a partir de una selección aleatoria. Para la elaboración de este algoritmo se tomaron como referencia las implementaciones del algoritmo KMeans de Hartigan y Wong [17] y McQueen [25]. La diferencia con el algoritmo KMeans radica en que los centros son elementos de la muestra.

El procedimiento utiliza un arreglo de largo fijo k para almacenar los centros de cada grupo. Para identificar a qué grupo pertenece cada elemento se utiliza un arreglo de largo fijo N, donde en cada posición se indica el índice del centro del grupo al que corresponde. A continuación se describe la serie de pasos que sigue el algoritmo implementado.

Paso 1: Inicializar aleatoriamente de los k centros de los grupos.

Paso 2: Asignar los $N - k$ elementos a un grupo.

Paso 3: Para cada grupo, encontrar un nuevo centro tal que el valor de fitness para esa nueva solución sea máximo.

Paso 4: Repetir Paso 2 y Paso 3 hasta que los centros se mantengan incambiados.

Un ejemplo de la iteración para el primer centro se presenta en la Figura 4.1.

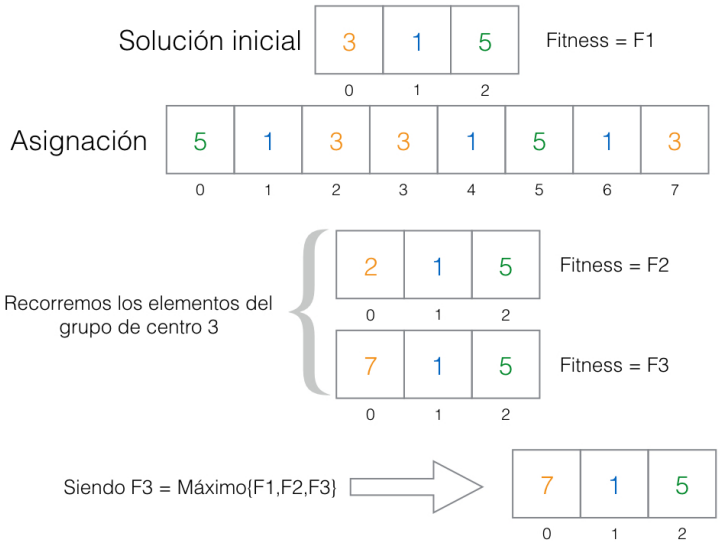


Figura 4.1: Ejemplo de aplicación del algoritmo KMedoid

4.4.2. Local Search

El algoritmo Local Search (búsqueda local) se basa en la propuesta de Sheng y Liu de utilizar el algoritmo KMedoid complementándolo con una búsqueda exhaustiva sobre cada grupo. Local Search es un algoritmo estocástico, que parte de una selección de centros aleatoria. El algoritmo Local Search comienza por establecer p "vecinos más cercanos" a cada centro. Es sobre este subconjunto que se itera para conseguir el nuevo centro que maximice la similitud. El algoritmo finaliza cuando todos los centros se mantienen incambiados de una iteración a la siguiente. La serie de pasos que sigue el algoritmo Local Search se describe a continuación.

Paso 1: Establecer la cantidad de vecinos más similares p.

Paso 2: Inicializar aleatoriamente los k centros de los grupos.

Paso 3: Asignar los $N - k$ elementos a un grupo.

Paso 4: Actualizar los centros y, para cada grupo:

a) Para los p elementos del grupo más cercanos al centro, encontrar un nuevo centro tal que el valor de la similitud para ese nuevo centro sea máximo.

b) Repetir *a*) hasta que el centro se mantenga incambiado.

Paso 4: Repetir Paso 3 y Paso 4 hasta que los centros se mantengan incambiados.

4.4.3. Algoritmo Ávido

El algoritmo ávido (greedy) construye la solución de manera iterativa, tomando en cada paso decisiones localmente óptimas. El procedimiento que sigue es secuencial y no determinista. El algoritmo comienza seleccionando el primer centro de forma aleatoria y, en cada paso posterior, busca el elemento de menor similitud con la solución en construcción. Este elemento se agrega a la solución como un nuevo centro. Este procedimiento continúa hasta que se completan los k centros. Se muestra el pseudocódigo en el Algoritmo 2.

Algoritmo 2 Greedy

Salida: solución

```
1: solución[0]=Random(0, N)
2: para i=1 hasta i=k hacer
3:    solución[i] = menos_similar(solución)
4: fin para
```

4.4.4. Algoritmo linkage

La herramienta Matlab provee la función linkage. Esta función toma como parámetros de entrada una matriz de distancia y un enumerado que representa el algoritmo para computar la distancia entre los clusters y retorna un árbol de agrupamiento jerárquico [3].

4.4.5. Híbrido

El algoritmo Híbrido (Hybrid) se basa en la propuesta de Sheng y Liu de utilizar Local Search dentro de un algoritmo evolutivo. Sheng y Liu proponen la siguiente secuencia de pasos que debe seguir el algoritmo evolutivo.

Paso 1: Inicializar aleatoriamente los k centros.

Paso 2: Repetir los pasos a) a d) hasta alcanzar el criterio de parada

- a) Seleccionar utilizando un torneo de tamaño 2.
- b) Con cierta probabilidad, realizar el cruzamiento Mix Subset Recombination.
- c) Con cierta probabilidad, realizar la mutación Bit Flip.
- d) Con cierta probabilidad, aplicar el operador Local Search.

A continuación, se describen los detalles correspondientes a la implementación del algoritmo evolutivo híbrido, en el que se utilizó una representación binaria.

Operadores Genéticos

Mix Subset Recombination Crossover. El procedimiento Mix Subset Recombination puede describirse con la secuencia de pasos que se detallan a continuación.

Paso 1: Concatenar ambos padres para obtener Xmix.

Paso 2: Reordenar aleatoriamente los elementos de Xmix.

Paso 3: Con cierta probabilidad, realizar la mutación Bit Flip.

Paso 4: Obtener ambos hijos de la división de Xmix en dos partes de igual tamaño.

Bit Flip Mutation. El procedimiento Bit Flip Mutation realiza la inversión de un bit.

Función correctiva de soluciones no factibles

Dado que la cantidad de grupos es un parámetro de entrada del algoritmo evolutivo híbrido pueden surgir inconsistencias luego de aplicar los operadores genéticos, ya que éstos no garantizan mantener esta cantidad constante. Es por ello que es necesario contar con un procedimiento de corrección de soluciones no factibles. La función correctiva implementada consiste en agregar/eliminar centros de forma aleatoria si la cantidad de centros es menor/mayor a la establecida como parámetro de entrada.

Criterio de parada

Sheng y Liu proponen dos alternativas como criterio de parada. El primer criterio de parada propuesto se cumple cuando el fitness de la mejor solución hallada permanece incambiado durante n generaciones, siendo n un natural positivo. La segunda alternativa, que es la elegida en la implementación utilizada en el proyecto, corresponde a ejecutar durante un cierto número fijo de generaciones.

Función de fitness

Sheng y Liu proponen para el cálculo del fitness utilizar el valor $1/SED$, siendo SED (Sum of Euclidean Distances) la fórmula descrita en la Ecuación 4.2.

$$SED = \sum_{i=1}^{i=n} \sum_{i=j, x_i \in C_j}^{k} d(x_i, m_j) \tag{4.2}$$

donde m_j representa el centro del cluster C_j

Capítulo 5

Implementación

Este capítulo describe los algoritmos implementados para la resolución del problema de clustering. En la Sección 5.1 se describe la biblioteca de desarrollo utilizada. La siguiente sección presenta la implementación del algoritmo evolutivo monoobjetivo y la Sección 5.3 describe las implementaciones del algoritmo evolutivo en su variante multiobjetivo.

5.1. Bibliotecas de desarrollo ECJ

ECJ es un framework de código libre de computación evolutiva implementado en Java, desarrollado en la Universidad de George Mason, Estados Unidos [24]. ECJ fue diseñado para necesidades experimentales complejas y de gran tamaño, haciendo especial énfasis en la programación genética. Algunas de las herramientas que ofrece ECJ son el soporte a algoritmos evolutivos multiobjetivo, una implementación del conocido generador de números aleatorios "Mersenne Twister"[27] y una estructuración en clases que hace posible incorporar código por parte del usuario de forma sencilla. Además, ECJ brinda herramientas de reporte estadístico, soporte para algoritmos evolutivos y para paralelizar las ejecuciones, así como facilidades para pausar y reanudar ejecuciones. El framework también ofrece dos implementaciones de algoritmos multiobjetivos, NSGA-II y SPEA2.

El framework ECJ se utilizó para implementar todos los algoritmos evolutivos referidos en este proyecto. Particularmente, para la solución del problema multiobjetivo se utilizó la implementación del algoritmo NSGA-II. Para la elección de los números aleatorios utilizados en la resolución de los problemas se utilizó el generador de números aleatorios "Mersenne Twister".

5.2. Algoritmo evolutivo versión monoobjetivo

En las Secciones 5.2.2 y 5.2.3 se describen los detalles correspondientes a la implementación del algoritmo evolutivo monoobjetivo con representación binaria y representación entera, respectivamente.

5.2.1. Procedimiento para el cálculo de fitness

Como indica Hruschka *et al.* [14] un posible procedimiento para el cálculo de fitness, orientado al problema del clustering, es la sugerida por Lucasius. Esta función consta de minimizar la distancia entre los centros. Se presenta un ejemplo de la función objetivo, siendo X el conjunto de los objetos $\{x_1, x_2, ...x_N\}$ y los centros $\{m_1, m_2...m_k\}$, la función de fitness se presenta en 5.1.

$$F = \sum_{i=1}^{N} d(x_i, m)$$
$$\text{donde } d(xi, m) = \min_{j \in 1,..k} d(x_i, m_j) \tag{5.1}$$

Debido a que el caso de estudio utiliza la medida de similitud entre los objetos de la muestra en lugar de utilizar la distancia entre ellos, la función de fitness busca maximizar la similitud de los elementos al centro. El pseudocódigo del cálculo de la función de fitness se encuentra en el Algoritmo 3.

Algoritmo 3 Fitness Lucasius

Entrada: individuo
Salida: fitness

```
1:  fitness = 0
2:  para todo elemento de la muestra hacer
3:      max_similitud = 0
4:      para todo centro del individuo hacer
5:          s = similitud(centro, elemento)
6:          si max_similitud < s entonces
7:              max_similitud = s
8:          fin si
9:      fin para
10:     fitness = fitness + max_similitud
11: fin para
12: devolver fitness
```

El Algoritmo 3 consta de dos bucles anidados. En la línea 2 se define el bucle exterior del procedimiento, donde se recorren todos los elementos de la muestra. En el bucle interior, línea 4, se recorre la solución y, para cada centro, se calcula la similitud de ese centro con el elemento de la muestra seleccionado en el bucle exterior. El resultado del bucle interior es el valor máximo de las similitudes calculadas. El bucle exterior realiza la suma de todas las similitudes para calcular el valor de fitness que retorna como resultado.

Se puede ver un ejemplo en la Figura 5.1, para una solución con cinco elementos donde los elementos 2 y 3 son los centros, recuadrados por los colores anaranjado y verde respectivamente. El elemento 0 pertenece al grupo de centro 2, ya que la similitud entre 0 y 2 es mayor que la de 0 y 3. De la misma manera, se hallan los centros para los elementos 1 y 4. Finalmente, los grupos encontrados son {0,1,2} y {3,4}, los elementos de cada cluster son identificados en la Figura 5.1 con el color del centro al que pertenecen. El cálculo del fitness para la ejemplo anterior es 72.

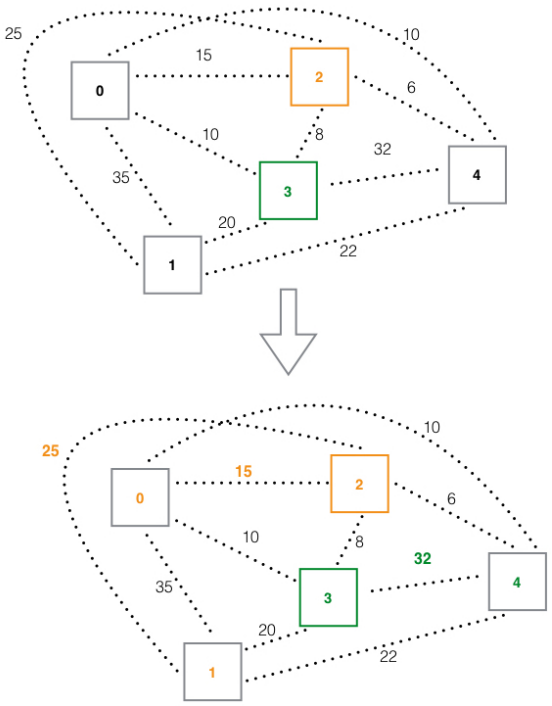


Figura 5.1: Procedimiento para seleccionar centros y calcular el fitness de una solución

5.2.2. Operadores para representación binaria

A continuación, se detallan los operadores de cruzamiento, operadores de mutación y la función de fitness correspondientes al algoritmo monoobjetivo con representación binaria. Además, se describe el método utilizado para evitar individuos no factibles.

Operadores de cruzamiento

Se implementan dos operadores de cruzamiento, el cruzamiento de un punto (Single Point Crossover, SPX) y el cruzamiento de dos puntos (Two Point Crossover, 2PX).

Cruzamiento de un punto. El cruzamiento de un punto se describe mediante el diagrama en el Algoritmo 4 y la Figura 5.2. Se selecciona una posición en uno de los individuos a cruzar y se intercambian los valores anteriores a este punto entre ambos individuos. La posición se determina de manera aleatoria y debe pertenecer al rango entre 0 y N, siendo N la cantidad de elementos a agrupar.

Algoritmo 4 Cruzamiento de un punto

Entrada: individuo_1, individuo_2
Salida: individuo_1, individuo_2

```
punto = random(0, N)
para indice = 0 hasta indice = punto hacer
    temporal = individuo_1[indice]
    individuo_1[indice] = individuo_2[indice]
    individuo_2[indice] = temporal
fin para
```

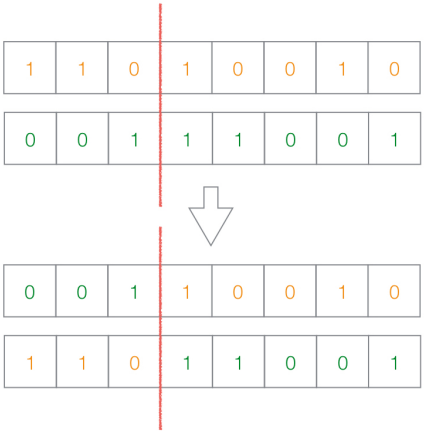


Figura 5.2: Cruzamiento de un punto en representación binaria

Cruzamiento de dos puntos. En el cruzamiento de dos puntos se seleccionan dos puntos de corte y se intercambian los genes de los individuos entre los puntos de corte seleccionados. Los puntos de corte son seleccionados de forma aleatoria. El algoritmo del cruzamiento de dos puntos puede verse ejemplificado en la Figura 5.3 y su funcionamiento se describe en el Algoritmo 5.

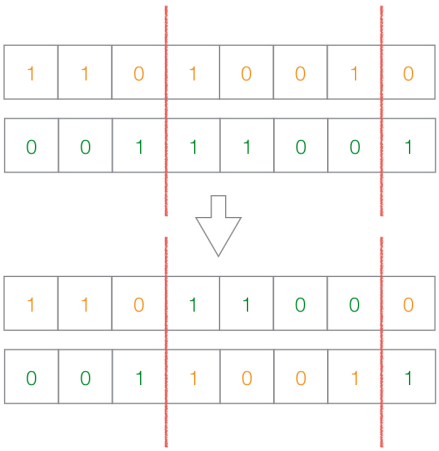


Figura 5.3: Cruzamiento de dos puntos en representación binaria

Algoritmo 5 Cruzamiento de dos puntos

Entrada: individuo_1, individuo_2
Salida: individuo_1, individuo_2

```
1: punto_1 = random(0, N)
2: punto_2 = random(0, N), punto_2 != punto_1
3: para indice = punto_1 hasta indice = punto_2 hacer
4:     temporal = individuo_1[indice]
5:     individuo_1[indice] = individuo_2[indice]
6:     individuo_2[indice] = temporal
7: fin para
```

Operadores de mutación

Se implementan tres operadores de mutación, la mutación de un gen (Bit Flip Mutation), la mutación donde se agregan centros a la solución (Add Mutation) y la mutación donde se eliminan centros a la solución (Delete Mutation).

Bit Flip Mutation. En este operador de mutación se recorre la solución y, para cada elemento, se lo cambia por su opuesto, con una probabilidad preestablecida. El algoritmo Bit Flip Mutation puede verse ejemplificado en la Figura 5.4 y su funcionamiento se describe en el Algoritmo 6.

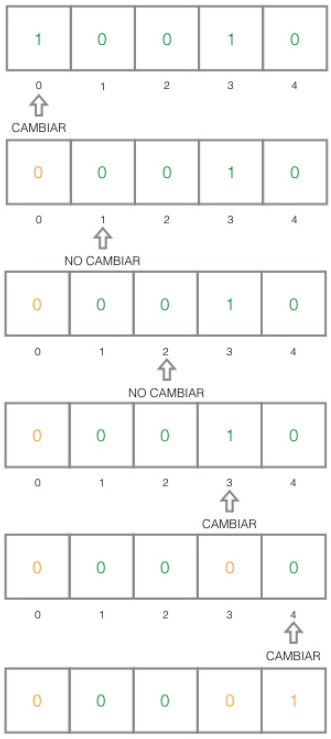


Figura 5.4: Procedimiento para la mutación de un gen

Algoritmo 6 Bit Flip Mutation

Entrada: individuo
Salida: individuo

```
1: para indice = 0 hasta indice = N-1 hacer
2:     si random(0, 1) < probabilidad entonces
3:         individuo[indice] = flip(individuo[indice])
4:     fin si
5: fin para
```

Add Mutation. Este operador de mutación recorre el individuo y, para cada elemento, se lo cambia a 1 con cierta probabilidad preestablecida. La mutación de adición de centros se presenta en el Algoritmo 7.

Delete Mutation. Este operador de mutación es análogo al operador Add Mutation con la diferencia que en lugar de cambiar los valores por 1 lo hace por 0. El pseudocódigo

Algoritmo 7 Add Mutation

Entrada: individuo
Salida: individuo

```
1: para indice = 0 hasta indice = individuo.largo hacer
2:     si Random(0, 1) < probabilidad entonces
3:         individuo[indice] = 1
4:     fin si
5: fin para
```

del algoritmo de mutación de eliminación de centros se presenta en el Algoritmo 8.

Algoritmo 8 Delete Mutation

Entrada: individuo
Salida: individuo

```
1: para indice = 0 hasta indice = individuo.largo hacer
2:     si Random(0, 1) < probabilidad entonces
3:         individuo[indice] = 0
4:     fin si
5: fin para
```

Ninguna de los tres operadores de mutación aseguran que la cantidad de centros permanezca constante, por lo tanto al finalizar la ejecución es necesario utilizar un procedimiento para evitar individuos no factibles.

Función correctiva de soluciones no factibles

Debido a que la cantidad de grupos es un parámetro de entrada del AE, pueden generarse inconsistencias en las soluciones luego de aplicado el operador de cruzamiento y/o la mutación, ya que éstos no toman en consideración el hecho de mantener la cantidad de grupos constante. Para resolver este inconveniente se implementa un método para evitar individuos no factibles. El método de corrección propuesto se presenta en el Algoritmo 9, donde k es la cantidad de centros deseada. El método de corrección se basa en agregar centros de manera aleatoria si la cantidad de centros es menor a la que se establece al comienzo y, en caso contrario, elimina los centros restantes seleccionándolos también de manera aleatoria. El procedimiento para corregir las soluciones no factibles se ejemplifica en la Figura 5.5.

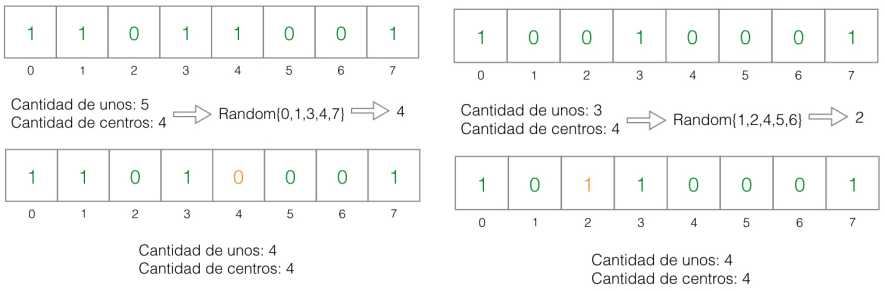


Figura 5.5: Función correctiva de soluciones no factibles en representación binaria

Algoritmo 9 Función correctiva de soluciones no factibles

Entrada: individuo
Salida: individuo

```
1: cantidad_centros_individuo = cantidad_unos_individuo()
2: mientras cantidad_centros_individuo < K hacer
3:    nuevo_centro = Random(0, N),
4:        donde centros_inidividuo.no_contiene(nuevo_centro)
5:    centros_individuo.agregar(nuevo_centro)
6:    cantidad_centros_individuo++
7: fin mientras
8: mientras cantidad_centros_individuo > K hacer
9:    centro_a_eliminar = Random(0, centros_inidividuo.largo)
10:   centros_individuo.eliminar(centro_a_eliminar)
11:   cantidad_centros_individuo - -
12: fin mientras
```

5.2.3. Operadores para representación entera

A continuación, se detallan los operadores de cruzamiento y los operadores de mutación para el AE monoobjetivo con representación entera. Además, se describe el método utilizado para evitar individuos no factibles en esta representación.

Operadores de cruzamiento

Se implementaron tres operadores de cruzamiento para los individuos con representación entera. Estos son: *Cruzamiento de un Punto*, cruzamiento *Generalized Cut and Splice* (GenC&S) y *Cruzamiento Híbrido*.

Cruzamiento de un punto. El cruzamiento de un punto para la representación entera tiene los mismos fundamentos que el anteriormente comentado para la representación binaria. En este operador de cruzamiento se selecciona un punto de corte y se intercambian los elementos previos a este punto entre los dos individuos. Un ejemplo para individuos de largo 5 se puede observar en la Figura 5.6 y el pseudocódigo de la implementación de este operador de cruzamiento se muestra en el Algoritmo 10.

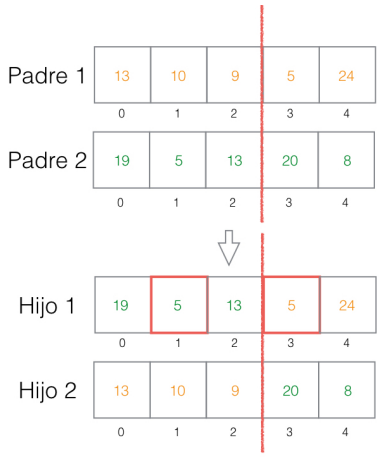


Figura 5.6: Procedimiento para el cruzamiento de un punto con representación entera

Algoritmo 10 Cruzamiento de un punto con representación entera

Entrada: individuo_1, individuo_2
Salida: individuo_1, individuo_2

```
1: punto = Random(0, K)
2: para indice = 0 hasta indice = punto hacer
3:    temporal = individuo_1[indice]
4:    individuo_1[indice] = individuo_2[indice]
5:    individuo_2[indice] = temporal
6: fin para
```

Cruzamiento GenC&S. El cruzamiento GenC&S se basa en la propuesta de Pereira *et al.* [32] de extender al cruzamiento Cut and Splice original de Deaven y Ho [8]. Deaven y Ho implementaron el operador de cruzamiento C&S con el objetivo de preservar las características sensibles al problema de clustering de los padres. El procedimiento GenC&S se describe en el Algoritmo 11.

Algoritmo 11 Cruzamiento GenC&S para la representación entera

Entrada: padre_1, padre_2
Salida: hijo_1, hijo_2

```
1: punto = padre_1[Random(0,K)]
2: s = Random(0,K)
3: hijo_1.agregar(punto)
4: lp_1=ordenarAscendente(padre_1,punto)
5: lp_2=ordenarAscendente(padre_2,punto)
6: para todo i = 1 hasta i < s hacer
7:    hijo_1.agregar(lp_1[i])
8: fin para
9: para todo j = s hasta j < K hacer
10:   si similitud(lp_2[i],hijo_1)<epsilon entonces
11:      hijo_1.agregar(lp_2[j])
12:   fin si
13: fin para
14: mientras hijo_1.largo < K hacer
15:   nuevo_centro = Random(0,N),similitud(nuevo_centro,hijo_1)<epsilon
16:   hijo_1.agregar(nuevo_centro)
17: fin mientras
```

GenC&S comienza seleccionando aleatoriamente un elemento denominado CP o *cutting point* del padre 1 y un número S entre 0 y K, línea 1 y línea 2 del Algoritmo 11 respectivamente. Luego, se crea una lista LP1 con los elementos del padre 1 ordenados de forma ascendente respecto a la similitud con CP, línea 4. Posteriormente, se realiza el mismo procedimiento generando LP2 con los elementos del padre 2. Luego, se copian los S primeros elementos de LP1 al primer hijo, esto se observa en el bucle de la línea 6. Finalmente, en la línea 9 del Algoritmo 11 se completan los K-S elementos restantes del hijo 1 con los valores de LP2, cuya similitud con CP sea menor a un épsilon. Si el largo del hijo 1 es inferior a K, se completa la solución con valores aleatorios entre 0 y N, esto se observa en la línea 14. El algoritmo de cruzamiento GenC&S puede verse ejemplificado

en la Figura 5.7.

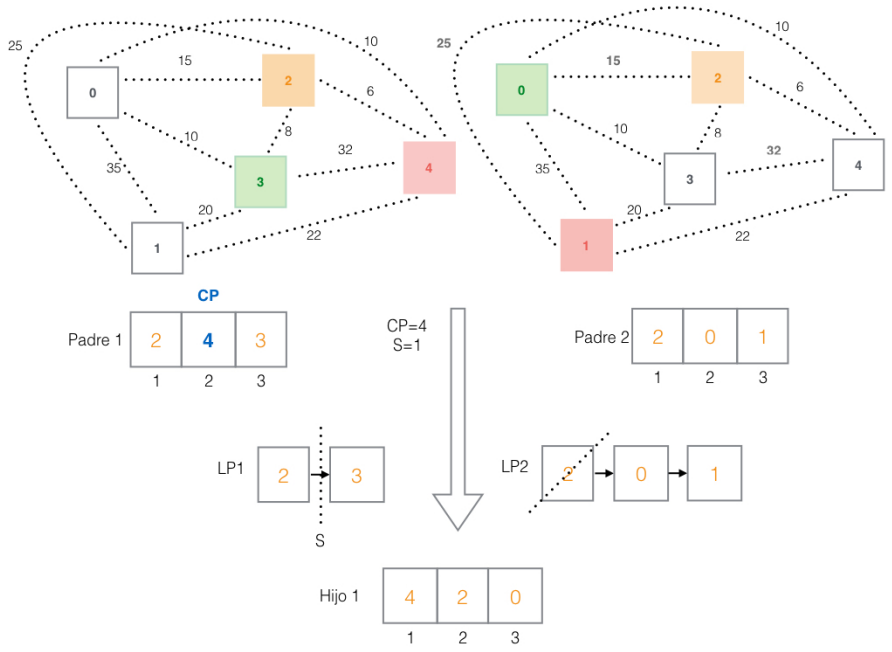


Figura 5.7: Procedimiento para el cruzamiento GenC&S

Cruzamiento híbrido. Se implementó un operador de cruzamiento híbrido entre el cruzamiento de un punto y el cruzamiento GenC&S. La motivación detrás de este operador de cruzamiento es combinar los operadores de cruzamiento descritos anteriormente, y evaluar los resultados experimentales obtenidos. El pseudocódigo de este procedimiento puede verse en el Algoritmo 12.

Algoritmo 12 Cruzamiento híbrido para la representación entera

Entrada: padre_1, padre_2
Salida: hijo_1, hijo_2

```
1:  punto = Random(0,K)
2:  para todo i = 1 hasta i = punto hacer
3:      hijo_1[i] = individuo_1[i]
4:  fin para
5:  para todo j = (punto + 1) hasta j = K hacer
6:      si similitud(individuo_2[j],hijo_1)<epsilon entonces
7:          hijo_1.agregar(individuo_2[j])
8:      fin si
9:  fin para
10: mientras hijo_1.largo < K hacer
11:     nuevo_centro = Random(0,N),similitud(nuevo_centro,hijo_1)<epsilon
12:     hijo_1.agregar(nuevo_centro)
13: fin mientras
```

En este operador de cruzamiento se determina de forma aleatoria un punto de corte, línea 1 del Algoritmo 12. Luego, en el bucle de la línea 2, se copian al hijo 1 los elementos

anteriores al punto de corte inclusive. Hasta este momento el procedimiento es análogo al cruzamiento de un punto. En el siguiente bucle, línea 5, el hijo 1 se completa con elementos del padre 2 que se encuentren luego del punto de corte y cuya similitud con los elementos del hijo 1 es menor que un cierto ε. Finalmente en la línea 10, si el largo del hijo 1 es menor que k, la tupla se completa con valores seleccionados aleatoriamente que no se encuentren en la solución y cuya distancia al hijo 1 sea inferior a ε.

Operadores de mutación

Se implementaron dos operadores de mutación: la mutación de un gen y la mutación de un gen adaptada.

Mutación de un gen. En la mutación de un gen para la representación entera se recorre la solución y según la probabilidad de mutación se cambia (o no) el elemento por otro que no esté incluido en la solución. El nuevo valor para el gen se selecciona aleatoriamente, de acuerdo a una distribución uniforme entre 0 y N. El pseudocódigo de este operador de mutación se presenta en el Algoritmo 13.

Algoritmo 13 Mutación de un gen para la representación entera

Entrada: individuo
Salida: individuo

```
1: para indice = 0 hasta indice = K-1 hacer
2:    si Random(0, 1) < probabilidad entonces
3:       nuevo_centro = Random(0,N), donde individuo.no_contiene(nuevo_centro)
4:       individuo[indice] = nuevo_centro
5:    fin si
6: fin para
```

Mutación de un gen adaptada. La mutación de un gen adaptada comienza recorriendo la solución y, para cada elemento, evalúa la probabilidad de mutarlo. En caso positivo, se busca en toda la solución el elemento más similar al que se desea mutar (elemento j). Ese valor de similitud se toma como parámetro γ. Luego se modifica el centro por otro elemento cuya similitud con el elemento j sea menor que γ. El pseudocódigo de este operador de mutación se presenta en el Algoritmo 14.

Algoritmo 14 Mutación de un gen adaptada

Entrada: individuo
Salida: individuo

```
1: para i = 0 hasta i = K-1 hacer
2:    si Random(0, 1) < probabilidad entonces
3:       mas_similar=mas_similar(individuo,individuo[i])
4:       nuevo_centro = Random(0,N), donde individuo.no_contiene(nuevo_centro) y si-
         militud(mas_similar,nuevo_centro)<similitud(mas_similar,individuo[i])
5:       individuo[i] = nuevo_centro
6:    fin si
7: fin para
```

Función correctiva de soluciones no factibles

Como la cantidad de grupos en la solución final es un parámetro de entrada para el AE, ninguno de los operadores evolutivos presentados anteriormente garantiza un resultado que mantenga constante la cantidad de grupos. Un ejemplo se puede apreciar en la Figura 5.6 presentada anteriormente. Para evitar que se generen individuos no factibles se diseñó un procedimiento de corrección donde se reemplaza al elemento repetido por un valor contenido en los padres no utilizado en el individuo hijo. Este procedimiento se visualiza en la Figura 5.8.

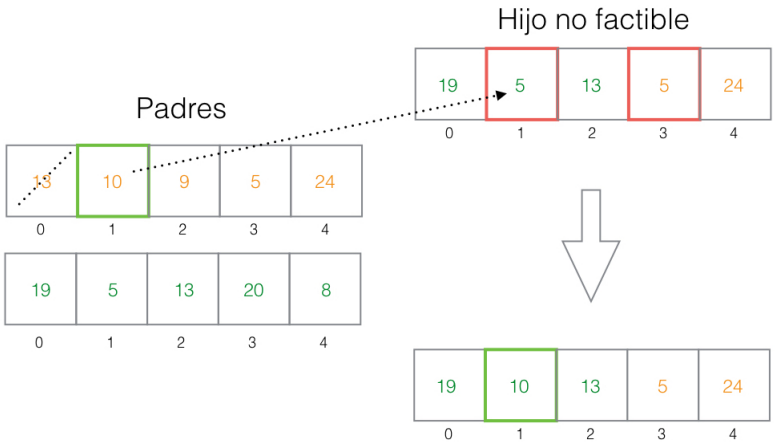


Figura 5.8: Procedimiento para evitar individuos no factibles en el AE que utiliza representación entera

5.3. Algoritmo evolutivo versión multiobjetivo

En esta sección se describen los detalles correspondientes a la implementación del algoritmo evolutivo multiobjetivo con representación binaria.

Representación binaria

Se utiliza una representación binaria para la resolución de la variante multiobjetivo del problema de clustering, debido a que la representación binaria alcanzó mejores resultados en la evaluación experimental para los AE en su variante monoobjetivo tal como se muestra en la sección 6.3.2. Los operadores evolutivos utilizados en esta representación son los mismos que se presentaron anteriormente para la variante monoobjetivo del AE con representación binaria.

A continuación, se describe el procedimiento de corrección de soluciones no factibles y las funciones objetivo.

Función correctiva de soluciones no factibles

Existe una única configuración que se considera no factible para el AE multiobjetivo. Ésta se obtiene cuando todos los elementos de la solución son 0. En este caso no hay ningún centro designado y, por lo tanto, ninguna agrupación correspondiente. Para corregir este caso se define una función de corrección cuya implementación se presenta en el Algoritmo 15.

Algoritmo 15 Función correctiva para el algoritmo evolutivo multiobjetivo

Entrada: individuo
Salida: individuo

```
1: si cantidad_centros(individuo)=0 entonces
2:    nuevo_centro = Random(0,N)
3:    individuo[nuevo_centro] = 1
4: fin si
```

Objetivos

Para la variante multiobjetivo del problema de clustering se definen dos objetivos: maximizar el valor agregado de similitud (como en la versión monoobjetivo) y minimizar la cantidad de centros de la solución. El pseudocódigo que describe el cálculo de las funciones objetivo para el AE multiobjetivo se presenta en el Algoritmo 16.

Algoritmo 16 Cálculo de las funciones objetivo para el algoritmo evolutivo multiobjetivo

Entrada: individuo
Salida: objetivo1, objetivo2

```
1: cant_centros = 0
2: suma_similitud = 0
3: para i = 0 hasta i = N hacer
4:    distancia = 0
5:    si individuo[indice] = 0 entonces
6:       para j = 0 hasta j = N hacer
7:          si individuo[j] = 1 y matriz[i][j] > similitud entonces
8:             similitud = matriz[i][j]
9:          fin si
10:      fin para
11:   si no
12:      cant_centros++
13:   fin si
14:   suma_similitud += similitud
15: fin para
16: objetivo1 = suma_similitud
17: objetivo2 = cant_centros
```

Capítulo 6

Evaluación experimental

El presente capítulo detalla la evaluación experimental de los AE implementados para resolver el problema de clustering. La Sección 6.1 describe el proceso de obtención de instancias de prueba del problema utilizadas para la evaluación experimental. La Sección 6.2 describe la metodología utilizada durante la evaluación experimental de los AE implementados. La Sección 6.3 presenta las pruebas realizadas y los resultados numéricos obtenidos con los algoritmos en la variante monoobjetivo para el problema de clustering. La Sección 6.4 presenta los resultados experimentales correspondientes al algoritmo que resuelve la variante multiobjetivo del problema de clustering.

6.1. Obtención de instancias de prueba

Con el fin de evaluar la calidad de los algoritmos implementados se utilizaron diversas instancias del problema de clustering, provenientes de diversas áreas del conocimiento. Se cuenta con una matriz que contiene los valores de similitud entre genes humanos preseleccionados por colegas de la universidad de Luxemburgo, con el fin de agrupar eficientemente la información relevante a la enfermedad de Parkinson. La similitud utilizada en la matriz se basa en el paquete del lenguaje R denominado GoSemSim [38]. Esta matriz (representada como la instancia #9) cuenta con 801 elementos a agrupar.

La instancia #1 consta de 46 elementos formada por cuencas aforadas de Uruguay. El objetivo de agrupar estas cuencas es lograr una clasificación del aporte de las estaciones hidrométricas que se tienen en DINAGUA[1].

Las instancias #2 a #8 y #10 a #12 se obtuvieron del repositorio de datos para el problema de clustering *Knowledge Extraction based on Evolutionary Learning* [2]. Estas instancias tienen 80, 101, 160, 297, 336, 306, 358, 768, 625 y 846 elementos a agrupar, respectivamente. Para estas instancias fue necesario realizar una transformación de las matrices para llevarlas al tipo *one mode* descrito en la Sección 2.1.

Las instancias de prueba fueron agrupadas en tres familias según la cantidad de elementos a agrupar. Las instancias chicas son #1, #2, #3 y #4. Las instancias #5, #6, #7 y #8 son instancias medianas y las restantes son las denominadas instancias grandes.

En el mes de marzo, se obtuvo una segunda matriz por parte de los colegas de Luxemburgo. Esta matriz corresponde a la instancia #13 y contiene 3056 elementos a agrupar. Esta conjunto de datos se utiliza en la evaluación multiobjetivo del problema de clustering.

[1] Pasantía de Soledad Bonner a cargo de Ing. Civil Rodolfo Chao e Ing. Civil Magdalena Crisci (IMFIA)

6.2. Metodología

En esta sección se presentan los lineamientos generales seguidos al momento de realizar la evaluación experimental de los algoritmos implementados.

Entorno de ejecución. La evaluación experimental de los AE implementados fue realizada utilizando una computadora personal con un procesador Intel Core i5 2.7 GHz con 8GB de memoria RAM disponible. Las ejecuciones de todas las instancias de prueba para cada algoritmo fueron realizadas sobre la misma infraestructura.

Ejecuciones independientes. Debido a la cualidad estocástica de los algoritmos evolutivos, distintas ejecuciones sobre una misma instancia de prueba pueden alcanzar diferentes resultados. Con el fin de obtener resultados estadísticamente significativos, se realizaron 30 ejecuciones independientes de cada algoritmo sobre cada instancia de prueba. Las ejecuciones independientes se logran inicializando el generador de números pseudoaleatorios del AE con una semilla diferente en cada ejecución. Como consecuencia, al reportar los resultados obtenidos por un algoritmo en una determinada instancia de prueba, se indica el mejor valor y el valor promedio de los valores alcanzados en las 30 ejecuciones independientes realizadas.

Comparación de resultados. La comparación de resultados entre los AE implementados abarca exclusivamente la calidad de las soluciones, debido a que la eficiencia computacional no es una restricción del problema. La calidad de las soluciones se define de acuerdo a los valores alcanzados en las funciones objetivo y, para la variante multiobjetivo del problema, a través de métricas específicas para MOP.

Para comparar los resultados obtenidos por dos algoritmos se utilizan tests estadísticos sobre las distribuciones de resultados obtenidos en las ejecuciones independientes de cada instancia. Inicialmente, se aplica el test de Kolmogorov-Smirnov sobre cada muestra a comparar, para establecer si dicha muestra sigue o no una distribución normal. El test de Kolmogorov-Smirnov plantea como hipótesis nula que la muestra estudiada proviene de una distribución normal. Si el p–valor obtenido es menor al nivel de confianza, se puede rechazar la hipótesis nula y concluir que la muestra estudiada no sigue una distribución normal. A lo largo del trabajo se utiliza un nivel de confianza del 95 % ($\alpha = 0,05$) para el test de Kolmogorov-Smirnov.

En caso que el test de Kolmogorov-Smirnov permita concluir que los resultados obtenidos no siguen una distribución normal, se debe utilizar un test no paramétrico para comparar los resultados de dos algoritmos. Con este propósito, se utiliza el test no paramétrico de Kruskal–Wallis [23], ya que no asume normalidad en las muestras a comparar. El test de Kruskal–Wallis plantea como hipótesis nula que las muestras que se comparan provienen de la misma distribución. Un p–valor menor al nivel de confianza permite rechazar la hipótesis nula y concluir que una muestra domina a la otra. En este trabajo se utiliza un nivel de confianza del 95 % ($\alpha = 0,05$) para evaluar los resultados del test de Kruskal–Wallis. Dado que el test no ofrece información acerca de cuál es la muestra que domina, se utiliza el valor promedio alcanzado en las 30 ejecuciones independientes para tomar la decisión.

6.3. Variante monoobjetivo del problema de clustering

Esta sección detalla el análisis experimental de los algoritmos evolutivos que resuelven el problema de clustering en su formulación monoobjetivo.

6.3.1. Parámetros

A continuación se describen los parámetros utilizados en todas las implementaciones de los algoritmos evolutivos en su versión monoobjetivo.

En la Sección 5.2 se presentaron dos tipos de representación para la variante monoobjetivo de AE, representación binaria y representación entera. Para cada tipo de representación se implementaron seis algoritmos, producto de combinaciones entre tres operadores de mutación y dos operadores de cruzamiento. Para estos algoritmos se utilizaron los parámetros que se describen a continuación.

- Probabilidad de cruzamiento $p_C = 0,75$.
- Probabilidad de mutación $p_M = 0,01$.
- Selección por torneo de parámetros (2,1), donde se sortean 2 individuos al azar de la población y el de mayor fitness es seleccionado.
- Criterio de parada: alcanzar la cantidad de generaciones $gen = 10000$.
- Inicialización aleatoria.
- Tamaño de la población $pob = 100$.

Algoritmo KMedoid. En la Sección 4.4.1 se presentó al algoritmo KMedoid. A continuación se describen los parámetros relevantes a la implementación del algoritmo.

- Inicialización aleatoria.
- Criterio de parada: que los centros se mantengan incambiados durante una iteración.

Algoritmo Local Search. El algoritmo Local Search fue presentado en la Sección 4.4.2. Los parámetros recomendados por los autores se enumeran a continuación.

- Cantidad de vecinos más cercanos acorde a la cantidad de elementos a agrupar y la cantidad de grupos a formar.
- Criterio de parada: que los centros se mantengan incambiados durante una iteración.

Algoritmo Ávido. El único parámetro relevante a la implementación del algoritmo Ávido es la inicialización, que se realizó de forma aleatoria.

Algoritmo Híbrido. En la Sección 4.4.5 se presentó el algoritmo Híbrido. Los autores recomiendan utilizar los siguientes parámetros para ese AE.

- Probabilidad de cruzamiento $p_C = 0,95$.
- Probabilidad de mutación $p_M = 0,02$.
- Probabilidad de aplicar el operador de cruzamiento *Mix Subset Recombination* $p_{mix} = 0,05$.
- Probabilidad de aplicar el algoritmo Local Search $p_{ls} = 0,2$.

- Cantidad de vecinos más cercanos acorde a la cantidad de elementos a agrupar y la cantidad de grupos a formar.
- Selección por torneo de parámetros (2,1).
- Criterio de parada: alcanzar la cantidad de generaciones $gen = 10000$.
- Inicialización aleatoria.
- Tamaño de la población $pob = 30$.

6.3.2. Resultados numéricos

A continuación, se presentan los resultados numéricos del análisis experimental de los AE, para la formulación monoobjetivo del problema y los algoritmos descritos en la Sección 4.4. Las instancias utilizadas fueron generadas siguiendo la metodología descrita en la Sección 6.1. Para cada instancia se realizaron 30 ejecuciones independientes.

Comparativa de operadores evolutivos utilizando la representación entera

En la Sección 5.2.3 se presentaron seis algoritmos evolutivos utilizando la representación entera, divididos en dos operadores de mutación y tres operadores de cruzamiento. Las combinaciones de los operadores se enumeran a continuación.

- *Oneoc-Oneoc.* Cruzamiento de un punto y mutación de un gen.
- *Oneoc-Oneadapt.* Cruzamiento de un punto y mutación de un gen adaptada.
- *Híbrido-Oneoc.* Cruzamiento híbrido y mutación de un gen.
- *Híbrido-Oneadapt.* Cruzamiento híbrido y mutación de un gen adaptada.
- *GenC&S-Oneoc.* Cruzamiento GenC&S y mutación de un gen.
- *GenC&S-Oneadapt.* Cruzamiento GenC&S y mutación de un gen adaptada.

Estos nombres serán utilizados cuando se haga referencia a los algoritmos evolutivos con la combinación de los operadores que indica su nombre.

En las Tablas 6.1, 6.2 y 6.3 se presentan los resultados alcanzados por los algoritmos evolutivos implementados con la representación entera para las instancias chicas, medianas y grandes, respectivamente. Se reporta el mejor valor de similitud alcanzado (máx(s)) y el promedio de los mejores valores de similitud (s) obtenidos en las 30 ejecuciones independientes. Se reporta también el p-valor resultante del test de Kolmogorov-Smirnov (p-valor K-S) sobre los valores de similitud obtenidos por cada algoritmo evolutivo con representación entera, de manera de contrastar normalidad en los resultados alcanzados.

Tabla 6.1: Tabla comparativa de los AE con representación entera para las instancias chicas

		Conjuntos de datos - instancias chicas			
Operadores	Métricas	#1	#2	#3	#4
Oneoc - Oneoc	máx(s)	18.657	1.963	12.459	16.502
	s	**18.657**	**1.963**	12.419	**16.428**
	p-valor K-S	2.20E-16	2.20E-16	0.031	0.219
Oneoc - Oneadapt	máx(s)	18.657	1.963	12.449	16.438
	s	18.061	1.864	12.208	16.238
	p-valor K-S	0.327	0.457	0.669	0.597
Híbrido - Oneoc	máx(s)	18.657	1.963	12.201	15.832
	s	18.617	**1.963**	11.991	15.518
	p-valor K-S	0.023	5.34E-08	0.772	0.962
Híbrido - Oneadapt	máx(s)	18.657	1.963	12.228	15.719
	s	18.612	**1.963**	12.066	15.523
	p-valor K-S	0.008	2.20E-16	0.998	0.986
GenC&S - Oneoc	máx(s)	18.657	1.963	12.459	16.502
	s	**18.657**	**1.963**	**12.444**	16.409
	p-valor K-S	2.20E-16	2.20E-16	0.005	0.828
GenC&S - Oneadapt	máx(s)	18.657	1.963	12.459	16.444
	s	18.493	1.898	12.34	16.325
	p-valor K-S	9.00E-05	0.340	0.880	0.197
	p-valor K-W	2.20E-16	2.20E-16	2.20E-16	2.20E-16

Tabla 6.2: Tabla comparativa de los AE con representación entera para las instancias medianas

		Conjuntos de datos - instancias medianas			
Operadores	Métricas	#5	#6	#7	#8
Oneoc - Oneoc	máx(s)	78.622	116.447	54.979	63.437
	s	**78.349**	116.18	**54.713**	63.274
	p-valor K-S	0.001	2.25E-07	0.000	0.008
Oneoc - Oneadapt	máx(s)	78.622	116.447	54.526	63.163
	s	77.817	114.745	51.782	62.145
	p-valor K-S	0.149	0.302	0.380	0.820
Híbrido - Oneoc	máx(s)	78.179	116.447	54.979	61.522
	s	77.794	115.764	54.518	60.618
	p-valor K-S	0.776	0.090	0.268	0.828
Híbrido - Oneadapt	máx(s)	78.047	116.447	54.979	61.936
	s	77.843	115.482	54.488	60.803
	p-valor K-S	0.352	0.001	0.424	0.960
GenC&S - Oneoc	máx(s)	78.622	116.447	54.979	63.437
	s	78.156	**116.391**	54.683	**63.298**
	p-valor K-S	0.000	8.16E-08	0.000	0.001
GenC&S - Oneadapt	máx(s)	78.622	116.447	54.979	63.432
	s	78.191	114.664	53.401	62.665
	p-valor K-S	0.001	0.228	0.715	0.528
	p-valor K-W	5.82E-16	2.27E-15	2.20E-16	2.20E-16

Tabla 6.3: Tabla comparativa de los AE con representación entera para las instancias grandes

		Conjuntos de datos - instancias grandes			
Operadores	Métricas	#9	#10	#11	#12
Oneoc - Oneoc	máx(s)	675.075	37.972	236.016	33.034
	s	673.568	37.773	**235.334**	**32.887**
	p-valor K-S	0.7707	0.577	0.854	0.459
Oneoc - Oneadapt	máx(s)	674.770	37.955	235.909	33.0191
	s	**673.627**	**37.776**	235.255	32.045
	p-valor K-S	0.534	0.476	0.657	0.275
Híbrido - Oneoc	máx(s)	642.534	36.142	224.639	31.44
	s	628.378	35.239	219.545	30.680
	p-valor K-S	0.717	0.735	0.477	0.835
Híbrido - Oneadapt	máx(s)	635.973	35.773	222.345	31.121
	s	625.772	35.093	218.635	30.553
	p-valor K-S	0.891	0.834	0.860	0.971
GenC&S - Oneoc	máx(s)	660.248	37.138	230.832	32.308
	s	656.556	36.819	229.390	32.056
	p-valor K-S	0.492	0.003	0.008	0.002
GenC&S - Oneadapt	máx(s)	660.684	37.163	230.985	32.330
	s	657.084	36.849	229.575	32.082
	p-valor K-S	0.906	0.002	0.005	0.01
	p-valor K-W	2.20E-16	2.20E-16	2.20E-16	2.20E-16

Si el p-valor del test de Kolmogorov-Smirnov es menor que 0.05, es posible rechazar la hipótesis nula de que los resultados producidos por los algoritmos siguen una distribución normal. Dado que existen algunos valores que permiten rechazar la hipótesis nula, se utilizó el test no paramétrico de Kruskal–Wallis para comparar los resultados obtenidos de los algoritmos evolutivos con distintas combinaciones de operadores. En la última fila de las tablas anteriores se reporta el p-valor del test de Kruskal–Wallis (p-valor K-W) sobre los valores de similitud obtenidos con cada algoritmo. Dado que en todas las instancias el p-valor K-W es menor que 0.05, podemos afirmar que uno de los algoritmos obtuvo mejores resultados que los otros, se destaca en negrita el que alcanzó mejor promedio de valores de similitud.

Se puede concluir que los algoritmos evolutivos con los operadores *Oneoc-Oneoc* y *GenC&S-Oneoc* alcanzan mejores resultados que los otros algoritmos con los que se realizó la comparación. En particular, el algoritmo *Oneoc-Oneoc* alcanzó los mejores resultados en siete instancias de prueba. Mientras que el algoritmo *GenC&S-Oneoc* alcanzó los mejores resultados en cinco instancias de prueba.

Comparativa de operadores evolutivos utilizando la representación binaria

En la Sección 5.2.2 se presentan seis algoritmos evolutivos utilizando la representación binaria, divididos en tres operadores de mutación y dos operadores de cruzamiento. Las combinaciones de los operadores se enumeran a continuación.

- *One-Flip.* Cruzamiento de un punto y mutación Bit Flip Mutation.
- *One-Add.* Cruzamiento de un punto y mutación Add Mutation.
- *One-Delete.* Cruzamiento de un punto y mutación Delete Mutation.

- *Two-Flip.* Cruzamiento de dos puntos y mutación Bit Flip Mutation.
- *Two-Add.* Cruzamiento de dos puntos y mutación Add Mutation.
- *Two-Delete.* Cruzamiento de dos puntos y mutación Delete Mutation.

Estos nombres serán utilizados cuando se haga referencia a los algoritmos evolutivos con la combinación de los operadores que indica su nombre.

En las Tablas 6.4, 6.5 y 6.6 se presentan los resultados alcanzados por los algoritmos evolutivos implementados con la representación binaria para las instancias chicas, medianas y grandes, respectivamente. Al igual que para la comparación de los algoritmos con representación entera, se presentan los mejores valores de similitud (máx(s)) y el promedio de los mejores valores de similitud (s) obtenidos en las 30 ejecuciones independientes. Se presenta además el p-valor resultante del test de Kolmogorov-Smirnov (p-valor K-S) sobre los valores de similitud obtenidos por cada algoritmo evolutivo con representación binaria, de manera de contrastar normalidad en los resultados alcanzados.

Tabla 6.4: Tabla comparativa de los AE con representación binaria para las instancias chicas

		Conjuntos de datos - instancias chicas			
Operadores	Métricas	#1	#2	#3	#4
One - Flip	máx(s)	18.657	1.963	12.376	16.112
	s	**18.657**	**1.963**	12.272	15.933
	p-valor K-S	2.20E-16	2.20E-16	0.8565	0.733
One - Add	máx(s)	18.657	1.963	12.389	16.183
	s	**18.087**	1.956	12.2	15.911
	p-valor K-S	0.004	0.000	0.922	0.867
One - Delete	máx(s)	18.657	1.963	12.459	16.502
	s	**18.657**	**1.963**	**12.459**	**16.5**
	p-valor K-S	2.20E-16	2.20E-16	2.20E-16	0.001
Two - Flip	máx(s)	18.657	1.963	12.358	16.325
	s	**18.657**	**1.963**	12.237	15.893
	p-valor K-S	2.20E-16	2.20E-16	0.216	0.434
Two - Add	máx(s)	18.657	1.963	12.338	16.107
	s	18.359	**1.963**	12.217	15.917
	p-valor K-S	0.002	2.20E-16	0.816	0.995
Two - Delete	máx(s)	18.657	1.963	12.459	16.502
	s	**18.657**	**1.963**	**12.459**	16.499
	p-valor K-S	2.20E-16	2.20E-16	2.20E-16	0.000
	p-valor K-W	2.20E-16	0.000	2.20E-16	2.20E-16

Tabla 6.5: Tabla comparativa de los AE con representación binaria para las instancias medianas

		Conjuntos de datos - instancias medianas			
Operadores	Operadores	#5	#6	#7	#8
One - Flip	máx(s)	78.622	116.447	54.979	61.598
	s	**78.609**	**116.447**	54.804	61.101
	p-valor K-S	0.000	2.20E-16	0.000	0.941
One - Add	máx(s)	78.622	116.447	54.429	62.216
	s	78.409	114.712	53.754	61.09
	p-valor K-S	0.003	0.000	0.120	0.782
One - Delete	máx(s)	78.622	116.447	54.979	63.437
	s	78.507	115.693	**54.979**	63.421
	p-valor K-S	0.000	0.022	2.20E-16	0.00147
Two - Flip	máx(s)	78.622	116.447	54.979	61.766
	s	78.548	116.407	54.879	61.084
	p-valor K-S	0.001	0.000	0.000	0.7865
Two - Add	máx(s)	78.622	115.431	54.526	61.579
	s	78.361	114.641	53.907	61.074
	p-valor K-S	0.000	0.000	0.035	0.991
Two - Delete	máx(s)	78.622	116.447	54.979	63.437
	s	78.423	115.341	**54.979**	**63.425**
	p-valor K-S	0.000	0.027	2.20E-16	0.000
	p-valor K-W	0.000	2.20E-16	2.20E-16	2.20E-16

Tabla 6.6: Tabla comparativa de los AE con representación binaria para las instancias grandes

		Conjuntos de datos - instancias grandes			
Operadores	Métricas	#9	#10	#11	#12
One - Flip	máx(s)	640.164	36.306	224.157	31.579
	s	633.908	35.884	221.675	31.195
	p-valor K-S	0.6344	0.3187	0.6657	0.739
One - Add	máx(s)	640.873	36.347	224.405	31.614
	s	634.473	35.916	221.87	31.223
	p-valor K-S	0.6203	0.5125	0.3981	0.6222
One - Delete	máx(s)	675.246	38.296	236.441	33.31
	s	**675.197**	**38.222**	**236.113**	**33.227**
	p-valor K-S	4.71E-05	5.24E-05	6.06E-05	2.59E-05
Two - Flip	máx(s)	637.192	36.138	223.116	31.433
	s	634.03	35.891	221.717	31.201
	p-valor K-S	0.9303	0.878	0.942	0.752
Two - Add	máx(s)	639.502	36.269	223.925	31.547
	s	634.214	35.902	221.782	31.210
	p-valor K-S	0.674	0.479	0.619	0.796
Two - Delete	máx(s)	675.246	38.294	236.435	33.299
	s	675.196	38.221	**236.113**	**33.227**
	p-valor K-S	0.001	0.001	0.001	0.001
	p-valor K-W	2.20E-16	2.20E-16	2.20E-16	2.20E-16

En la última fila de las Tablas 6.4, 6.5 y 6.6 se reporta el p-valor del test de Kruskal–Wallis (p-valor K-W) sobre los valores de similitud obtenidos con cada algoritmo. Dado que se puede rechazar la hipótesis nula para todos los algoritmos, podemos afirmar que uno de los algoritmos obtuvo mejores resultados que los otros, se destaca en negrita el que alcanzó mejor promedio de valores de similitud.

En la Tabla 6.4 se observa que la combinación de operadores que obtuvo mejores resultados es *One-Delete*, mientras que en la Tabla 6.5 se puede ver que las combinaciones de operadores que alcanzan los mejores resultados son *One-Flip* y *Two-Delete*. En la Tabla 6.6 los operadores que obtuvieron los mejores resultados fueron *One-Delete* y *Two-Delete*, donde el *One-Delete* alcanzó el mejor resultado para las cuatro instancias de prueba. Además, *One-Delete* alcanzó mejores soluciones que *Two-Delete* si se consideran los máximos valores de similitud en lugar de los valores promedio.

Representación entera vs. representación binaria

A continuación, se presenta una comparativa de los resultados experimentales obtenidos por los algoritmos implementados para resolver el problema de clustering en su formulación monoobjetivo. En particular, se compara la representación binaria y la representación entera tomando la combinación de operadores evolutivos que obtuvieron los mejores resultados analizados anteriormente. El objetivo de esta comparación es encontrar la mejor combinación de operadores y su representación para continuar con el análisis de la evaluación experimental.

En la Tabla 6.7 se reportan los promedios de los mejores resultados para las combinaciones de operadores que alcanzaron los valores promedio más altos. Se destaca en negrita el que alcanzó mejor promedio de valores de similitud.

Tabla 6.7: Tabla comparativa de los AE: *One-Oneoc* vs. *GenC&S-Oneoc* vs. *One-Flip* vs. *One-Delete* vs. *Two-Delete*.

Conjuntos de datos	Similitud promedio				
	Representación entera		Representación binaria		
	One-Oneoc	GenC&S-Oneoc	One-Flip	One-Delete	Two-Delete
#1	**18.657**	**18.657**	**18.657**	**18.657**	**18.657**
#2	**1.963**	**1.963**	**1.963**	**1.963**	**1.963**
#3	12.419	12.444	12.272	**12.459**	**12.459**
#4	16.428	16.409	15.933	**16.500**	16.499
#5	78.349	78.156	**78.609**	78.507	78.423
#6	116.18	**116.391**	116.447	115.693	115.341
#7	54.713	54.683	54.804	**54.979**	**54.979**
#8	63.274	63.298	61.101	63.421	**63.425**
#9	673.568	656.556	633.908	**675.197**	675.196
#10	37.773	36.489	35.884	**38.222**	38.221
#11	235.334	229.575	221.1675	**236.113**	**236.113**
#12	32.887	32.082	31.195	**33.227**	**33.227**

En la Tabla 6.7 se observa que no hay una combinación de operadores que sea superior a las otras para todos los conjuntos de datos, sin embargo la combinación de operadores que obtuvo mejores resultados en la mayoría de las instancias de prueba fue *One-Delete*. Por esta razón, el algoritmo evolutivo con representación binaria, el operador de mutación

Delete Mutation y el operador de cruzamiento de un punto, es seleccionado para realizar la comparación con los algoritmos presentados de la sección 4.4. Además, tanto la representación como los operadores evolutivos son tomados para la variante multiobjetivo del algoritmo evolutivo.

AE monoobjetivo vs. Algoritmos de la literatura

A continuación, se presenta una comparativa de los resultados experimentales obtenidos por los algoritmos de la literatura y el algoritmo evolutivo que alcanzó los mejores resultados. Como se presentó en la Sección 4.4, los algoritmos de la literatura implementados para la comparación son los siguientes:

1. KMedoid
2. Local Search
3. Algoritmo Ávido
4. Algoritmo Linkage
5. Híbrido

En las Tablas 6.8, 6.9 y 6.10 se presentan los resultados alcanzados por los algoritmos enumerados anteriormente y el mejor algoritmo evolutivo seleccionado en la Sección 6.3.2 para las instancias chicas, medianas y grandes, respectivamente. Se presentan los mejores valores de similitud (máx(s)) y el promedio de los mejores valores de similitud (s) obtenidos en las 30 ejecuciones independientes. Además, se reporta el p-valor resultante del test de Kolmogorov-Smirnov (p-valor K-S) sobre los valores de similitud obtenidos por cada algoritmo evolutivo, de manera de contrastar normalidad en los resultados alcanzados.

Tabla 6.8: Tabla comparativa AE monoobjtivo vs. algoritmos de la literatura para las instancias chicas

		Conjuntos de datos - instancias chicas			
Algoritmos comparativos	Métrica	#1	#2	#3	#4
Algoritmo Ávido	máx(s)	9.325	1.615	7.476	10.213
	s	7.281	1.115	5.771	7.41
	p-valor K-S	0.854	0.704	0.652	0.504
Linkage	máx(s)	17.009	1.651	10.175	14.037
	s	17.009	1.651	10.175	14.037
	p-valor K-S	2.20E-16	2.20E-16	2.20E-16	2.20E-16
KMedoid	máx(s)	18.657	1.963	12.459	16.395
	s	17.033	1.948	12.139	15.999
	p-valor K-S	0.104	1.06E-07	0.049	0.342
Local Search	máx(s)	18.657	1.963	11.657	15.523
	s	15.49	1.696	10.502	13.234
	p-valor K-S	0.459	0.037	0.944	0.997
Híbrido	máx(s)	18.657	1.963	12.459	16.483
	s	**18.657**	**1.963**	**12.452**	16.222
	p-valor K-S	2.20E-16	2.20E-16	0.000	0.872
One - Delete	máx(s)	18.657	1.963	12.459	16.502
	s	**18.657**	**1.963**	**12.459**	**16.5**
	p-valor K-S	2.20E-16	2.20E-16	2.20E-16	0.001
	p-valor K-W	2.20E-16	2.20E-16	2.20E-16	2.20E-16

Tabla 6.9: Tabla comparativa AE monoobjtivo vs. algoritmos de la literatura para las instancias medianas

		Conjuntos de datos - instancias medianas			
Algoritmos comparativos	Métricas	#5	#6	#7	#8
Algoritmo Ávido	máx(s)	60.876	94.620	40.471	39.198
	s	47.685	83.610	29.311	31.807
	p-valor K-S	0.501	0.506	0.714	0.874
Linkage	máx(s)	76.077	109.676	50.770	62.252
	s	76.077	109.676	50.770	62.252
	p-valor K-S	2.20E-16	2.20E-16	2.20E-16	2.20E-16
KMedoid	máx(s)	78.622	116.447	54.979	63.437
	s	76.469	116.297	54.979	62.509
	p-valor K-S	0.267	6.13E-08	2.20E-16	0.122
Local Search	máx(s)	77.730	115.426	51.763	59.143
	s	69.108	108.863	41.677	52.987
	p-valor K-S	0.932	0.824	0.579	0.799
Híbrido	máx(s)	78.622	116.447	54.979	63.437
	s	**78.622**	**116.447**	**54.979**	63.242
	p-valor K-S	2.20E-16	2.20E-16	2.20E-16	0.016
One - Delete	máx(s)	78.622	116.447	54.979	63.437
	s	78.507	115.693	**54.979**	**63.421**
	p-valor K-S	1.21E-06	0.02242	2.20E-16	0.001
	p-valor K-W	2.20E-16	2.20E-16	2.20E-16	2.20E-16

Tabla 6.10: Tabla comparativa AE monoobjtivo vs. algoritmos de la literatura para las instancias grandes

		Conjuntos de datos - instancias grandes			
Algoritmos comparativos	Métricas	#9	#10	#11	#12
Algoritmo Ávido	máx(s)	542.033	26.583	190.812	24.596
	s	499.535	22.898	170.654	22.796
	p-valor K-S	0.6209	0.712	0.486	0.607
Linkage	máx(s)	523.189	32.194	200.665	27.948
	s	523.189	30.611	198.750	27.018
	p-valor K-S	2.20E-16	2.20E-16	2.20E-16	2.20E-16
KMedoid	máx(s)	671.800	37.803	236.236	33.217
	s	667.940	37.092	236.100	32.847
	p-valor K-S	0.909	0.888	0.633	0.947
Local Search	máx(s)	647.082	33.887	210.875	29.958
	s	615.639	32.940	205.956	28.555
	p-valor K-S	0.8177	0.587	0.974	0.782
Híbrido	máx(s)	665.735	37.462	230.956	33.227
	s	661.482	36.734	229.556	33.100
	p-valor K-S	0.405	0.1785	0.564	0.472
One - Delete	máx(s)	675.246	38.296	236.441	33.31
	s	**675.197**	**38.222**	**236.113**	**33.227**
	p-valor K-S	4.71E-05	5.24E-05	6.06E-05	2.59E-05
	p-valor K-W	2.20E-16	2.20E-16	2.20E-16	2.20E-16

En la última fila de las Tablas 6.8, 6.9 y 6.10 se reporta el $p-$valor del test de Kruskal-Wallis ($p-$valor K-W) sobre los mejores valores de similitud obtenidos de cada algoritmo. Es posible rechazar la hipótesis nula, es por esto que se destaca en negrita el

algoritmo que obtuvo mejores valores promedio de similitud. Se observa que el algoritmo que alcanza los mejores resultados en la mayoría de las instancias es el algoritmo evolutivo con representación binaria y operadores *One-Delete*, el segundo algoritmo que obtiene los mejores resultados es el Híbrido, que es también un algoritmo evolutivo.

En la Figura 6.1 se muestra, para cada instancia de prueba, el porcentaje de mejora del mejor AE en su variante monoobjetivo respecto a los otros algoritmos implementados.

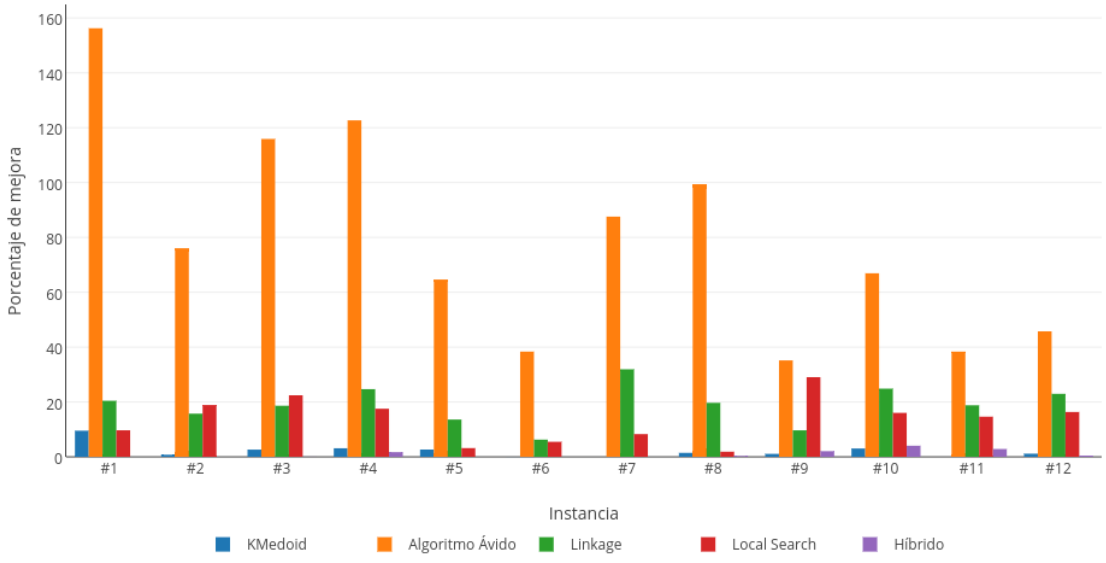


Figura 6.1: Porcentajes de mejora del AE monoobjetivo sobre los otros algoritmos

En la Figura 6.1 se observa que, para las instancias chicas, el algoritmo evolutivo es capaz de mejorar hasta en un 9.5 % el fitness alcanzado por el algoritmo KMedoid (4.0 % en promedio). Para las instancias medianas, el porcentaje de mejora del AE sobre el algoritmo KMedoid es de hasta 2.7 % (0.9 % en promedio). Mientras que para las instancias grandes, el algoritmo evolutivo mejora hasta en un 3.0 % los resultados del algoritmo KMedoid (1.3 % en promedio). Se destaca que el AE obtiene mejores resultados que el algoritmo KMedoid, el cual es el más utilizado en la literatura analizada en el marco del proyecto.

Además, se observa que, para las instancias chicas, el algoritmo evolutivo es capaz de mejorar hasta en un 156.2 % el fitness alcanzado por el algoritmo ávido (117.7 % en promedio). Para las instancias medianas, el porcentaje de mejora del AE sobre el algoritmo ávido es de hasta un 99.4 % (72.5 % en promedio). Mientras que para las instancias grandes, el AE mejora hasta en un 66.9 % los resultados del algoritmo ávido (46.6 % en promedio). Se puede concluir que el AE monoobjetivo alcanza resultados significativamente mejores que lo obtenidos por el algoritmo ávido en todas las instancias de prueba.

Respecto al algoritmo Linkage, para las instancias chicas el AE obtiene mejoras de hasta un 22.5 % (17.1 % en promedio), mientras que para las instancias medianas las mejoras son de un 8.3 % como máximo (en promedio 4.7 %) y para las instancias grandes

el AE obtiene mejoras de hasta un 29.1 % respecto al algoritmo Linkage (23.9 % en promedio).

El porcentaje de mejora del algoritmo evolutivo respecto al algoritmo Local Search es como máximo de 24.7 % (19.9 % en promedio) para las instancias chicas, mientras que para las instancias medianas es de hasta un 31.9 % (17.9 % en promedio) y para las instancias grandes el algoritmo evolutivo obtiene resultados hasta un 16.4 % mejores que el algoritmo Local Search (14.2 % en promedio).

En la Figura 6.1 se observa que el porcentaje de mejora del algoritmo evolutivo respecto al Híbrido es cercano a cero para la mayoría de las instancias. En el mejor caso (instancia grande #10), el AE fue capaz de mejorar hasta en un 4.0 % (2.3 % en promedio) los resultados obtenidos por el algoritmo Híbrido. Es importante destacar que, en este caso, la comparación se realiza entre dos algoritmos evolutivos.

Se constata que el AE es capaz que encontrar mejores soluciones en la mayoría de las instancias que todos los algoritmos de la literatura que fueron implementados.

6.4. Variante multiobjetivo del problema de clustering

En la presente sección se detalla el análisis experimental correspondiente a la variante multiobjetivo del problema de clustering.

6.4.1. Operadores evolutivos y parámetros

Como se concluye en la Sección 6.3.2, el algoritmo que obtiene mejores resultados en la variante monoobjetivo es el que utiliza una representación binaria, el operador de cruzamiento de un punto (*One*) y el operador de mutación donde se eliminan centros de la solución (*Delete*). Por esta razón, la implementación para el AE multiobjetivo utiliza una representación binaria, un operador de cruzamiento *One* y un operador de mutación *Delete*, adaptados a esta variante del algoritmo.

A continuación se describen los parámetros utilizados en la implementación del AE en su versión multiobjetivo. Como se mencionó en la Sección 5.3 se utiliza NSGA-II para la implementación de este algoritmo. Los parámetros utilizados son los siguiente:

- Probabilidad de cruzamiento $p_C = 0,75$.
- Probabilidad de mutación $p_M = 0,01$.
- Selección por torneo de tamaño 2.
- Criterio de parada: alcanzar la cantidad de generaciones $gen = 1000$.
- Inicialización aleatoria.
- Tamaño de la población $pob = 100$.

6.4.2. Hipervolumen relativo como métrica para optimización multiobjetivo

Deb [9] en su análisis, concluye que existen dos meta-objetivos para los problemas multiobjetivos, el primero es encontrar un conjunto de soluciones que se aproxime al frente de Pareto y, el segundo es que ese conjunto de soluciones sea lo mas diverso posible.

Se utiliza el hipervolumen relativo porque es una medida tanto de la convergencia como de la diversidad del conjunto de soluciones.

El hipervolumen relativo (RHV) es la tasa entre los volúmenes (en el espacio de las funciones objetivo) cubierto por el frente de Pareto calculado y el frente de Pareto aproximado. El valor ideal de hipervolumen relativo es 1. Se considera el frente de Pareto aproximado al conjunto soluciones no dominadas obtenidas al combinar las soluciones encontradas en el total de ejecuciones independientes del AE multiobjetivo, debido a que no se conoce el frente Pareto real. El RHV es calculado para todas las instancias de prueba.

6.4.3. Resultados numéricos

A continuación, se presentan los resultados numéricos del análisis experimental realizado sobre el algoritmo NSGA–II correspondientes a la variante multiobjetivo (*MO*) del problema de clustering. Se realizaron 30 ejecuciones independientes de cada AE para cada una de las instancias descritas en la Sección 6.1.

Hipervolumen relativo

Para calcular el hipervolumen relativo es necesario calcular el hipervolumen del conjunto de soluciones no dominadas resultante de 30 ejecuciones del AE multiobjetivo (frente de Pareto real) y calcular el hipervolumen del frente obtenido en cada ejecución independiente. El hipervolumen relativo es la tasa entre estos dos valores. La Tabla 6.11 muestra, para cada instancia, el valor promedio del hipervolumen relativo de las 30 ejecuciones independientes.

Tabla 6.11: Promedio del hipervolumen relativo para la variante multiobjetivo del AE

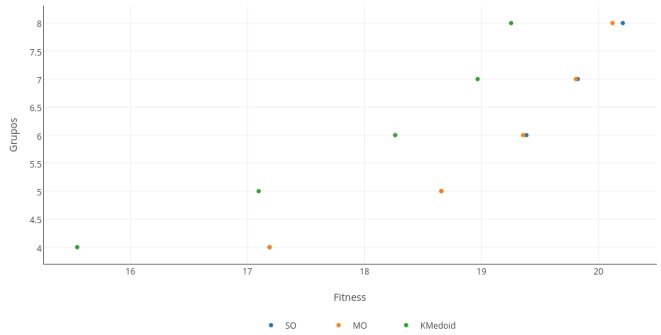
	Instancias												
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13
RHV	0.99	1.00	1.00	1.00	0.98	0.99	0.99	0.99	0.98	0.99	0.99	0.98	0.99

Se observa que para todas las instancias de prueba los valores de RHV son cercanos a 1, esto refleja una buena convergencia al frente de Pareto real, y que el algoritmo es capaz de conservar buena diversidad en el frente de soluciones halladas.

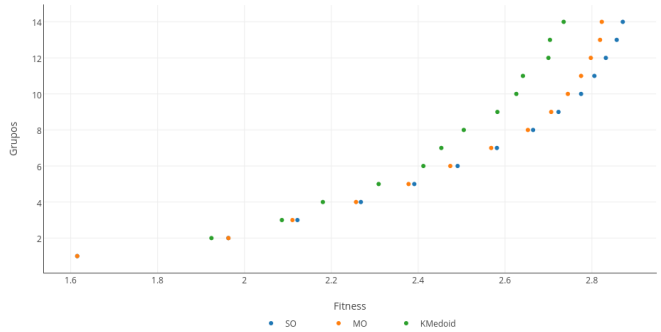
Porcentaje de mejora

Con el fin de comparar los resultados calculados por el AE multiobjetivo con otras técnicas, se utilizaron los algoritmos correspondientes a la variante monoobjetivo del problema, variando el número de grupos. En particular, se utilizó el AE monoobjetivo (SO) y el algoritmo KMedoid. Realizando 30 ejecuciones independientes de cada algoritmo y variando la cantidad de grupos, se obtuvieron dos conjuntos de soluciones para ser comparados contra los frentes de Pareto calculados por el AE multiobjetivo.

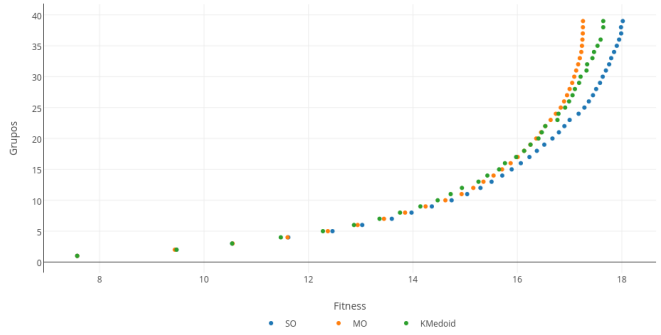
En las Figuras 6.2, 6.3 y 6.4 se muestra el frente de Pareto real hallado con el algoritmo evolutivo en su variante multiobjetivo y los conjuntos de soluciones calculados para SO y KMedoid.



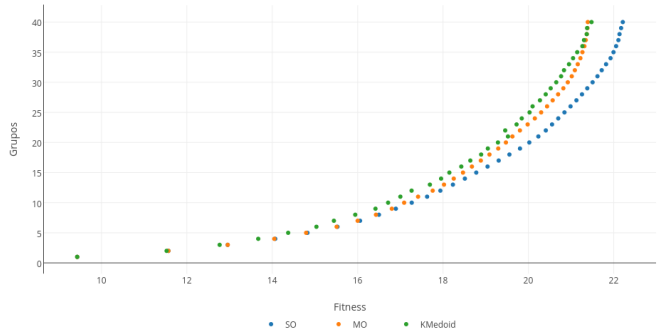
(a) Conjuntos de soluciones para la instancia #1



(b) Conjuntos de soluciones para la instancia #2

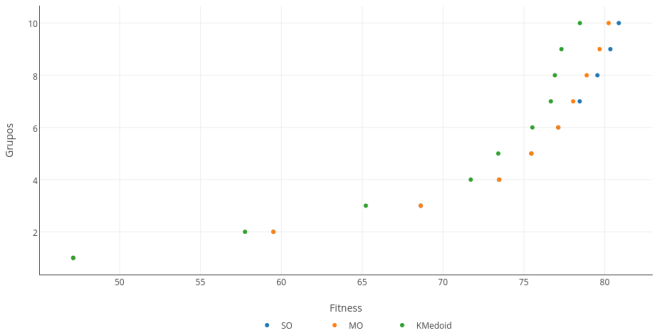


(c) Conjuntos de soluciones para la instancia #3

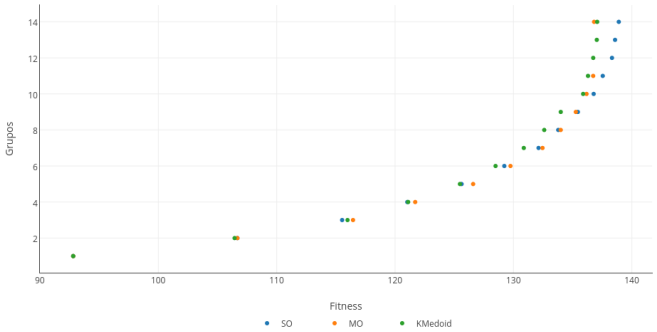


(d) Conjuntos de soluciones para la instancia #4

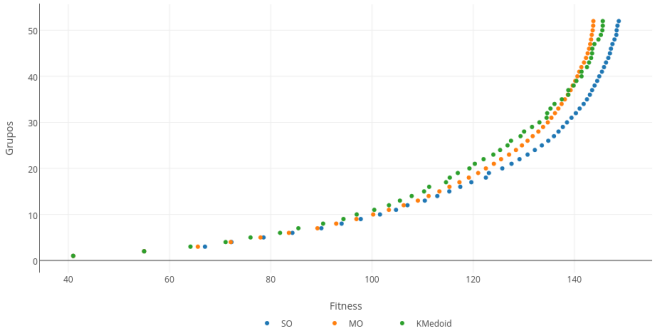
Figura 6.2: AE Monoobjetivo vs. AE Multiobjetivo vs. KMedoid - comparación mejores resultados para instancias chicas



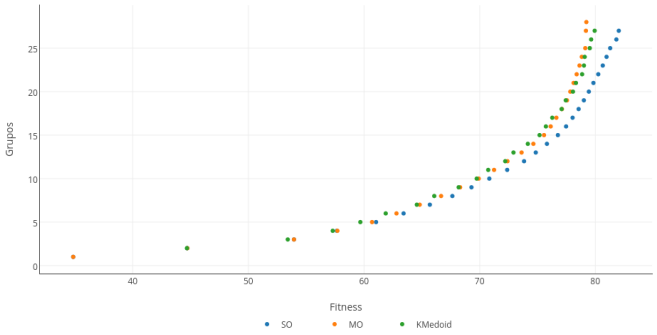
(a) Conjuntos de soluciones para la instancia #5



(b) Conjuntos de soluciones para la instancia #6

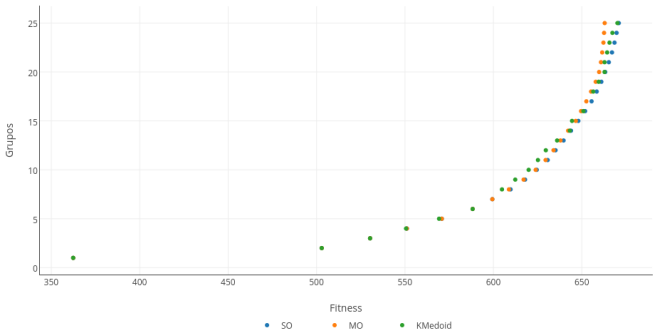


(c) Conjuntos de soluciones para la instancia #7

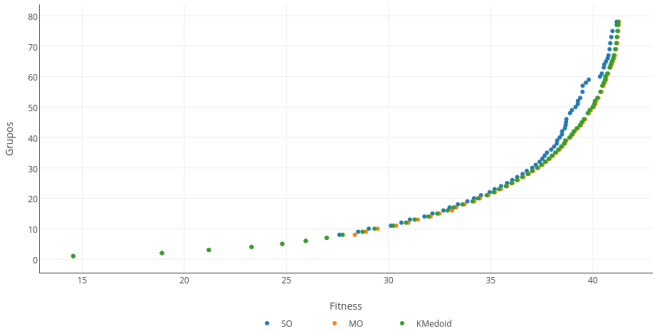


(d) Conjuntos de soluciones para la instancia #8

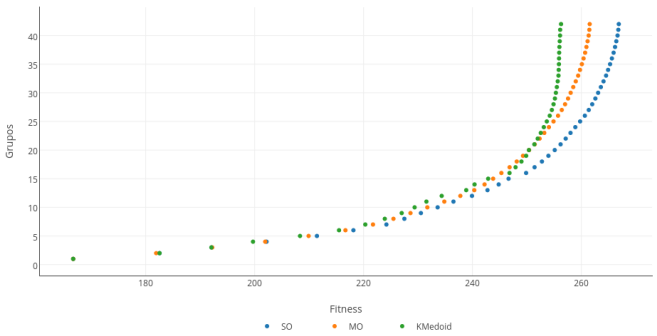
Figura 6.3: AE Monoobjetivo vs. AE Multiobjetivo vs. KMedoid - comparación mejores resultados para instancias medianas



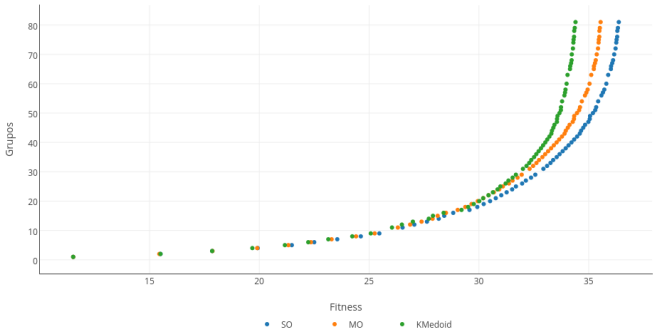
(a) Conjuntos de soluciones para la instancia #9



(b) Conjuntos de soluciones para la instancia #10



(c) Conjuntos de soluciones para la instancia #11



(d) Conjuntos de soluciones para la instancia #12

Figura 6.4: AE Monoobjetivo vs. AE Multiobjetivo vs. KMedoid - comparación mejores resultados para instancias grandes

En las gráficas se puede apreciar que, cuando la cantidad de grupos es pequeña, los conjuntos de soluciones calculados para SO y Kmedoid y el frente de Pareto aproximado son cercanos. Sin embargo, al aumentar la cantidad de grupos, se evidencia la ventaja del SO sobre los algoritmos MO y KMedoid.

Para realizar la comparación de estos tres algoritmos se utilizó el porcentaje de mejora del algoritmo SO respecto del MO y KMedoid. La Tabla 6.12 muestra el valor máximo de este porcentaje.

Tabla 6.12: Porcentaje de mejora (%) del AE en su variante monoobjetivo sobre la variante multiobjetivo y KMedoid

Instancias de prueba	SO sobre KMedoid	SO sobre MO
#1	9.57	0.43
#2	5.99	1.68
#3	2.40	4.23
#4	4.66	3.69
#5	4.95	0.83
#6	1.31	1.50
#7	5.82	3.38
#8	2.66	3.42
#9	7.03	1.19
#10	2.66	2.10
#11	4.00	2.00
#12	5.42	2.28
#13	31.42	8.76

Se observa que el máximo porcentaje de mejora del algoritmo SO sobre el algoritmo KMedoid se obtiene para la instancia #13 (con 3056 elementos a agrupar), con un 31.42 %. El mayor porcentaje de mejora del algoritmo SO sobre el algoritmo MO, se da en el instancia #13 también, con un 8.76 %. Para las instancias de prueba restantes el porcentaje de mejora del algoritmo SO respecto del MO es aproximadamente de 2 %, por lo tanto los valores obtenidos por el AE multiobjetivo se aproximan a los resultados obtenidos por el AE monoobjetivo. Las diferencias entre el AE SO y el MO son pequeñas, considerando que el SO se enfoca exclusivamente en maximizar la similitud, mientras que el MO debe simultáneamente minimizar la cantidad de grupos. De todas formas, el enfoque multiobjetivo puede ser útil para ofrecerle a un tomador de decisión, un conjunto de soluciones con distinto nivel de compromiso.

Capítulo 7

Conclusiones y trabajo a futuro

Esta sección presenta las conclusiones del trabajo realizado en el marco del proyecto y las principales líneas de trabajo futuro.

7.1. Conclusiones

Este proyecto de grado presentó la resolución del problema de clustering utilizando algoritmos evolutivos. Se presentaron dos variantes del problema: una variante monoobjetivo, donde se plantea maximizar la sumatoria de las similitudes de los elementos al centro del grupo al que pertenece; y una variante multiobjetivo, donde se considera como objetivo complementario minimizar la cantidad de grupos formados. Se estudió el problema, se relevaron los principales trabajos de la literatura relacionada y se seleccionaron los algoritmos más relevantes de la literatura para la comparación con los algoritmos evolutivos implementados. Un total de doce algoritmos evolutivos fueron implementados para la variante monoobjetivo, seis utilizando una representación entera y seis utilizando una representación binaria. Se implementó un algoritmo evolutivo con un enfoque multiobjetivo explícito (NSGA–II) para la variante multiobjetivo del problema de clustering.

El análisis experimental fue realizado sobre un conjunto de instancias del problema, obtenidas de la literatura relacionada. Para la evaluación experimental de los algoritmos que resuelven la variante monoobjetivo del problema se utilizó un conjunto de 12 instancias de prueba, mientras que para los algoritmos que resuelven la variante multiobjetivo se utilizó, además de las 12 instancias, una instancia de prueba con 3056 elementos que fue brindada por los colegas de la Universidad de Luxemburgo.

Los AE implementados para resolver la variante monoobjetivo del problema de clustering fueron comparados entre ellos para seleccionar la representación y los operadores evolutivos que alcanzaron los mejores resultados. Los resultados experimentales determinan que el AE con representación binaria y operadores *One-Delete* permite alcanzar mejores resultados que los otros algoritmos evolutivos implementados. Habiendo seleccionado este AE, se lo comparó a algoritmos implementados en base a ideas presentadas en trabajos de la literatura relacionada, descritos en la Sección 4.4. El algoritmo con representación binaria y operadores *One-Delete* fue capaz de mejorar los resultados del algoritmo ávido hasta en un 156.2 %, mientras que respecto al algoritmo KMedoid fue capaz de alcanzar mejoras de hasta un 9.5 %. Los resultados experimentales muestran que el AE en su variante monoobjetivo fue capaz de encontrar mejoras estadísticamente sig-

nificativas sobre la mayoría de los resultados obtenidos por los algoritmos implementados para la comparación en las 12 instancias de prueba. No se obtienen mejoras significativas respecto al algoritmo Hibrído (4.0 % en el mejor caso), esto puede adjudicarse a que el algoritmo Híbrido es también un algoritmo evolutivo.

El algoritmo implementado para resolver la variante multiobjetivo del problema utiliza la representación y los operadores evolutivos adaptados del mejor algoritmo evolutivo monoobjetivo evaluado en la Sección 6.3.2. Los resultados experimentales muestran que el AE multiobjetivo logró resultados mejores que el algoritmo KMedoid en la mayoría de las instancias de prueba a medida que aumenta la cantidad de grupos. Respecto al AE monoobjetivo, cuando la cantidad de grupos disminuye el AE multiobjetivo alcanza los mismos resultados mientras que a medida que aumenta la cantidad de grupos el AE monoobjetivo consigue resultados aproximadamente un 2 % superiores.

7.2. Trabajo futuro

A continuación, se detallan las principales líneas de trabajo futuro que surgen a partir del trabajo realizado en el marco de este proyecto.

En el proyecto se implementaron diversos algoritmos para resolver el problema de clustering, en la Sección 6.3.2 se seleccionó el AE que obtuvo mejores resultados, sin embargo el tiempo de ejecución de este algoritmo es mayor al registrado por los algoritmos evolutivos que utilizan la representación entera. Por esta razón, se plantea como una de las líneas de trabajo futuro la implementación de una variante del algoritmo seleccionado en la Sección 6.3.2 que utilice el modelo de subpoblaciones distribuidas, de forma de obtener buenos resultados en calidad de soluciones y tiempos de ejecución reducidos.

Los algoritmos evolutivos implementados a lo largo del estudio, utilizan una única inicialización de la población. Sería de interés evaluar distintos mecanismos de inicialización como por ejemplo, utilizar el resultado de una ejecución del algoritmo KMedoid para inicializar la población. Realizar la comparación de la inicialización aleatoria con una inicialización utilizando el algoritmo KMedoid.

La evaluación experimental se basó principalmente en analizar dos tipos de representación y diferentes operadores evolutivos dejando al margen la configuración paramétrica de cada algoritmo. Sería útil experimentar, para cada algoritmo, con diferentes valores de sus parámetros y realizar la evaluación experimental comparando cada algoritmo con sus mejores parámetros.

Capítulo 8

Bibliografía

[1] Ilog cplex 11.0. `http://www-eio.upc.es/lceio/manuals/cplex-11/html/`. [Online; accedido 4-Agosto-2016].

[2] Knowledge extraction based on evolutionary learning. `http://sci2s.ugr.es/keel/category.php?cat=clas`. [Online; accedido 15-Marzo-2016].

[3] Matlab. `http://www.mathworks.com/help/stats/linkage.html`. [Online; accedido 26-Junio-2016].

[4] D. Aloise, A. Deshpande, P. Hansen, y P. Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

[5] A. Bokan, R. Patiño, y Y. Túpac. Validación de clusters usando ieka y sl-som. en *X Congreso de la Sociedad Peruana de Computación*, pp. 161–170, 2011.

[6] C. Darwin. *On the origin of Species by Means of Natural Selection.* New York, Appleton and Co., 1860.

[7] S. Dasgupta. *The hardness of k-means clustering.* Department of Computer Science and Engineering, University of California, San Diego, 2008.

[8] D. Deaven y K. Ho. Molecular geometry optimization with a genetic algorithm. vol. 75, pp. 288–291. American Physical Society, 1995.

[9] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms.* Wiley, 2001.

[10] K. Deb. Multi-objective evolutionary algorithms. en *Springer Handbook of Computational Intelligence*, pp. 995–1015. Springer Nature, 2015.

[11] K. Deb, A. Pratap, S. Agarwal, y T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6 (2):182–197, 2002.

[12] Y. Deng y J. Bard. A reactive grasp with path relinking for capacitated clustering. *Journal of Heuristics*, 17(2):119–152, 2011.

[13] B. Everitt, S. Landau, y M. Leese. *Cluster Analysis.* Wiley Publishing, 4ta edición, 2009.

[14] A. Freitas. *A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery*, pp. 819–845. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[15] M. Gendreau y J. Potvin. *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer, 2010.

[16] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[17] J. Hartigan y M. Wong. A K-means clustering algorithm. *Applied Statistics*, 28: 100–108, 1979.

[18] J. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence (Complex Adaptive Systems)*. The MIT Press, 1992.

[19] A. Jain y R. Dubes. *Algorithms for Clustering Data*. 1988.

[20] J. Karimov y M. Ozbayoglu. High quality clustering of big data and solving empty-clustering problem with an evolutionary hybrid algorithm. en *Conference on Big Data, 2015 IEEE International*, pp. 1473–1478, 2015.

[21] L. Kaufman y P. Rousseeuw. *Finding groups in data : an introduction to cluster analysis*. Wiley series in probability and mathematical statistics. Wiley, New York, 1990.

[22] K. Krishna y M. Narasimha. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3):433–439, 1999.

[23] W. Kruskal y W. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[24] S. Luke. ECJ. A Java-based Evolutionary Computation Research System . `https://cs.gmu.edu/~eclab/projects/ecj/`, 2016. [Online; accedido 18-Junio-2016].

[25] J. MacQueen. Some methods for classification and analysis of multivariate observations. en *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, Berkeley, Calif., 1967. University of California Press.

[26] M. Mahajan, P. Nimbhorkar, y K. Varadarajan. The planar k-means problem is np-hard. en *Proceedings of the 3rd International Workshop on Algorithms and Computation*, pp. 274–285, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] M. Matsumoto y T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[28] G. Mecca, S. Raunich, y A. Pappalardo. A new algorithm for clustering search results. *Data & Knowledge Engineering*, 62(3):504 – 522, 2007.

[29] M. Mitchell. *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. The MIT Press, 1996.

[30] Nisha y P. Kaur. Cluster quality based performance evaluation of hierarchical clustering method. en *1st International Conference on Next Generation Computing Technologies (NGCT)*, pp. 649–653, 2015.

[31] H. Park y C. Jun. A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl.*, 36(2):3336–3341, 2009.

[32] F. Pereira y J. Marques. *Analysis of Crossover Operators for Cluster Geometry Optimization*, pp. 77–89. Springer Netherlands, Dordrecht, 2011.

[33] W. Sheng y X. Liu. A hybrid algorithm for k-medoid clustering of large data sets. en *Congress on Evolutionary Computation, 2004. CEC2004.*, vol. 1, pp. 77–82, 2004.

[34] N. Srinivas y K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.*, 2(3):221–248, 1994.

[35] P. Tan, M. Steinbach, y V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[36] S. Theodoridis y K. Koutroumbas. *Pattern Recognition, Second Edition*. Academic Press, 2003.

[37] L. Tucker. The extension of factor analysis to three-dimensional matrices. en *Contributions to mathematical psychology.*, pp. 110–127. Holt, Rinehart and Winston, New York, 1964.

[38] G. Yu, F. Li, Y. Qin, X. Bo, Y. Wu, y S. Wang. Gosemsim: an r package for measuring semantic similarity among go terms and gene products. *Bioinformatics*, 26(7):976–978, 2010.