



Universidad Simón Bolívar
Decanato de Estudios Profesionales
Coordinación de Ingeniería de la Computación

Metaheurísticas Bio-inspiradas para Selección de Instancias

Por:
Alejandro Flores V.

Realizado con la asesoría de:

Emely Arráiz B.

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de Computación

Sartenejas, septiembre de 2014



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL PROYECTO DE GRADO

**METAHEURÍSTICAS BIO-INSPIRADAS
PARA SELECCIÓN DE INSTANCIAS**

Presentado por:
Alejandro Flores V.

Este Proyecto de Grado ha sido aprobado por el siguiente jurado examinador:

Emely Arráiz B.

@jurado1

@jurado2

Sartenejas, @día de @mes de @año

Resumen

En este trabajo se considera el problema de *Selección de Instancias* (SI), como estrategia de reducción de datos durante la aplicación de procesos de KDD. Estando los datos agrupados en instancias independientes (una entrada en una base de datos), el problema de SI busca seleccionar el subconjunto de instancias de menor cardinalidad, que mantenga o mejore la precisión de clasificación al ser usado como conjunto de entrenamiento. El estudio de este problema se ha popularizado a lo largo de los últimos años debido a la creciente necesidad de reducir los datos a ser almacenados y posteriormente analizados. Dada su inherente complejidad, los trabajos sobre el problema de SI se han centrado en la formulación de algoritmos heurísticos que puedan generar (en tiempos aceptables) buenas soluciones al problema. Durante la última década, numerosos autores han acudido al uso de metaheurísticas para encontrar soluciones al problema de SI, debido a su comprobada capacidad para encontrar buenas soluciones a gran variedad de problemas. En el presente trabajo se implementan 5 metaheurísticas (*i.e.* GGA, SGA, CHC, PBIL y PSO) para conseguir soluciones al problema de SI. Adicionalmente, se proponen una serie de modificaciones sobre las estrategias de generación de soluciones iniciales, con la finalidad de reducir la cardinalidad de las soluciones encontradas y guiar el inicio de la búsqueda a soluciones que satisfagan los objetivos del problema. La evaluación experimental muestra que la disminución de la probabilidad de aparición de bits al 5 % permite una disminución sustancial de los tiempos de ejecución de las metaheurísticas, mejorando los resultados en términos de la reducción alcanzada, y sin afectar negativamente la precisión del clasificador. Adicionalmente, un estudio comparativo entre las metaheurísticas evaluadas, concluye que PBIL obtiene los mejores resultados en función de los objetivos del problema. Sin embargo, el comportamiento exhibido por CHC resulta interesante con miras en su aplicación a conjuntos de datos de mayor tamaño, puesto que es la metaheurística que se ve menos afectada por el número de instancias.

Palabras clave: reducción de datos, selección de instancias, metaheurísticas.

Índice general

Resumen	I
Índice de Figuras	IV
Lista de Tablas	V
Índice de algoritmos	VI
Acrónimos y Símbolos	VII
Introducción	1
1. Selección de Instancias	4
1.1. Reducción de Datos	4
1.2. Selección de Instancias	6
1.2.1. Regla del Vecino Más Cercano (NN)	7
1.3. Algoritmos de aproximación para Selección de Instancias	10
1.3.1. Métodos basados en la regla NN	10
1.3.2. Métodos basados en eliminación ordenada	11
1.3.3. Métodos basados en muestreo aleatorio	12
1.3.4. Métodos basados en metaheurísticas	12
1.3.5. Criterios de comparación	13
2. Metaheurísticas	14
2.1. Descripción general	14
2.2. Metaheurísticas inspiradas en la naturaleza	15
2.2.1. Algoritmos Evolutivos	15
2.2.1.1. Algoritmo Genético Generacional (GGA)	16
2.2.1.2. Algoritmo Genético Estacionario (SGA)	17
2.2.1.3. CHC Adaptive Search Algorithm	18
2.2.1.4. Population-Based Incremental Learning (PBIL)	19
2.2.2. Inteligencia de Enjambre	21
2.2.2.1. Particle Swarm Optimization (PSO)	21

3. Adaptación al problema de Selección de Instancias	23
3.1. Consideraciones generales	23
3.1.1. Representación	23
3.1.2. Función de evaluación	24
3.1.3. Generación de soluciones iniciales	25
3.1.3.1. Condensed Nearest Neighbor	26
3.1.3.2. Nearest Enemy Hypersphere Selection	28
3.1.3.3. Closest Nearest Enemy	29
3.1.3.4. Farthest Nearest Enemy	30
3.2. Modificaciones particulares	30
3.2.1. Adaptación de GGA, SGA y CHC	30
3.2.2. Adaptación de PBIL	31
3.2.3. Adaptación de PSO	31
4. Evaluación Experimental	33
4.1. Diseño Experimental	33
4.1.1. Conjuntos de datos	34
4.1.2. Particiones y ejecuciones	35
4.1.3. Parámetros	36
4.1.4. Tablas de resultados	37
4.2. Resultados	38
4.2.1. Comparación entre inicializaciones	38
4.2.1.1. Inicialización disminuyendo la probabilidad de aparición de bit	39
4.2.1.2. Inicialización usando algoritmos heurísticos	41
4.2.2. Comparación entre metaheurísticas	42
4.2.2.1. Resultados para conjuntos de datos pequeños	43
4.2.2.2. Resultados para conjuntos de datos medianos	44
4.2.2.3. Resultados para conjuntos de datos grandes	46
4.2.2.4. Análisis de los resultados	47
Conclusiones y Recomendaciones	53
Bibliografía	55
A. Entonación de las metaheurísticas	61
A.1. Entonación de GGA	61
A.2. Entonación de SGA	62
A.3. Entonación de CHC	63
A.4. Entonación de PBIL	64
A.5. Entonación de PSO	66

Índice de figuras

1.1.	Diagramas de Voronoi y NN	8
3.1.	Algoritmos de Selección de Instancias	27
3.2.	Selección obtenida por NEHS	29
4.1.	Tamaño <i>vs</i> error de validación modificando δ	40
4.2.	Tamaño y error de validación usando algoritmos heurísticos	42
4.3.	Distribución de los resultados de cada metaheurística	50
4.4.	Tiempo de ejecución de cada metaheurística	52
A.1.	Resultados de entonación de GGA	62
A.2.	Resultados de entonación de SGA	63
A.3.	Resultados de entonación de CHC	64
A.4.	Resultados de entonación de PBIL	65
A.5.	Resultados de entonación de PSO	67

Índice de Tablas

4.1.	Conjuntos de datos pequeños	34
4.2.	Conjuntos de datos medianos	34
4.3.	Conjuntos de datos grandes	35
4.4.	Parámetros usados en cada metaheurística	37
4.5.	Resultados al disminuir la probabilidad de aparición de bit δ	39
4.6.	Ranking de valores probabilidad de aparición de bit δ	40
4.7.	Resultados usando probabilidades constantes y basadas en heurísticas .	41
4.8.	Ranking de probabilidades constantes o basadas en heurísticas	42
4.9.	Resultados de metaheurísticas usando conjuntos de datos pequeños .	44
4.10.	Ranking de metaheurísticas usando conjuntos de datos pequeños . . .	44
4.11.	Resultados de metaheurísticas usando conjuntos de datos medianos .	45
4.12.	Ranking de metaheurísticas usando conjuntos de datos medianos . . .	45
4.13.	Resultados de metaheurísticas usando conjuntos de datos grandes .	46
4.14.	Ranking de metaheurísticas usando conjuntos de datos grandes	47
4.15.	Comparación con los resultados publicados por <i>Cano et al.</i>	47
4.16.	Resultados promedio por metaheurística y conjunto de datos	48
4.17.	Ranking de metaheurísticas usando todos los conjuntos de datos . .	49
4.18.	Pruebas de <i>Wilcoxon</i> entre PBIL y las demás metaheurísticas . .	50
A.1.	Valores evaluados para los parámetros de GGA	62
A.2.	Valores evaluados para los parámetros de SGA	63
A.3.	Valores evaluados para los parámetros de CHC	64
A.4.	Valores evaluados para los parámetros de PBIL	64
A.5.	Valores evaluados para los parámetros de PSO	66

Índice de algoritmos

2.1.	Algoritmo Genético Generacional	17
2.2.	Algoritmo Genético Estacionario	18
2.3.	CHC Adaptive Search Algorithm	19
2.4.	Population-Based Incremental Learning	20
2.5.	Particle Swarm Optimization	22
3.1.	Generador de vector de probabilidades inicial	26
3.2.	Condensed Nearest Neighbor	28
3.3.	Nearest Enemy Hypersphere Selection	29
3.4.	Population-Based PSO	32

Acrónimos y Símbolos

KDD	Knowledge Discovery in Databases
MD	Minería de Datos
SI	Selección de Instancias
NN	Nearest Neighbor
NE	Nearest Enemy
CNN	Condensed Nearest Neighbor
NEHS	Nearest Enemy Hypersphere Selection
GGA	Generational Genetic Algorithm
SGA	Steady-State Genetic Algorithm
CHC	CHC Adaptive Search Algorithm
PBIL	Population-Based Incremental Learning
PSO	Particle Swarm Optimization

\in	Relación de pertenencia, « <i>es un elemento de</i> »
\subseteq	Subconjunto propio
\setminus	Diferencia de conjuntos

Introducción

Durante las últimas décadas, los avances tecnológicos han llevado a un aumento significativo en la cantidad de información generada por la actividad humana. Nuevos campos de estudio han emergido con la finalidad analizar esta enorme cantidad de datos. Bajo el contexto del campo de *Descubrimiento de Conocimiento en Bases de Datos* (KDD por sus siglas en inglés), los procesos de *Minería de Datos* (MD) buscan patrones en los conjuntos de datos con el objetivo de construir modelos de representación que permitan (entre otras cosas) clasificar datos desconocidos. Sin embargo, debido a que estos datos se encuentran agrupados en muestras de un evento particular (o instancias), surge el problema de aparición de instancias redundantes, con datos inconsistentes o ruidosos, etc. En este sentido, la reducción de los datos juega un rol fundamental en la aplicación efectiva de técnicas de MD.

El problema de *Selección de Instancias* (SI) busca escoger un subconjunto de las instancias originales, con la finalidad de reducir la cantidad de datos almacenados sin perjudicar (o incluso mejorando) la capacidad de representación original. Puede verse como un *problema de optimización combinatoria* en el que cada instancia puede pertenecer o no al subconjunto seleccionado; esto implica un espacio de posibles soluciones de tamaño 2^n , siendo n el número de instancias iniciales. Este problema ha sido estudiado ampliamente por numerosos autores, por lo que existe abundante literatura sobre algoritmos heurísticos para encontrar soluciones al problema de SI.

Debido a que las metaheurísticas son métodos de búsqueda estocástica de propósito general, han sido adaptadas en numerosas ocasiones para encontrar soluciones a problemas de optimización combinatoria. Las metaheurísticas proveen un esquema de búsqueda flexible, que permite recorrer el espacio de soluciones de diversos problemas.

Aunque no garantizan optimalidad, en general, las metaheurísticas son capaces de encontrar buenas soluciones en espacios de búsqueda particularmente grandes. Por esta razón, durante los últimos años, muchos autores han estudiado el uso de metaheurísticas para encontrar soluciones al problema de SI.

En este sentido, algunos de los primeros trabajos se centran en adaptar metaheurísticas de trayectoria para conseguir soluciones al problema de SI. Los trabajos de *Cerverón et al.* [CF01] y *Zhang et al.* [ZS02] introducen modificaciones al algoritmo de *Búsqueda Tabú*. No obstante, la mayoría de los estudios se han enfocado en adaptaciones de *Algoritmos Evolutivos*. En la literatura se encuentran adaptaciones de *Algoritmos de Estimación de Distribución* [SLI⁺01], *Algoritmo Genético Inteligente* [HLL02], *Algoritmo Memético Estacionario* [GCH08], etc. Destaca el trabajo de *Cano et al.* [CHL03], que realiza un estudio comparativo entre algoritmos heurísticos al problema de SI y adaptaciones de *Algoritmo Genético Generacional* (GGA), *Algoritmo Genético Estacionario* (SGA), *CHC Adaptive Search Algorithm* (CHC) y *Population-Based Incremental Learning* (PBIL).

Sin embargo, uno de los principales obstáculos para la aplicación de metaheurísticas en el problema de SI, es que se requiere un alto número de iteraciones para encontrar soluciones que cumplan satisfactoriamente los objetivos del problema en términos de reducción de los datos y precisión de representación.

En el presente trabajo se realiza un estudio comparativo entre 5 metaheurísticas: GGA, SGA, CHC, PBIL y PSO. En función del obstáculo planteado, se proponen una serie de modificaciones a las estrategias de generación de soluciones iniciales de estas metaheurísticas. La idea es iniciar el proceso de búsqueda en regiones del espacio de soluciones que mejoren los resultados en términos de los objetivos del problema. Esto con la finalidad de reducir el número de iteraciones necesarias para alcanzar buenas soluciones al problema y así permitir su aplicación sobre conjuntos de datos con gran número de instancias.

Este trabajo está estructurado como sigue. En el Capítulo 1 se describe en amplitud el problema de SI y los enfoques seguidos en la literatura para su solución. En el Capítulo 2 se describen de forma general las metaheurísticas usadas en este estudio, mientras que en el Capítulo 3 se presentan las modificaciones propuestas sobre las

metaheurísticas para conseguir soluciones al problema de SI y resolver el obstáculo mencionado. En el Capítulo 4 se detalla el diseño experimental seguido y se presentan y analizan los resultados obtenidos. Finalmente, se presentan las conclusiones y recomendaciones finales del trabajo.

Capítulo 1

Selección de Instancias

Este capítulo describe el problema de *Selección de Instancias* (SI), como estrategia de reducción de datos para la aplicación de técnicas de *Minería de Datos* (MD). El problema es definido formalmente, se describen sus principales características y se realiza un breve análisis del estado del arte.

1.1. Reducción de Datos

Como parte del proceso de *Descubrimiento de Conocimiento en Bases de Datos* (Knowledge Discovery in Databases o *KDD*), la fase de preprocesamiento de los datos juega un rol fundamental para la aplicación efectiva de técnicas de *Minería de Datos* (*MD*). Una de las estrategias de mayor uso durante la fase de preprocesamiento es la de *reducción de datos*.

El problema de *reducción de datos* consiste en decidir qué datos deben ser utilizados durante la aplicación de algoritmos de *MD* con el objetivo de construir modelos representativos de los datos originales. Dicha decisión debe basarse en la relevancia de los datos con respecto a los objetivos que se persiguen, o inclusive, por limitaciones técnicas. En términos prácticos, la importancia del problema de *reducción de datos* radica en los siguientes factores: *a) Tiempo y Espacio*: Mientras mayor sea el número de datos a utilizar, mayor será el espacio necesario para almacenarlos y el tiempo requerido para analizarlos. *b) Sensibilidad al ruido*: Al aumentar el número de instancias en el conjunto de datos, también lo hace la probabilidad de aparición de datos

atípicos, inconsistentes o redundantes. Su eliminación se vuelve necesaria para evitar un impacto negativo en los modelos de representación creados a partir de los datos.

En función de estos criterios, y basados en la definición de los datos, se han formulado diferentes estrategias para llevar a cabo la fase de reducción. En los procesos de KDD, un conjunto de datos está definido en función de un conjunto de clases Ω y un conjunto T de n observaciones de un evento, cada una con m mediciones, donde:

Definición 1. Una **instancia** t_i (con $i = 1 \dots n$) es una observación del evento; donde $t_i = (v_{i,1}, v_{i,2}, \dots, v_{i,m})$ es una tupla de m valores/mediciones (un punto en un espacio m -dimensional). Adicionalmente, cada instancia t_i pertenece a la *clase* $\omega_{t_i} \in \Omega$.

Definición 2. Un **atributo** p_j (con $j = 1 \dots m$) define el conjunto de mediciones «*de un mismo tipo*» para todas las observaciones, *i.e.* $p_j = \{v_{i,j} \mid i = 1 \dots n\}$. Cada atributo puede presentarse en diferentes formatos: *nominales*, *discretos*, o *continuos*.

En la literatura se han definido varias estrategias para realizar el proceso de *reducción de datos*. Entre ellas las más relevantes son:

- **Selección de Instancias** [BL97, LM02]: Busca la reducción del conjunto de datos mediante la selección de un subconjunto de instancias, de forma tal que dicho subconjunto conserve las capacidades de representación del conjunto original.
- **Selección de Atributos** [BL97, LM98]: Esta técnica permite eliminar atributos del conjunto de datos original, que no contribuyen (o que influyen negativamente) a la construcción de un modelo representativo.
- **Discretización de Atributos** [FI93, LHTD02]: Esta estrategia busca convertir atributos *continuos* en *discretos* (cuantificando el espacio de posibles valores), o disminuir el número de valores *discretos* (combinando valores adyacentes).

Digamos que se tiene una base de datos de pacientes en un hospital, y se desea entrenar un clasificador para predecir la existencia de enfermedades cardíacas en nuevos pacientes. Por cada paciente se registra su nombre, sexo, edad, altura, si es fumador o no, cuántas horas de ejercicio hace por semana, si tiene historia de enfermedades cardíacas en su familia, etc. Un algoritmo de selección de atributos eliminaría el atributo

del nombre del paciente, puesto que no contribuye en la predicción esperada, mientras que siguiendo alguna estrategia de discretización de atributos, el atributo de edad podría dividirse en intervalos de 15 o 20 años. Finalmente, un algoritmo de selección de instancias buscará eliminar de la base de datos a pacientes con información faltante, repetidos o con valores fuera de lo común.

Este estudio se centra en el problema de *Selección de Instancias*. En las siguientes secciones, el problema será definido formalmente y se describirán sus principales características.

1.2. Selección de Instancias

Dado un conjunto inicial de instancias $T = \{t_i \mid i = 1 \dots n\}$ y un conjunto de clases Ω , donde cada instancia está formada por una tupla de valores $t_i = (v_{i,1}, v_{i,2}, \dots, v_{i,m})$ y pertenece a la clase $\omega_{t_i} \in \Omega$, el problema de *Selección de Instancias* (SI) consiste en seleccionar un subconjunto $R \subseteq T$ que mantenga (o mejore) la capacidad de representación del conjunto original T .

Este problema puede ser formulado como un *problema de optimización combinatoria*, donde se busca el $R^* \subseteq T$ de menor cardinalidad que mantenga (o mejore) la capacidad de representación del conjunto original. Cada instancia perteneciente a T , puede o no pertenecer a R , lo que significa que el problema de SI tiene un espacio de posibles soluciones igual a 2^n .

En particular, la literatura se ha enfocado en la aplicación del problema de SI para su uso en clasificadores [GK14, Tou02]. El subconjunto seleccionado se usa como conjunto de entrenamiento, en base al cuál el clasificador estima la clase $\hat{\omega}$ de instancias previamente desconocidas. En este sentido, el problema de optimización de SI busca conseguir un $R^* \subseteq T$ *consistente* y de cardinalidad mínima, donde:

Definición 3. Un conjunto R es **consistente** con T , si y solo si toda instancia $t \in T$ es clasificada correctamente (*e.i.* $\hat{\omega}_t = \omega_t$) mediante el uso de un clasificador M y las instancias en R como conjunto de entrenamiento.

La complejidad del problema de selección de instancias ha sido estudiada por diferentes autores: *Bien y Tibshirani* [BT12] describen una reducción del problema de SI al problema de *Conjunto de Cobertura*, cuya versión de optimización es NP-Dura. Más aún, *Wilfong* [Wil91] y *Zukhba* [Zuk10] muestran que el problema de SI es NP-Duro.

En general, la literatura relacionada con el problema de SI se ha enfocado en el uso de clasificadores k -NN [GDCH12] por su simplicidad y sobretodo, por su capacidad de representación de modelos sin información adicional sobre la distribución de los datos. El caso particular del problema de SI usando clasificadores k -NN también es conocido como *Selección de Prototipos*. A continuación se describen estos clasificadores.

1.2.1. Regla del Vecino Más Cercano (NN)

Inicialmente descrita por *Fix y Hodges* [FH51], la regla del *Vecino Más Cercano* (Nearest Neighbor o NN) es una regla de inferencia basada en la idea de que instancias con atributos similares (cercanas en un espacio de m dimensiones) tienden a compartir la misma clase. La regla NN estima la clase $\hat{\omega}_x$ de un punto x en un espacio m -dimensional, dado un conjunto T de instancias de entrenamiento y una función de distancia φ entre dos puntos en dicho espacio:

$$\hat{\omega}_x = \omega_{t^*}, \quad t^* = \arg \min_{t \in T} \varphi(t, x) \quad (1.1)$$

La generalización de la regla de inferencia NN se conoce como el clasificador k -NN: dado un $k \in \mathbb{N}$, se estima la clase $\hat{\omega}_x$ de un punto x en función a la clase de las k instancias más cercanas a x . En general, se usa la estrategia del «*voto de la mayoría*», asignando la clase más común entre las k instancias más cercanas. En particular, el clasificador 1-NN corresponde a la regla NN.

k -NN es un clasificador no paramétrico de *aprendizaje perezoso* (debido a que la etapa de aprendizaje consiste en guardar el conjunto de entrenamiento), caracterizado por su sencillez en términos de implementación. Esta simplicidad y su probada utilidad para numerosas aplicaciones, han hecho del clasificador k -NN uno de los más estudiados en la literatura.

Uno de los trabajos de mayor relevancia es el de *Cover* y *Hart* [CH67], quienes mostraron que cuando el número de instancias de entrenamiento tiende a infinito, el clasificador k -NN garantiza un error no mayor al doble de la tasa de error de Bayes: la menor tasa de error posible para un clasificador dado. Adicionalmente, probaron que para un conjunto de entrenamiento de cardinalidad finita, el clasificador 1-NN es admisible dentro de la clase de clasificadores k -NN: *e.i.* No existe $k > 1$ tal que k -NN tenga menor probabilidad de error frente a 1-NN, para toda posible distribución de los datos.

Adicionalmente, algunos trabajos en geometría computacional han contribuido significativamente en la comprensión del problema. En este sentido, el clasificador 1-NN para espacios euclidianos puede definirse de forma alternativa en función de *Diagramas de Voronoi* [Vor08]: una partición del espacio \mathbb{R}^m en *Celdas de Voronoi*, cada una definida por una instancia $t \in T$ donde t es el *vecino más cercano* para todos los puntos dentro del espacio dentro de dicha celda (ver Figura 1.1). Esto ha permitido el desarrollo de nuevos enfoques para la búsqueda de vecinos más cercanos basados en *Diagramas de Voronoi*, como el descrito por *Kolahdouzan* y *Shahabi* [KS04].

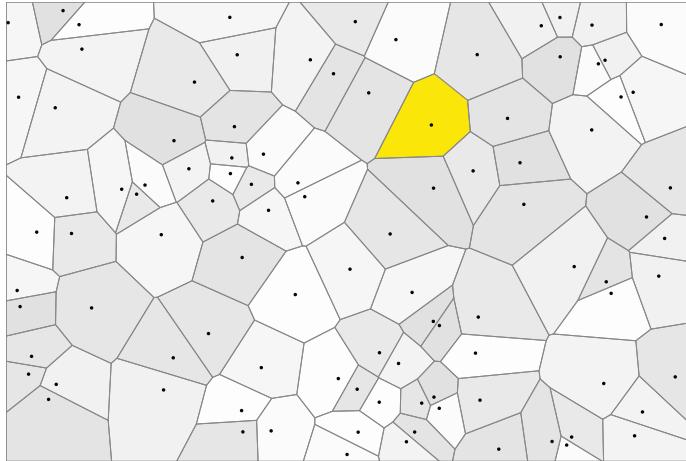


FIGURA 1.1: Diagrama de Voronoi para instancias en un espacio \mathbb{R}^2 . En amarillo la Celda de Voronoi de un punto $t \in T$, representando el espacio de puntos para los que t es su *vecino más cercano*.

Similarmente, esta relación ha permitido avances importantes en términos de complejidad. En particular, mediante el uso de *kd-trees* [Ben75] (árboles de búsqueda binaria en múltiples dimensiones) se ha logrado disminuir la complejidad en tiempo

de clasificación, de $\mathcal{O}(n)$ (de un enfoque “ingenuo” revisando todas las instancias) a $\mathcal{O}(\log n)$, a costas de un aumento en el tiempo necesario para el entrenamiento del clasificador: de $\mathcal{O}(1)$ a $\mathcal{O}(n \log n)$, el tiempo necesario para la construcción del árbol.

Sin embargo, los clasificadores k -NN presentan ciertas propiedades desalentadoras; el problema de conseguir el vecino más cercano de un punto dado, requiere –en cualquiera de los casos– almacenar todas las instancias de entrenamiento: *e.i.* $\mathcal{O}(n)$ en espacio. Adicionalmente, trabajos más recientes [KL04] muestran que en espacios euclidianos de altas dimensiones, la búsqueda del vecino más cercano requiere $\mathcal{O}(n)$ en tiempo: un fenómeno conocido como la «*maldición de la dimensionalidad*» (*curse of dimensionality* en inglés). Finalmente, según *Shwartz* y *David* [SSBD14] los clasificadores NN tienden a sobre-ajustar el modelo con respecto al conjunto de entrenamiento (*overfitting* en inglés); efecto que puede mitigarse aumentando el k del clasificador [DGKL94, SSBD14] y eliminando instancias del conjunto de datos [GKK13].

Algunas definiciones A continuación se definen algunos conceptos relevantes para la descripción de métodos heurísticos de SI en futuras secciones. Para un subconjunto de instancias cualquiera $Q \subseteq T$:

Definición 4. Los **asociados** en Q de una instancia t son aquellas instancias en Q para las cuales t pertenece a su conjunto de k instancias más cercanas:

$$\text{asociados}_Q(t) = \{q \in Q \mid t \in k\text{NN}(q)\} \quad (1.2)$$

Definición 5. Los **enemigos** en Q de una instancia $t \in T$ son aquellas instancias en Q con una *clase* diferente a la *clase* de t :

$$\text{enemigos}_Q(t) = \{q \in Q \mid \omega_q \neq \omega_t\} \quad (1.3)$$

Definición 6. El **enemigo más cercano** (*nearest enemy* o NE) en Q de una instancia $t \in T$, es la instancia más cercana a t con diferente clase, y es denotada como $\text{NE}_Q(t)$. *i.e.* $\text{NE}_Q(t)$ es la instancia más cercana a t perteneciente a $\text{enemigos}_Q(t)$:

$$\text{NE}_Q(t) = \arg \min_{e \in \text{enemigos}_Q(t)} \varphi(t, e) \quad (1.4)$$

1.3. Algoritmos de aproximación para Selección de Instancias

Debido a la complejidad del problema de SI, la literatura se ha enfocado en la definición de algoritmos heurísticos para conseguir soluciones aproximadas. De nuevo, el uso de clasificadores k -NN es una práctica extendida a lo largo de estos trabajos, por lo que las características de la regla NN han servido de inspiración para el desarrollo de la mayoría de los métodos de selección. Sin embargo, otras propuestas se basan en el orden de revisión de las instancias, así como en la realización de un muestreo aleatorio sobre el conjunto de datos. Finalmente, algunos enfoques se centran en aplicar procesos de búsqueda de propósito general, guiados por los objetivos del problema.

A continuación se describen brevemente algunos de los algoritmos heurísticos más recurrentes en la literatura relacionada al problema de SI.

1.3.1. Métodos basados en la regla NN

- *Condensed Nearest Neighbor* (CNN) [Har68]

Inicialmente el conjunto R se inicializa con una instancia cualquiera. Luego se itera sobre cada instancia $t \in T$; si t no es clasificada correctamente usando R , t se agrega a R . CNN consigue un conjunto consistente, reduciendo considerablemente el conjunto de datos original. Sin embargo, no asegura un conjunto consistente mínimo, pues depende del orden en el que son revisadas las instancias en T .

- *Edited Nearest Neighbor* (ENN) [Wil72]

Comienza con $R = T$. Luego itera sobre las instancias en R ; aquellas que no sean bien clasificadas usando R son eliminadas. Tiende a eliminar instancias ruidosas o cercanas a los bordes de decisión. Sin embargo, depende del orden en que itera sobre las instancias, y presenta bajas tasas de reducción dado que mantiene puntos internos.

- *Repeated Edited Nearest Neighbor* (RENN) [Wil72]

Aplica ENN al conjunto de datos R (inicialmente $R = T$) hasta que no ocurran cambios en R . Amplía la distancia entre clases y “suaviza” los bordes de decisión.

- *Reduced Nearest Neighbor* (RNN) [Gat72]

RNN extiende a CNN, usándola como solución inicial $R = R_{CNN}$. Luego, itera sobre cada instancia $t \in R$: si todas las instancias en T son correctamente clasificadas usando $R \setminus \{t\}$, se elimina t de R . En caso contrario, se mantiene R y continua la iteración. La precisión de RNN puede mejorar respecto a CNN, pero es más costoso y su consistencia depende de la consistencia del conjunto resultante de CNN y del orden en que se iteran las instancias en R .

1.3.2. Métodos basados en eliminación ordenada

- *Decremental Reduction Optimization Procedure 1* (DROP1) [WM97]

Comienza con una solución inicial $R = T$. Itera sobre cada instancia $t \in R$: si todos sus *asociados* en R son correctamente clasificados con $R \setminus \{t\}$, t se elimina de R . Reduce considerablemente el conjunto de datos inicial, pero obtiene baja precisión de clasificación, y el subconjunto resultante depende del orden en que se iteró sobre T .

- *Decremental Reduction Optimization Procedure 2* (DROP2) [WM97]

Es una mejora sobre DROP1 en la cuál se elimina una instancia t cuando todos sus *asociados* en T son clasificadas correctamente usando $R \setminus \{t\}$. Además, DROP2 ordena las instancias con respecto a la distancia de su *enemigo* más cercano, en un intento de eliminar primero instancias centrales, y luego los puntos en los bordes de decisión.

- *Decremental Reduction Optimization Procedure 3* (DROP3) [WM97]

Dado que el orden en que se iteran las instancias en DROP2 se ve alterado por puntos ruidosos, DROP3 filtra instancias ruidosas antes de ordenar el conjunto de entrenamiento.

1.3.3. Métodos basados en muestreo aleatorio

- *Random Mutation Hill Climbing* (RMHC) [Ska94]

Se selecciona un subconjunto de instancias aleatorias R de tamaño fijo. En cada iteración el algoritmo intercambia una instancia en R por una en $T \setminus R$; si el cambio mejora la precisión, se mantiene, en caso contrario se deshace.

1.3.4. Métodos basados en metaheurísticas

Las metaheurísticas son métodos de búsqueda estocástica de propósito general, usadas para encontrar soluciones óptimas o casi óptimas a problemas de optimización combinatoria. Por esta razón, muchos trabajos se han enfocado en el uso de estas técnicas para conseguir soluciones al problema de SI.

Algunos de los primeros trabajos se enfocaron en adaptar el algoritmo de *Búsqueda Tabú* para solucionar el problema de SI. En particular, los estudios de *Cerverón et al.* [CF01] y *Zhang et al.* [ZS02] describen dos enfoques diferentes de modificación del algoritmo.

Sin embargo, la mayoría de los estudios se han enfocado en el uso de *Algoritmos Evolutivos* (AE), adaptándolos para la búsqueda de soluciones al problema de selección. Entre ellos destaca el trabajo realizado por *Cano et al.* [CHL03]; un completo estudio comparativo entre algoritmos “tradicionales” de SI y adaptaciones del *Algoritmo Genético Generacional* (GGA), *Algoritmo Genético Estacionario* (SGA), *CHC Adaptive Search Algorithm* (CHC) y *Population-Based Incremental Learning* (PBIL). Con este estudio, resulta evidente la utilidad de los AE frente a los algoritmos tradicionales de SI en función de la capacidad de reducción y precisión de los conjuntos seleccionados.

Existen también otras adaptaciones y modificaciones sobre AE, entre los que destacan: *Estimation of Distribution Algorithm* (EDA) [SLI⁺01], *Intelligent Genetic Algorithm* (IGA) [HLL02], *Steady-State Memetic Algorithm* (SSMA) [GCH08] y *Genetic Algorithm* [GPY08] basado en Error Cuadrático Medio, *Clustered Crossover* y *Fast Smart Mutation* (GA-MSE-CC-PSM).

1.3.5. Criterios de comparación

Para comparar métodos de SI se consideran una serie de criterios usados para evaluar las ventajas y desventajas de cada algoritmo. A continuación se describen los factores más relevantes:

- *Reducción*: El objetivo principal de métodos de SI es el de reducir el número de instancias del conjunto de datos. Esto no solo disminuye el espacio necesario para almacenar los datos, sino que acelera el proceso de clasificación.
- *Precisión*: Un algoritmo exitoso debe reducir el conjunto de datos, afectando en la menor medida posible su capacidad de generalización.
- *Tiempo*: A pesar de que el proceso de preprocesamiento y aprendizaje debe realizarse solo una vez, la complejidad de los algoritmos pueden volverlos poco prácticos para su uso sobre conjuntos de datos “grandes”.

Capítulo 2

Metaheurísticas

2.1. Descripción general

Las metaheurísticas son métodos estocásticos de búsqueda de propósito general sobre espacios combinatorios. Son usados generalmente para tratar problemas de optimización combinatoria, donde su complejidad hace imposible evaluar todas las soluciones factibles en un tiempo razonable. Estos algoritmos son capaces de conseguir “buenas” soluciones a un problema en períodos de tiempo aceptables. Sin embargo, para muchos problemas la complejidad de estos algoritmos sigue siendo un factor prohibitivo, debido al uso de funciones “costosas” para la evaluación de soluciones intermedias.

La idea es desarrollar algoritmos que recorran solo una fracción del espacio de soluciones, y que sean capaces de encontrar buenas soluciones al problema en cuestión. Para lograrlo, las metaheurísticas combinan procesos de *diversificación* e *intensificación* (o *exploración* y *explotación* respectivamente) [Yan08]. La fase de *diversificación* implica la generación de soluciones distintas con el objeto de explorar el espacio de búsqueda, mientras que la fase de *intensificación* se refiere al mejoramiento de soluciones (conseguir óptimos locales) mediante el uso de métodos de búsqueda local. La selección de las mejores soluciones asegura la convergencia a soluciones óptimas, mientras que la exploración aleatoria de soluciones evita que el algoritmo quede “atrulado” en óptimos locales. La combinación en el uso de ambos procesos hace posible conseguir buenas soluciones al problema, sin la necesidad de recorrer el espacio de búsqueda completo.

Cada metaheurística está caracterizada por las estrategias que usa para cada fase, así como el orden y la frecuencia en que las aplica. Esto permite clasificarlas en función de su similitud. En este sentido, a continuación se describe un conjunto de metaheurísticas caracterizadas por tener a la naturaleza como fuente de inspiración.

2.2. Metaheurísticas inspiradas en la naturaleza

La habilidad de la naturaleza para moldear soluciones a situaciones complejas mediante procesos y reglas caracterizadas por su simplicidad, la ha convertido en una fuente inagotable de inspiración para el desarrollo de algoritmos de optimización. Estos algoritmos a menudo presentan buen desempeño para aproximar soluciones a todo tipo de problemas, dado que no requieren información sobre la distribución del espacio de búsqueda. Por esta razón, existe una amplia literatura sobre enfoques bio-inspirados [BS12] para resolver gran variedad de problemas en diversas áreas de computación.

En particular, los enfoques más comunes en la literatura sobre metaheurísticas inspiradas en la naturaleza se apoyan en *a)* la evolución de poblaciones (Algoritmos Evolutivos) y *b)* el comportamiento colectivo (Inteligencia de Enjambre).

2.2.1. Algoritmos Evolutivos

Los Algoritmos Evolutivos (AE) son metaheurísticas basadas en procesos de evolución biológica con el objetivo de explorar en amplitud espacios de solución con distribución desconocida. Con el fin de replicar los procesos evolutivos, los AE mantienen un conjunto de soluciones candidatas al problema (*una población de cromosomas/individuos*), que modifican iterativamente apoyándose en el uso de operadores de *mutación, recombinación y/o selección*.

Los AE codifican cada cromosoma como una cadena de genes de tamaño l (análogo a la estructura del ADN), donde cada gen representa una parte de la solución al problema en cuestión. A partir de esta representación, los AE definen un conjunto de operadores que cumplen la función de las estrategias de *exploración y explotación*:

- *Mutación*: Modifica los genes de soluciones intermedias con la finalidad de explorar el espacio de soluciones e introducir nueva información a la población. Simula la variabilidad en las poblaciones, fenómeno clave para la aparición de nuevos genes que aumenten la posibilidad de supervivencia.
- *Recombinación/Cruce*: Permite el intercambio de información entre individuos de la población. Simula la reproducción entre individuos, necesaria para la transmisión de genes relevantes a las siguientes generaciones.
- *Selección*: Las estrategias de selección permiten definir aquellos individuos que participarán en la fase de reproducción, y por ende, los genes que pasarán a la siguiente generación. Esto simula el proceso de selección natural en el que sobreviven los individuos mejor adaptados al ambiente.

En la literatura se han desarrollado diferentes esquemas que definen el uso de estos operadores. Los AE más “tradicionales” son conocidos como *Algoritmos Genéticos* (AG) [Hol75], que suponen la aplicación más directa de los conceptos del proceso evolutivo. Sin embargo, dentro de la clase de AE existe otro grupo de algoritmos que aplican dichos conceptos de forma diferente. La clase de *Algoritmos de Estimación de Distribución* (*Estimation of Distribution Algorithm* o EDA) aplican los operadores de *mutación*, *recombinación* y *selección* sobre una población de soluciones implícita en un modelo de distribución probabilístico.

A continuación se describen cuatro algoritmos pertenecientes a la clase de AE: GGA, SGA y CHC, variantes del grupo de AG, y PBIL, perteneciente a los EDA.

2.2.1.1. Algoritmo Genético Generacional (GGA)

GGA (*Generational Genetic Algorithm*) es el esquema “tradicional” de aplicación de los AG [Bac96, Muh91]. Mantiene una población de individuos que evolucionan durante un número de iteraciones. Su principal característica es que en cada iteración se genera una nueva población, *i.e.* un proceso de evolución *generacional*.

En cada iteración el proceso evolutivo consiste en la creación de una nueva población de tamaño pop mediante: *a)* la selección de los individuos para el proceso

de reproducción (*padres*), *b*) la recombinación (con probabilidad *cp*) de pares de individuos *padres* usando una estrategia particular de cruce, y *c*) la mutación de los individuos de la nueva población (llamados *descendencia*), usando una probabilidad de mutación de cada gen igual a *mp*. Ver el algoritmo 2.1.

Algoritmo 2.1 Algoritmo Genético Generacional

Input: *pop* tamaño de la población, *cp* probabilidad de cruce, *mp* probabilidad de mutación

Output: Una solución al problema

```

1:  $P \leftarrow$  Generar población aleatoria de pop individuos
2:  $s^* \leftarrow$  el mejor individuo en  $P$ 
3: while  $\neg$  Condición de parada do
4:    $P' \leftarrow \emptyset$ 
5:   while  $|P'| < pop$  do
6:      $p_1 \leftarrow$  Seleccionar un individuo en  $P$ 
7:      $p_2 \leftarrow$  Seleccionar un individuo en  $P$ 
8:      $c_1, c_2 \leftarrow$  recombinar  $p_1$  y  $p_2$  con probabilidad cp
9:     Mutar  $c_1$  y  $c_2$  con probabilidad mp
10:     $P' \leftarrow P' \cup \{c_1, c_2\}$ 
11:     $P \leftarrow P'$ 
12:    if El mejor individuo en  $P$  es mejor que  $s^*$  then
13:       $s^* \leftarrow$  el mejor individuo en  $P$ 
14: return  $s^*$ 
```

2.2.1.2. Algoritmo Genético Estacionario (SGA)

Descrito por *Whitley et al.* [WK88], SGA (*Steady-State Genetic Algorithm*) es una modificación del esquema general de AG que sigue una estrategia reproductiva no generacional. SGA comienza con una población de tamaño *pop*, y en cada iteración se producen un máximo de dos nuevos individuos (no una nueva población).

En cada iteración *a*) se seleccionan dos individuos padres de la población actual, *b*) se crea su descendencia (con probabilidad *cp*) mediante algún método de cruce/-recombinación, *c*) se agrega variabilidad mediante la mutación (con probabilidad *mp*) de la nueva descendencia, y *d*) se sigue alguna estrategia de selección para reemplazar individuos en la población por la nueva descendencia, y así mantener el tamaño de la población igual a *pop*. Ver el algoritmo 2.2.

Algoritmo 2.2 Algoritmo Genético Estacionario

Input: pop tamaño de la población, cp probabilidad de cruce, mp probabilidad de mutación

Output: Una solución al problema

- 1: $P \leftarrow$ Generar población aleatoria de pop individuos
 - 2: $s^* \leftarrow$ el *mejor* individuo en P
 - 3: **while** \neg Condición de parada **do**
 - 4: $p_1 \leftarrow$ Seleccionar un individuo en P
 - 5: $p_2 \leftarrow$ Seleccionar un individuo en P
 - 6: $c_1, c_2 \leftarrow$ recombinar p_1 y p_2 con probabilidad cp
 - 7: Mutar c_1 y c_2 con probabilidad mp
 - 8: Seguir algún criterio de reemplazo de individuos en P por c_1 y c_2
 - 9: **if** El *mejor* individuo en P es *mejor* que s^* **then**
 - 10: $s^* \leftarrow$ el *mejor* individuo en P
 - 11: **return** s^*
-

Frente a un esquema generacional como el descrito por GGA, el tiempo de ejecución reportado por SGA es mucho menor. Esto se debe a que en cada iteración, SGA genera un máximo de 2 nuevos individuos en vez de una población completamente nueva.

2.2.1.3. CHC Adaptive Search Algorithm

CHC [Esh90] se basa en el esquema de evolución generacional aplicado por GGA: mantiene una población de individuos de tamaño fijo (pop), generando una nueva población en cada iteración. Sin embargo, en cada iteración CHC aplica una estrategia de reemplazo “elitista”, donde sobreviven los mejores individuos entre la población actual y la descendencia producida.

La fase de reproducción aplicada por CHC tiene dos particularidades. En primer lugar, implementa un operador de recombinación uniforme media llamado HUX (*Half Uniform Crossover*) [Esh90], que intercambia la mitad de los genes que difieren entre los dos padres de forma aleatoria. Adicionalmente, CHC emplea “prevención de incesto”: antes de realizar el cruce usando HUX, calcula la *distancia de Hamming* entre ambos padres; si dicha distancia es mayor a cierto umbral (inicialmente $l/4$, donde l es la longitud de los cromosomas), se realiza el cruce. En caso de no generarse ninguna descendencia durante una iteración particular, se disminuye el umbral en 1.

Durante el proceso de evolución de CHC no se aplica el operador de mutación: cuando el umbral de prevención de incesto llega a cero se considera que la población convergió, y comienza un proceso de repoblación en el que se usa la mejor solución encontrada hasta el momento. Se modifican hasta 35 % de sus genes de forma aleatoria para generar los $\text{pop} - 1$ individuos restantes de la nueva población, y luego continuar el proceso evolutivo.

El pseudocódigo de CHC de presenta en el algoritmo 2.3.

Algoritmo 2.3 CHC Adaptive Search Algorithm

Input: pop tamaño de la población

Output: Una solución al problema

```

1:  $P \leftarrow$  Generar población aleatoria de  $\text{pop}$  individuos
2:  $s^* \leftarrow$  el mejor individuo en  $P$ 
3:  $\mu \leftarrow l/4$                                       $\triangleright$  Umbral de cruce
4: while  $\neg$  Condición de parada do
5:   for  $i \in [1 \dots \text{pop}/2]$  do
6:      $p_1 \leftarrow$  Seleccionar un individuo en  $P$ 
7:      $p_2 \leftarrow$  Seleccionar un individuo en  $P$ 
8:     if  $\text{hamming}(p_1, p_2) > \mu$  then
9:        $c_1, c_2 \leftarrow$  recombinar  $p_1$  y  $p_2$  usando HUX
10:       $P \leftarrow P \cup \{c_1, c_2\}$ 
11:    if  $|P| = \text{pop}$  then
12:       $\mu \leftarrow \mu - 1$ 
13:      if  $\mu = 0$  then
14:         $P \leftarrow$  Generar población de  $\text{pop}$  individuos usando  $s^*$ 
15:         $\mu \leftarrow l/4$ 
16:    else
17:       $P \leftarrow$   $\text{pop}$  mejores individuos en  $P$ 
18:      if El mejor individuo en  $P$  es mejor que  $s^*$  then
19:         $s^* \leftarrow$  el mejor individuo en  $P$ 
20: return  $s^*$ 
```

2.2.1.4. Population-Based Incremental Learning (PBIL)

PBIL es una metaheurística perteneciente a la clase de *Algoritmos de Estimación de Distribución* desarrollada por Baluja [Bal94] para su uso sobre cromosomas con representación binaria. PBIL destaca por ser más simple que los algoritmos genéticos

tradicionales y por lograr mejores soluciones para gran variedad de problemas [Bal95, BC95].

Este algoritmo mantiene una población *implícita* de soluciones, mediante el uso de un vector de probabilidades V de tamaño l , donde V_i (con $i \in [1 \dots l]$) es la probabilidad que el i -esimo bit/gen de una solución en la población esté “prendido” (sea igual a 1). PBIL usa este vector de probabilidades para generar poblaciones de tamaño pop en cada iteración, y guiar el proceso evolutivo en base a las soluciones generadas.

Inicialmente $V_i = 0.5 \quad \forall i \in [1 \dots l]$. Luego en cada iteración: *a)* se generan pop cromosomas binarios basados en las probabilidades en V , *b)* se “acerca” V hacia la mejor solución generada (usando una tasa de aprendizaje lr), *c)* se “aleja” V de la peor solución generada (usando una tasa de aprendizaje negativa nlr), *d)* se sigue una estrategia de mutación sobre V en la que se aumenta o disminuye V_i en ms (*mutation shift*) con probabilidad de mutación mp . Ver algoritmo 2.4.

Algoritmo 2.4 Population-Based Incremental Learning

Input: pop tamaño de la población, mp probabilidad de mutación, ms mutation shift, lr learning rate, nlr negative learning rate

Output: Una solución al problema

```

1:  $V \leftarrow$  Vector de probabilidades de tamaño  $l$ 
2:  $s^* \leftarrow$  Una solución cualquiera
3: while  $\neg$  Condición de parada do
4:    $P \leftarrow$  Generar población de tamaño  $\text{pop}$  según las probabilidades en  $V$ 
5:    $b \leftarrow$  El mejor individuo en  $P$ 
6:    $w \leftarrow$  El peor individuo en  $P$ 
7:   if  $b$  es mejor que  $s^*$  then
8:      $s^* \leftarrow b$ 
9:   for  $i \in [1 \dots l]$  do            $\triangleright$  Actualizar el vector de probabilidades
10:     $V_i \leftarrow V_i * (1 - \text{lr}) + b_i * \text{lr}$ 
11:    if  $b_i \neq w_i$  then
12:       $V_i \leftarrow V_i * (1 - \text{nlr}) + b_i * \text{nlr}$ 
13:    if  $\text{Unif}(0, 1) < \text{mp}$  then           $\triangleright$  Mutación con probabilidad  $\text{mp}$ 
14:       $V_i \leftarrow V_i * (1 - \text{ms}) + \text{UnifDiscreta}(0, 1) * \text{ms}$ 
15: return  $s^*$ 
```

2.2.2. Inteligencia de Enjambre

Inteligencia de Enjambre [BDT99] (IE) es un paradigma emergente entre los sistemas de cómputo bio-inspirados. Surge como una extensión de los AE, pero no se basa en la adaptación genética de poblaciones, sino en el comportamiento colectivo de grupos de organismos. Las estrategias de IE exhiben patrones de búsqueda descentralizada y auto-organizada, mediante la simulación de la inteligencia colectiva de grupos de “agentes” sencillos.

Durante la última década se han desarrollado numerosos enfoques basados en la explotación de inteligencia colectiva, inspirados en el comportamiento de colonias de hormigas, abejas, luciérnagas, etc. Uno de los más estudiados se conoce como PSO, inspirado en el vuelo de grupos de aves.

2.2.2.1. Particle Swarm Optimization (PSO)

PSO [KE95] se inspira en el comportamiento de organismos biológicos, en particular del vuelo de una bandada. Cada ave o “partícula” representa una solución que se mueve en el espacio de soluciones del problema, y modifica su “vuelo” en relación a su propia experiencia y la de sus “compañeras”. Diferentes estudios [SE98, KS98] muestran que PSO obtiene mejores resultados que los algoritmos genéticos y en menor tiempo de cómputo.

La i -esima partícula de PSO tiene asociado *a*) un vector de posición $x_i \in \mathbb{R}^l$ en un espacio euclíadiano l -dimensional que representa una solución al problema, y *b*) un vector de velocidad $v_i \in \mathbb{R}^l$ que modifica la posición de la partícula en cada iteración. Ver algoritmo 2.5.

El vector de velocidad se modifica en función de varios parámetros: *a*) el peso de inercia $w \in \mathbb{R}$ que evita cambios bruscos respecto a la velocidad anterior, *b*) el peso del mejor local $c_1 \in \mathbb{R}$ que indica la importancia de la mejor solución encontrada por dicha partícula, y *c*) el peso del mejor global $c_2 \in \mathbb{R}$ que establece la importancia de la mejor solución global encontrada hasta el momento. Estos parámetros se encargan de dar dirección a la búsqueda de cada partícula. Usualmente los valores del vector

Algoritmo 2.5 Particle Swarm Optimization

Input: part número de partículas, vmax velocidad máxima, w peso de inercia, c1 peso del mejor local, c2 peso del mejor global

Output: Una solución al problema

```

1: for  $i \in [1 \dots \text{part}]$  do
2:    $\vec{X}_i \leftarrow$  Solución inicial aleatoria  $\in \mathbb{R}^l$ 
3:    $\vec{V}_i \leftarrow$  Vector de velocidades aleatorias entre  $[-\text{vmax}, \text{vmax}]$ 
4:    $p_i \leftarrow \vec{X}_i$                                  $\triangleright$  La mejor solución encontrada por la particula  $i$ 
5:    $s^* \leftarrow$  La mejor solución  $p_i, i \in [1 \dots \text{part}]$ 
6: while  $\neg$  Condición de parada do
7:   for  $i \in [1 \dots \text{part}]$  do
8:      $\vec{X}_i \leftarrow \vec{X}_i + \vec{V}_i$ 
9:      $\vec{V}_i \leftarrow w\vec{V}_i + c1 \text{ Unif}(0, 1)(p_i - \vec{X}_i) + c2 \text{ Unif}(0, 1)(s^* - \vec{X}_i)$ 
10:    Limitar valores en  $\vec{V}_i$  entre  $[-\text{vmax}, \text{vmax}]$ 
11:    if  $\vec{X}_i$  es mejor que  $p_i$  then
12:       $p_i \leftarrow \vec{X}_i$ 
13:      if  $p_i$  es mejor que  $s^*$  then
14:         $s^* \leftarrow p_i$ 
15: return  $s^*$ 
```

de velocidad se limitan a ciertos rangos para permitir la explotación de soluciones locales: se usa el parámetro $\text{vmax} \in \mathbb{R}$ para limitar la velocidad entre $[-\text{vmax}, \text{vmax}]$.

A pesar de estar definido para encontrar soluciones con representación en \mathbb{R}^l , PSO puede modificarse para optimizar soluciones con representaciones variadas, definiendo apropiadamente los operadores usados para la actualización de la velocidad. Otro enfoque radica en mapear el espacio de búsqueda del problema a un dominio continuo: en el caso de soluciones con representación binaria se puede limitar la posición de las partículas entre $[0, 1]$ y obtener soluciones redondeando dichos valores.

Capítulo 3

Adaptación al problema de Selección de Instancias

Al tratarse de métodos de búsqueda de propósito general, las metaheurísticas son capaces de encontrar soluciones a todo tipo de problemas de optimización combinatoria. Para ello debe definirse una representación que codifique las posibles soluciones al problema y una función de evaluación que permita evaluar dichas soluciones en función a los objetivos de optimización del problema.

A continuación se describen éstas y otras consideraciones generales para la aplicación de metaheurísticas al problema de *Selección de Instancias*. Adicionalmente se plantea para cada metaheurística, las estrategias a usar durante el proceso de búsqueda, así como algunas modificaciones particulares.

3.1. Consideraciones generales

3.1.1. Representación

Una solución cualquiera al problema de SI está dada por un subconjunto de instancias R del conjunto inicial T (*i.e.* $R \subseteq T$). Por lo tanto, para un orden dado de las instancias $t_i \in T$ ($i = 1 \dots n$, con $n = |T|$), una codificación usando mapas de bits es suficiente para representar el espacio de soluciones al problema.

En este sentido, una solución particular al problema de SI está descrita por una cadena de bits s de tamaño n . Cada bit s_i ($i = 1 \dots n$) tiene valor 1/0 (“prendido” o “apagado” respectivamente), lo que denota la pertenencia o no de la instancia correspondiente $t_i \in T$ al subconjunto seleccionado. $R_s \subseteq T$ es el subconjunto seleccionado representado por la cadena de bits s , donde:

$$R_s = \{t_i \in T \mid i = 1 \dots n \wedge s_i = 1\} \quad (3.1)$$

Este conjunto R_s permite evaluar la “bondad” de la solución s en función de los objetivos de optimización del problema.

3.1.2. Función de evaluación

El objeto de la aplicación de metaheurísticas es conseguir soluciones óptimas o casi óptimas a variedad de problemas. Para ello es necesario definir una estrategia de comparación entre soluciones, *i.e.* saber cuándo una solución es «*mejor*» que otra, que permita seleccionar aquellas soluciones que mejor se adapten a los objetivos del problema en cuestión.

Para el caso del problema de SI, el objetivo es encontrar un subconjunto R del menor tamaño posible, que mantenga un alto porcentaje de precisión de clasificación. *Cano et al.* [CHL03] definen una función de evaluación que combina ambos objetivos en función de un parámetro $\alpha \in [0, 1]$; a continuación se presenta una modificación de dicha función (su complemento) usada en el presente trabajo:

$$eval(R) = \alpha \ error(R) + (1 - \alpha)|R| \quad (3.2)$$

Donde $error(R)$ es el número de instancias en T que son erróneamente clasificadas usando el conjunto $R \subseteq T$ como conjunto de entrenamiento de un clasificador 1-NN. El parámetro α combina los objetivos de la búsqueda: se usa $\alpha = 0.5$ siguiendo lo descrito por [CHL03] para obtener soluciones que satisfagan ambos objetivos del problema.

Para esta definición, el objetivo de las metaheurísticas para selección de instancias es *minimizar* la función de evaluación descrita. Para ello, deben minimizar ambos objetivos de la función: el error de clasificación usando el subconjunto seleccionado y el tamaño de dicho subconjunto. En este sentido, dadas dos posibles soluciones a y b , a es «*mejor*» que b si y solo si $\text{eval}(R_a) < \text{eval}(R_b)$.

3.1.3. Generación de soluciones iniciales

Para muchos problemas con esquemas de representación binaria, la generación de soluciones iniciales usada por metaheurísticas sigue una estrategia común: en una solución aleatoria codificada como una cadena de bits, cada bit tiene 50 % de probabilidad de estar prendido ($\delta = 50\%$). Esta estrategia genera soluciones con valor esperado de la mitad de los bits prendidos. Para el problema de SI esto implica soluciones con una reducción inicial del 50 % sobre el conjunto T , y hace necesario un alto número de iteraciones para lograr obtener soluciones con reducciones significativas [CHL03]. Por esta razón, en el presente trabajo se propone la disminución del parámetro δ al 5 % de probabilidad para cada bit, generando soluciones iniciales con reducciones cercanas al 95 %. Adicionalmente, se evalúa el impacto de una disminución menos drástica, con un valor de $\delta = 25\%$. En este sentido, rol de la función objetivo no recae en disminuir los porcentajes de reducción de las soluciones, sino en mantenerlos en niveles aceptables, permitiendo explotar el espacio de búsqueda para conseguir soluciones con mayor precisión.

Sin embargo, el punto inicial de la búsqueda en términos de precisión sigue siendo aleatorio. Una estrategia común en metaheurísticas de trayectoria es la generación de soluciones iniciales usando algoritmos de aproximación; *Cerveron et al.* [CF01] plantean una modificación del algoritmo de Búsqueda Tabú para el problema de SI, en la cuál usan CNN para generar la solución inicial de la búsqueda. Esta idea puede trasladarse a metaheurísticas poblacionales siguiendo un enfoque probabilístico: dada una selección inicial $R_0 \subseteq T$, los bits correspondientes a instancias en R_0 tienen mayor probabilidad de aparición que los bits de instancias en $T \setminus R_0$. Estas probabilidades constituyen un vector de probabilidades a ser usado por PBIL como vector inicial, o por los algoritmos genéticos (GGA, SGA y CHC) para generar soluciones iniciales.

El algoritmo 3.1 implementa este esquema de generación; para un δ particular (que determina el número de bits prendidos en las soluciones iniciales), el vector de probabilidades V generado por el algoritmo cumple con que un máximo del 70 % de los bits prendidos en soluciones generadas usando V , pertenecen a la solución inicial R_0 . Esto contribuye a guiar la búsqueda realizada por las metaheurísticas poblacionales, manteniendo la variabilidad.

Algoritmo 3.1 Generador de vector de probabilidades inicial

Input: R_0 solución inicial

Output: Vector de probabilidades en base a R_0

```

1:  $high \leftarrow \min(0.9, \frac{\delta|T|^{0.7}}{|R_0|})$ 
2:  $low \leftarrow \frac{\delta|T|-high|R_0|}{|T \setminus R_0|}$ 
3:  $V \leftarrow$  Vector de probabilidades de tamaño n
4: for  $i \in [1 \dots n]$  do
5:   if  $t_i \in R_0$  then
6:      $V_i \leftarrow high$ 
7:   else
8:      $V_i \leftarrow low$ 
9: return  $V$ 
```

Cualquier solución generada por algoritmos de aproximación para el problema de SI (sección 1.3) sirve como “semilla” de este generador. En el presente trabajo se prueba el impacto del uso de soluciones iniciales calculadas por CNN. También se incluyen dos selecciones basadas en el orden según el *enemigo más cercano* (NE - “*Nearest Enemy*”), seleccionando las instancias con mayor distancia NE (selección por *Farthest NE*), o menor distancia NE (selección por *Closest NE*). Finalmente, se prueba un método de selección propio llamado *Nearest Enemy Hypersphere Selection* (NEHS). En la figura 3.2 se pueden observar los subconjuntos seleccionados por estos cuatro (4) algoritmos.

3.1.3.1. Condensed Nearest Neighbor

En 1968 *Hart* [Har68] fue el primero en proponer un método de reducción de instancias a ser usadas por la regla NN; este método se conoce como *Condensed Nearest Neighbor* (CNN). El objetivo de CNN es realizar un proceso de selección previo al entrenamiento del clasificador 1-NN, consiguiendo un conjunto $R \subseteq T$ que

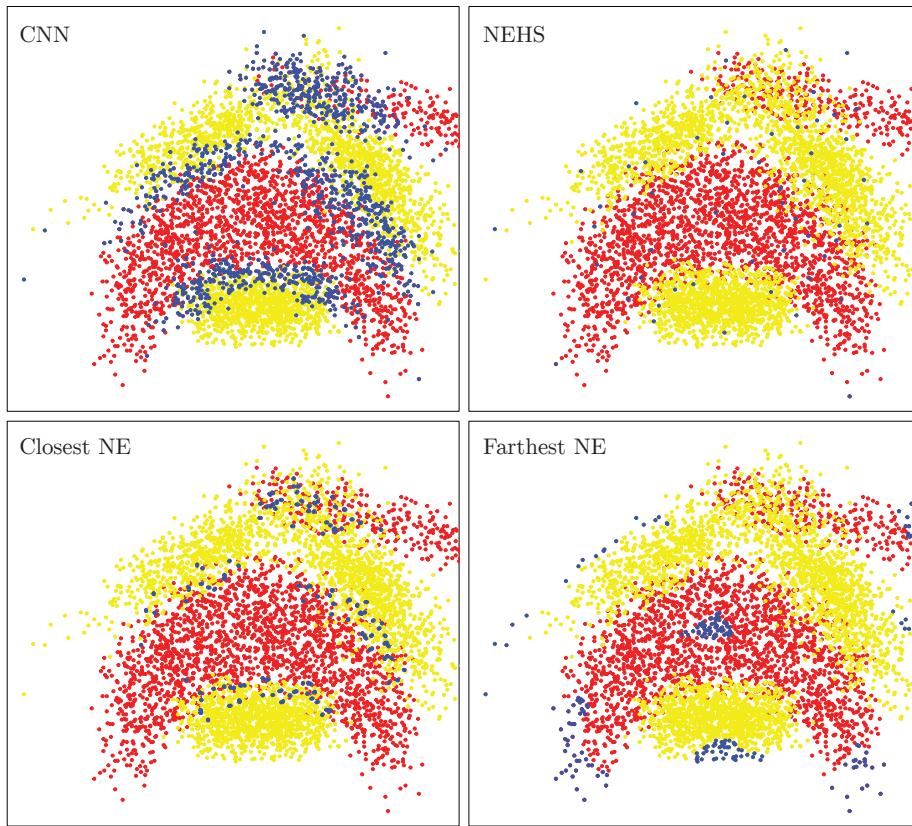


FIGURA 3.1: Subconjunto de instancias seleccionadas por CNN, NEHS, Closest NE y Farthest NE, para el conjunto de instancias BANANA. Las instancias están pintadas en amarillo y rojo según su clase en el conjunto de datos, y en azul aquellas instancias pertenecientes a la selección correspondiente.

mantenga la efectividad del conjunto de datos original T para clasificar instancias desconocidas.

Se trata de un algoritmo incremental en el que se incluyen en R aquellas instancias en T que son mal clasificadas usando el conjunto reducido como conjunto de entrenamiento en un clasificador 1-NN. Dicho proceso se repite hasta que no se incluyan nuevas instancias en el conjunto R luego de una iteración completa sobre las instancias en T ; ver algoritmo 3.2.

CNN logra una reducción considerable sobre el conjunto de datos original, y asegura un conjunto consistente con T . Sin embargo, al ser dependiente del orden de revisión de las instancias, CNN no asegura un conjunto consistente *mínimo*. Este algoritmo tiende

Algoritmo 3.2 Condensed Nearest Neighbor

Input: T conjunto de instancias inicial**Output:** Conjunto de instancias $R \subseteq T$

```

1:  $R \leftarrow \{\text{Una instancia cualquiera } t \in T\}$ 
2: repeat
3:    $R' \leftarrow R$ 
4:   for all  $t \in T$  do
5:     if  $t$  es mal clasificada usando  $R$  con un clasificador 1-NN then
6:        $R \leftarrow R \cup \{t\}$ 
7: until  $R = R'$ 
8: return  $R$ 

```

a seleccionar instancias cercanas a los bordes de decisión, viéndose particularmente afectado por datos ruidosos [AR11, JG].

3.1.3.2. Nearest Enemy Hypersphere Selection

La idea tras el uso de la regla NN es que instancias “cercanas” en un espacio cualquiera (*i.e.* con atributos similares) comparten su clasificación; esto conlleva a la división de dicho espacio en regiones con instancias de igual clase. La idea fundamental de *Nearest Enemy Hypersphere Selection* (NEHS) es seleccionar instancias en los centros de dichas regiones, con la finalidad de reducir la cantidad de instancias necesarias para generalizar el espacio descrito por el conjunto original.

Para esto hace uso de la distancia del *enemigo más cercano*: cada instancia $t \in T$ tiene un enemigo más cercano $\text{NE}(t) \in T$, cuya distancia $\varphi(t, \text{NE}(t))$ define el radio de una hiperesfera en el espacio m -dimensional de atributos con centro en t , dentro de la cual toda instancia comparte la clase ω_t .

NEHS implementa una estrategia *greedy* que busca seleccionar las mayores hiperesferas que no intersecten entre sí. Realiza la selección de hiperesferas comenzando con las de mayor tamaño, con el objetivo de cubrir el mayor espacio posible (ver el algoritmo 3.3). Sin embargo, para evitar la selección de puntos cercanos a los bordes de decisión (y posiblemente ruidosos), NEHS ignora un porcentaje Δ de instancias en T con menor distancia NE, *i.e.* las hiperesferas de menor radio. Esto implica una

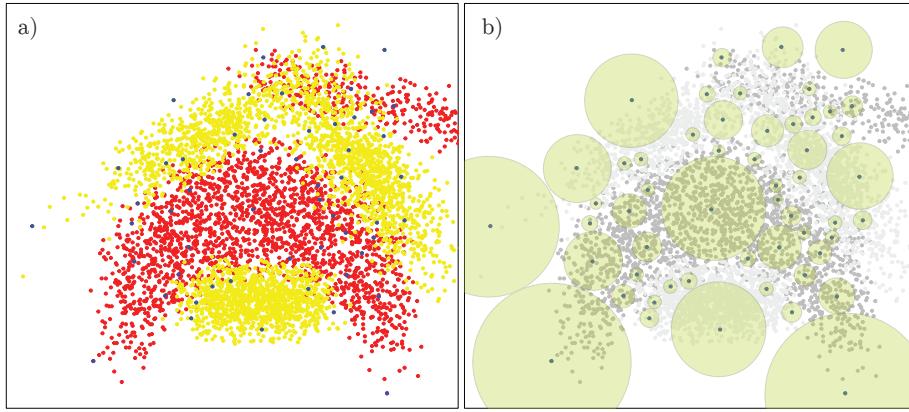


FIGURA 3.2: a) Subconjunto de instancias seleccionadas por NEHS, e b) Hiperesferas correspondientes a la selección de NEHS, con radio en función a la distancia del enemigo más cercano.

selección con bordes de decisión mucho más “suaves” que los del conjunto original. El presente trabajo usa un $\Delta = 33\%$.

Algoritmo 3.3 Nearest Enemy Hypersphere Selection

Input: T conjunto de instancias inicial, Δ porcentaje de instancias a excluir

Output: Conjunto de instancias $R \subseteq T$

- 1: $R \leftarrow \emptyset$
 - 2: **for all** $t \in T$ en orden descendiente de distancia NE
excluyendo las últimas $\Delta|T|$ instancias **do**
 - 3: **if** $\neg\exists r \in R$ tal que $\varphi(t, r) < \varphi(r, \text{NE}(r)) + \varphi(t, \text{NE}(t))$ **then**
 - 4: $R \leftarrow R \cup \{t\}$
 - 5: **return** R
-

3.1.3.3. Closest Nearest Enemy

La distancia del *enemigo más cercano* (NE) de una instancia cualquiera $t \in T$ (*i.e.* $\varphi(t, \text{NE}(t))$), indica su cercanía a los bordes de decisión establecidos mediante un clasificador 1-NN. La selección por *Closest NE* escoge aquellas instancias con *menor* distancia NE entre las instancias en T , *i.e.* instancias pertenecientes –o cercanas– a los bordes de decisión de los datos. En el presente trabajo, *Closest NE* selecciona el porcentaje $\delta = 5\%$ de instancias con menor distancia NE.

$$R_{\text{ClosestNE}} = \{\delta|T| \text{ instancias con menor distancia NE}\} \quad (3.3)$$

3.1.3.4. Farthest Nearest Enemy

Similar a la selección por *Closest NE*, *Farthest NE* se basa en el uso de la distancia NE para incluir instancias en el conjunto reducido $R \subseteq T$. Sin embargo, *Farthest NE* selecciona aquellas instancias con *mayor* distancia NE, *i.e.* instancias alejadas de los bordes de decisión y cercanas a los centros de sus respectivas regiones. Este método también usa un porcentaje de selección igual a $\delta = 5\%$ de las instancias en T .

$$R_{\text{FarthestNE}} = \{\delta|T| \text{ instancias con mayor distancia NE}\} \quad (3.4)$$

3.2. Modificaciones particulares

3.2.1. Adaptación de GGA, SGA y CHC

Para aplicar algún algoritmo genético al problema de SI, es necesario describir las estrategias de selección, recombinación, mutación y reemplazo que seguirá el algoritmo durante el proceso evolutivo.

GGA, SGA y CHC requieren de un operador que seleccione individuos de la población para participar en el proceso reproductivo. Se emplea el método de *selección por torneo* en su versión “elitista”: se escoge al azar un conjunto de individuos entre la población original, y es seleccionado el más apto entre ellos. El tamaño de dicho conjunto debe ser pequeño; generalmente se usa un “tamaño de torneo” entre 2 y 5 [MG95]. En este trabajo se usan torneos de tamaño 3.

Debido a que CHC aplica un operador de recombinación particular (HUX) y no requiere de un operador de mutación, dichos operadores deben ser definidos solo para GGA y SGA. El operador de *crossover* aplicado en ambos algoritmos es el de *recombinación en un punto*. Dados dos cromosomas “padres”, se elige aleatoriamente un

punto de corte en la longitud de ambos cromosomas; los cromosomas “hijos” resultan de combinar la sección izquierda del corte de un padre, con la sección derecha del corte del otro padre. El operador de mutación sigue –en ambos casos– el esquema estándar de modificación de cada bit con una cierta probabilidad.

Finalmente, es necesario describir los criterios de reemplazo. Al tratarse de estrategias evolutivas generacionales, GGA y CHC reemplazan la población de forma incondicional en favor de la descendencia. En cambio, el criterio de reemplazo usado en SGA es “elitista”: se sustituyen los padres por la descendencia generada, solo si dicha descendencia es mejor.

3.2.2. Adaptación de PBIL

La estrategia evolutiva de PBIL no requiere modificaciones adicionales para su aplicación al problema de selección de instancias. PBIL está diseñado para encontrar soluciones con representación binaria, lo que permite su aplicación directa al problema. La única modificación realizada al esquema estándar de PBIL es en el método de generación del vector de probabilidades iniciales, reflejando lo descrito en la sección 3.1.3.

3.2.3. Adaptación de PSO

La adaptación de PSO al problema de SI es mayor, debido a que PSO está pensado para problemas con representación en espacios euclidianos, *i.e.* es necesario el mapeo de vectores en \mathbb{R}^l a soluciones con representación binaria. En este sentido, se adopta el esquema poblacional de PBIL en el que el genotipo de la población se representa de forma probabilística.

A esta adaptación de PSO la llamaremos *Population-Based PSO* (ver algoritmo 3.4), en la cuál el vector de posición \vec{X}_i de cada partícula es un vector de probabilidades que representa el genotipo de una población particular. Esto permite la generación de soluciones en codificación binaria para su respectiva evaluación, en base al vector de probabilidades que describe dicha población. A diferencia del PSO tradicional, la

selección de óptimos locales y globales se hace en base a las soluciones generadas mediante \vec{X}_i , y no en base al vector \vec{X}_i por si solo. Sin embargo, la modificación de los vectores de probabilidad y velocidad (\vec{X}_i y \vec{V}_i respectivamente) sigue la estrategia de actualización estándar en base a la mejor solución local y la mejor solución global.

Algoritmo 3.4 Population-Based PSO

Input: pop tamaño de la población, part número de partículas, vmax velocidad máxima, w peso de inercia, c_1 peso del mejor local, c_2 peso del mejor global

Output: Una solución al problema

```

1: for  $i \in [1 \dots \text{part}]$  do
2:    $\vec{X}_i \leftarrow$  Vector de probabilidades de tamaño  $l$ 
3:    $\vec{V}_i \leftarrow$  Vector de velocidades aleatorias entre  $[-\text{vmax}, \text{vmax}]$ 
4:    $p_i \leftarrow$  Generar una solución a partir de  $\vec{X}_i$ 
5:    $s^* \leftarrow$  La mejor solución  $p_i, i \in [1 \dots \text{part}]$ 
6: while  $\neg$  Condición de parada do
7:   for  $i \in [1 \dots \text{part}]$  do
8:      $\vec{X}_i \leftarrow \vec{X}_i + \vec{V}_i$ 
9:     Limitar valores en  $\vec{X}_i$  entre  $[0, 1]$ 
10:     $\vec{V}_i \leftarrow w\vec{V}_i + c_1 \text{Unif}(0, 1)(p_i - \vec{X}_i) + c_2 \text{Unif}(0, 1)(s^* - \vec{X}_i)$ 
11:    Limitar valores en  $\vec{V}_i$  entre  $[-\text{vmax}, \text{vmax}]$ 
12:     $P \leftarrow$  Generar población de tamaño  $\text{pop}$  a partir de  $\vec{X}_i$ 
13:    if El mejor individuo en  $P$  es mejor que  $p_i$  then
14:       $p_i \leftarrow$  El mejor individuo en  $P$ 
15:      if  $p_i$  es mejor que  $s^*$  then
16:         $s^* \leftarrow p_i$ 
17: return  $s^*$ 
```

Esta versión de PSO puede clasificarse como una metaheurística de la clase de *Algoritmos de Coevolución Cooperativa* [DGH09] (puesto que mantiene diferentes poblaciones que evolucionan de forma colaborativa) y *Algoritmos de Estimación de Distribución* (debido a que adopta las ideas de representación descritas por PBIL).

Capítulo 4

Evaluación Experimental

Este capítulo consiste en la descripción del diseño experimental empleado durante la evaluación de los métodos introducidos en los capítulos anteriores, así como el análisis de los resultados obtenidos de dicha evaluación. El objetivo de este estudio radica en la comparación empírica de cinco metaheurísticas (*i.e.* GGA, SGA, CHC, PBIL y PSO) adaptadas para encontrar soluciones al problema de selección de instancias. Adicionalmente, se pretende estudiar el impacto de la aplicación de estrategias alternativas de generación de soluciones iniciales: disminuyendo la probabilidad de aparición de cada bit (de 50 % a 5 %) y modificando dichas probabilidades en función de los subconjuntos de instancias generados por diferentes algoritmos heurísticos (*i.e.* CNN, NEHS, Closest NE y Farthest NE).

4.1. Diseño Experimental

En este trabajo, el estudio experimental busca medir el impacto de *a)* las metaheurísticas usadas y *b)* las estrategias de inicialización aplicadas, en los resultados obtenidos en términos de los objetivos del problema de SI y el tiempo de ejecución. Para esto se hace uso de varios conjuntos de datos, que ayuden a identificar los efectos de los diferentes factores a evaluar. La metodología seguida debe permitir la generalización del comportamiento tanto de las metaheurísticas como de las estrategias de generación de soluciones iniciales, con la finalidad de comparar los resultados obtenidos y establecer los métodos más efectivos frente al problema de selección de instancias.

4.1.1. Conjuntos de datos

Un factor esencial en la evaluación de métodos de selección de instancias es el conjunto de datos utilizado; la distribución de los datos, el número de instancias y atributos, y la cantidad de datos ruidosos, son solo algunos de los elementos que modifican el espacio de búsqueda del problema y por ende la efectividad de los algoritmos heurísticos para encontrar buenas soluciones. Los conjuntos de datos usados en este trabajo pertenecen al *UCI Machine Learning Repository* [BL13] y al *KEEL Data-Mining Software Tool* [AFL⁺10].

Los 14 conjuntos seleccionados fueron separados en 3 grupos en función del número de instancias de cada conjunto. En la tabla 4.1 se presentan 9 conjuntos con menor cantidad de instancias (menor a 1000).

CONJUNTO	INSTANCIAS	ATRIBUTOS	CLASES
Cleveland	297	13	5
Glass	214	9	7
Iris	150	4	3
Led7Digit	500	7	10
Monk	432	6	2
Pima	768	8	2
WDBC	569	30	2
Wine	178	13	3
Wisconsin	683	9	2

TABLA 4.1: Conjuntos de datos pequeños

En la tabla 4.2 se describen dos conjuntos de datos caracterizados como de tamaño medio. Estos conjuntos se caracterizan por tener un número de instancias entre 1000 y 6000.

CONJUNTO	INSTANCIAS	ATRIBUTOS	CLASES
Banana	5300	2	2
Segmentation	2100	19	7

TABLA 4.2: Conjuntos de datos medianos

Finalmente, los 3 conjuntos con mayor número de instancias se presentan en la tabla 4.3. Estos conjuntos contienen un número de instancias mayor a 6000 junto con un número de atributos alto.

CONJUNTO	INSTANCIAS	ATRIBUTOS	CLASES
Penbased	10992	16	10
Satimage	6435	36	6
Thyroid	7200	21	3

TABLA 4.3: Conjuntos de datos grandes

4.1.2. Particiones y ejecuciones

Los conjuntos de datos considerados en la sección anterior son particionados usando la estrategia de validación cruzada en 10 grupos (*10-fold cross-validation*). El conjunto inicial de instancias T es dividido en 10 subconjuntos disjuntos de igual tamaño T_1, T_2, \dots, T_{10} . Cada conjunto T_i mantiene la proporción de distribución de las clases del conjunto original T . En función de esta partición, se definen los pares de conjuntos (T'_i, T_i) , con $i = 1 \dots 10$, donde $T'_i = T \setminus T_i$.

El conjunto T'_i , también conocido como el conjunto de “entrenamiento”, es usado por las metaheurísticas durante el proceso de búsqueda para evaluar soluciones intermedias, en vez de usar T . Las instancias restantes (pertenecientes al conjunto T_i) son usadas como conjunto de “validación”, *i.e.* la mejor solución encontrada durante la ejecución, es usada para clasificar las instancias desconocidas de T_i .

En base a esta estrategia, se definen los criterios de comparación empleados en el presente trabajo. Dada una solución $R \subseteq T'_i$ al problema de SI encontrada por una metaheurística, se considera:

- *Error de Entrenamiento*

El porcentaje de instancias en T'_i mal clasificadas usando R como conjunto de prototipos en un clasificador 1-NN. Para el cálculo de esta métrica no se remueven las instancias antes de ser clasificadas (*Leave-one-out cross-validation*), por lo que toda instancia en R se considera correctamente clasificada.

- *Error de Validación*

El porcentaje de instancias en T_i mal clasificadas usando R como conjunto de prototipos en un clasificador 1-NN.

- *Tamaño*

El tamaño de R en función de T'_i (*i.e.* $\frac{100|R|}{|T'_i|}$).

- *Tiempo*

Segundos empleados por el algoritmo durante la búsqueda.

Cada metaheurística es evaluada usando los 10 pares de conjuntos (T'_i, T_i) . Por cada par de conjunto entrenamiento-validación se realizan 3 repeticiones. *i.e.* un total de 30 ejecuciones de una metaheurística para un conjunto de datos y un tipo de inicialización particular. Las ejecuciones fueron realizadas de manera independiente en máquinas de tipo `c3.2xlarge` de *Amazon EC2*, que disponen de 8 CPUs Intel Xeon E5-2680 v2 (Ivy Bridge), 15GB de memoria RAM y 80GB de disco duro de estado sólido.

Todos los experimentos se realizaron bajo el sistema operativo Amazon Linux AMI 2014.03.2 y utilizando GCC 4.8.2. En este sentido, la implementación realizada aprovecha la arquitectura usada durante la implementación, evaluando las soluciones intermedias mediante la creación de hilos.

4.1.3. Parámetros

Cada metaheurística depende de un conjunto de parámetros que regulan el proceso de búsqueda sobre el espacio de posibles soluciones. El valor de cada parámetro y la interacción entre ellos, determinan el comportamiento del algoritmo y su capacidad para encontrar buenas soluciones. Sin embargo, los valores que indican el buen comportamiento de una metaheurística, son altamente dependientes del problema en general, e incluso de la instancia particular que se pretenda evaluar. Esto lo convierte en un proceso complejo, que a menudo conlleva a un diseño factorial con el fin de estudiar la influencia de los parámetros y sus interacciones en asegurar la calidad de las soluciones obtenidas.

En el presente trabajo se realizó la entonación de las metaheurísticas seleccionadas en los capítulos anteriores. Los resultados obtenidos de este proceso se describen en el Apéndice A. En la tabla 4.4 se presentan los parámetros seleccionados para cada metaheurística.

PARÁMETROS	ALGORITMOS				
	GGA	SGA	CHC	PBIL	PSO
Iteraciones	1000	1000	1000	1000	1000
Población	50	30	30	40	5
Prob. de Cruce	0.9	1.0	-	-	-
Prob. de Mutación	0.001	0.001	-	0.001	-
Mutation Shift	-	-	-	0.01	-
Learning Rate	-	-	-	0.1	-
Neg. Learning Rate	-	-	-	0.01	-
Partículas	-	-	-	-	5
Velocidad Máxima	-	-	-	-	0.2
Inercia	-	-	-	-	0.9
c1	-	-	-	-	0.1
c2	-	-	-	-	0.1

TABLA 4.4: Parámetros usados en cada metaheurística

4.1.4. Tablas de resultados

En la sección 4.2 son presentados y analizados los resultados obtenidos del proceso experimental. Las tablas de resultados deben permitir identificar las mejores estrategias de inicialización o las mejores metaheurísticas, según corresponda.

En primer lugar, se presenta una tabla con los resultados promedio de cada metaheurística, en función de los 4 criterios de comparación descritos en la subsección 4.1.3. Para una metaheurística particular, se promedian sus resultados para los conjuntos de datos que correspondan.

Sin embargo, al promediar los resultados puede perderse información relevante, puesto que el comportamiento de una metaheurística depende del conjunto de datos que se use para evaluarla. Por esta razón, se introducen las tablas de ranking, que presentan información adicional que permite comparar las metaheurísticas bajo un mismo contexto. Por cada conjunto de datos, las metaheurísticas son rankeadas de acuerdo a sus resultados en una métrica particular. A partir del ordenamiento en cada conjunto de datos, la tabla muestra el ranking promedio de cada metaheurística y el número de veces que logró el mejor resultado. Estos valores son calculados usando las métricas de tamaño, error de validación y la combinación lineal de ambos.

Ambas tablas son usadas para presentar los resultados por cada estrategia de inicialización. Para la primera tabla, se promedian los resultados no solo por conjunto de datos, sino también por metaheurística. Para las tablas de ranking ocurre lo mismo: los ordenamientos son realizados por cada combinación de metaheurística y conjunto de datos evaluados.

4.2. Resultados

En esta sección se presentan los resultados obtenidos durante la evaluación experimental. En la sección 4.2.1 se comparan diferentes estrategias de inicialización de soluciones con representación binaria, *a)* evaluando el impacto de disminuir la probabilidad de aparición de bits, y *b)* modificando dicha probabilidad de forma individual a lo largo de la cadena de bits, de un valor constante δ para todos los bits, a valores que favorezcan a las instancias pertenecientes a los conjuntos seleccionados por algoritmos heurísticos al problema de SI. Una vez seleccionada la mejor estrategia de inicialización, en la sección 4.2.2 se comparan los resultados obtenidos por las 5 metaheurísticas descritas. Se concluye con un estudio que busca determinar (en caso que existan) aquellas metaheurísticas que se comporten consistentemente mejor que el resto.

4.2.1. Comparación entre inicializaciones

Bajo el contexto del problema de SI, modificar la estrategia de inicialización de soluciones tiene un objetivo claro: iniciar el proceso de búsqueda en “buenas” soluciones. Una buena solución al problema de SI debe minimizar tanto la cardinalidad del conjunto R , como el error de clasificación usando R como conjunto de prototipos. En este sentido, se propone disminuir la probabilidad de aparición de bits δ para reducir la cardinalidad de soluciones iniciales, y modificar la probabilidad de cada bit de forma individual, beneficiando a instancias seleccionadas por diferentes algoritmos heurísticos, con la intención de reducir el error de clasificación.

Con la finalidad de comparar diferentes estrategias de inicialización, se realizaron experimentos sobre los conjuntos de datos grandes (Tabla 4.3). Los resultados obtenidos son presentados y analizados a continuación.

4.2.1.1. Inicialización disminuyendo la probabilidad de aparición de bit

Con la finalidad de reducir la cardinalidad de las soluciones iniciales generadas, en un intento de reducir el número de iteraciones necesarias para conseguir porcentajes de reducción aceptables, este estudio propone la reducción de la probabilidad de aparición de bits δ . En esta sección se estudia el impacto de la disminución propuesta, usando 3 posibles valores: 50 %, 25 % y 5 %.

En la tabla 4.5 se presentan los resultados promedio de la evaluación de las 5 metaheurísticas sobre los conjuntos de datos grandes, modificando únicamente el parámetro δ (probabilidad de aparición de bit).

δ	TIEMPO	TAMAÑO	% de ERROR	
			Entrenamiento	Validación
50 %	1324.96	37.09	2.94	6.60
25 %	1193.26	19.83	4.14	6.71
5 %	893.24	4.67	6.25	7.28

TABLA 4.5: Resultados promedio de las 5 metaheurísticas frente a los conjuntos de datos grandes, usando una probabilidad de aparición de bit δ igual a 50 %, 25 % y 5 %.

Estos datos corroboran la hipótesis de que al usar $\delta = 50 \%$, es necesario un mayor número de iteraciones para converger a soluciones con porcentajes de reducción aceptables [CHL03]. Como era de esperarse, la disminución del parámetro δ implica una reducción similar en términos del tamaño de las soluciones encontradas. Esto a su vez, supone una disminución importante en el tiempo de ejecución de los algoritmos, debido a que la evaluación de soluciones intermedias depende en gran medida de la cardinalidad de dichas soluciones.

En la figura 4.1 resultan evidentes las diferencias en el tamaño de las soluciones encontradas usando las 3 probabilidades seleccionadas. No es únicamente una diferencia promedio: la inicialización usando $\delta = 5 \%$ logra que las 5 metaheurísticas consigan soluciones más consistentes en términos de reducción.

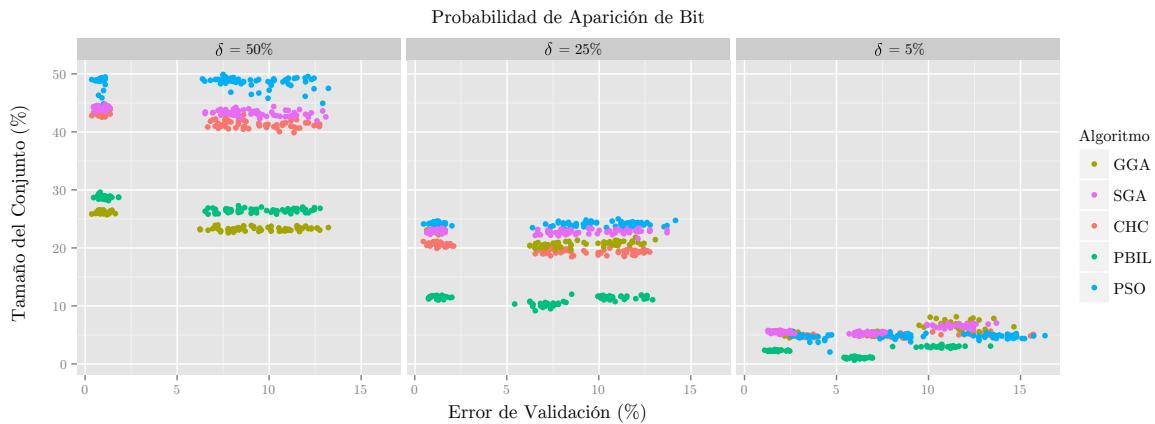


FIGURA 4.1: Tamaño del conjunto *vs* error de validación en ejecuciones de las metaheurísticas usando δ igual a 50 %, 25 % y 5 %.

Sin embargo, la disminución del parámetro δ conlleva a un aumento en los errores de entrenamiento y validación. No obstante, en ambos casos los errores se mantienen en valores aceptables, y en el caso del error de validación, la diferencia es poco significativa.

ERROR DE VALIDACIÓN			TAMAÑO			ERROR + TAMAÑO		
δ	Rank	Mejor	δ	Rank	Mejor	δ	Rank	Mejor
50 %	1.73	9	5 %	1.0	15	5 %	1.0	15
25 %	1.93	4	25 %	2.0	0	25 %	2.0	0
5 %	2.33	2	50 %	3.0	0	50 %	3.0	0

TABLA 4.6: Ranking de probabilidades δ según el tamaño y error de validación, considerando los promedios de las ejecuciones de cada algoritmo frente a los conjuntos de datos grandes. Por cada valor de δ , se presenta su ranking promedio (*Rank*) y el número de veces que obtuvo los mejores resultados (*Mejor*).

A pesar de que las diferencias en el error de validación favorecen el uso de $\delta = 50 \%$, la disminución de la probabilidad de aparición de bit al 5 % supone beneficios importantes en relación a la reducción alcanzada (ver la tabla 4.6) y al tiempo de ejecución. Por esta razón, se decidió usar $\delta = 5 \%$ durante el resto de la evaluación experimental.

4.2.1.2. Inicialización usando algoritmos heurísticos

En esta sección se evalúa el impacto de generar soluciones iniciales, modificando de forma independiente la probabilidad de aparición de bits lo largo la cadena que representa una solución al problema de SI. El enfoque estándar asigna un valor de probabilidad constante igual a δ , *i.e.* todos los bits tienen igual probabilidad de estar “prendidos”. En el presente trabajo, se propone una técnica de modificación que asigna valores de probabilidad diferente en base a la selección realizada por algoritmos heurísticos.

INICIALIZACIÓN	TIEMPO	TAMAÑO	% de ERROR	
			Entrenamiento	Validación
Constante	893.24	4.67	6.25	7.28
NEHS	958.94	4.70	6.20	7.28
CNN	921.68	5.11	6.27	7.85
FarthestNE	932.19	4.98	6.94	7.67
ClosestNE	932.75	5.56	7.21	8.95

TABLA 4.7: Resultados promedio de las 5 metaheurísticas frente a los conjuntos de datos grandes, usando valores de probabilidad constante y basados en algoritmos heurísticos.

En la tabla 4.7 se presentan los resultados promedio de las 5 metaheurísticas descritas, generando soluciones iniciales con un valor de probabilidad constante o con valores en función de las selecciones generadas por CNN, NEHS, Closest NE y Farthest NE. Los datos en tiempo de ejecución son previsibles, debido al tiempo que requieren estos algoritmos heurísticos para generar sus soluciones. Sin embargo, contrario a lo esperado, estas modificaciones no mejoran significativamente los porcentajes de error obtenidos mediante el uso de una probabilidad constante. Más aún, las soluciones basadas en CNN, Closest NE y Farthest NE empeoran los resultados en todos los aspectos. Solo probabilidades en función de NEHS reportan resultados alentadores, mejorando el error de entrenamiento e igualando el error de validación frente al uso de una probabilidad constante.

La figura 4.2 confirma los aspectos señalados: el comportamiento de inicializaciones basadas en CNN, Closest NE y Farthest NE difieren significativamente del comportamiento de aquellas basadas en NEHS o con probabilidad constante. Estas últimas muestran un comportamiento similar en función de ambos objetivos del problema.

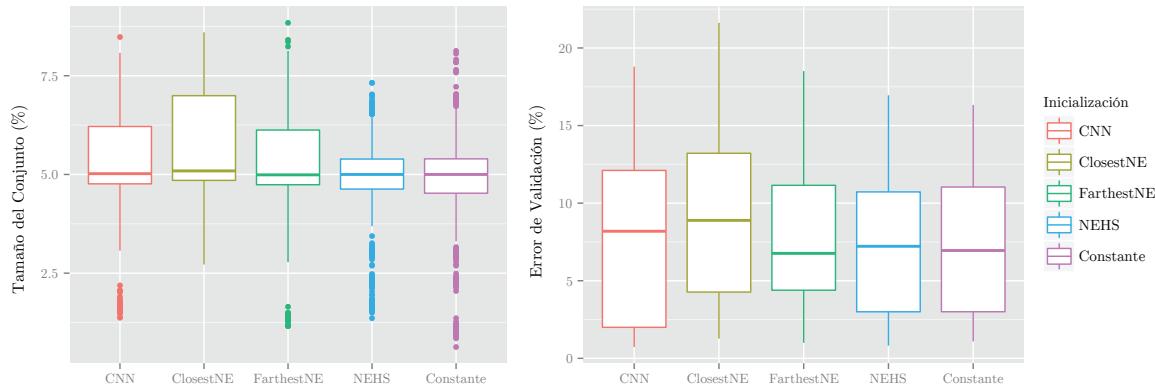


FIGURA 4.2: Diagramas de caja del tamaño del conjunto y error de validación usando valores de probabilidad constante y basados en algoritmos heurísticos.

Sin embargo, la tabla 4.8 sugiere que una inicialización basada en NEHS resulta conveniente en función de los dos objetivos del problema, siendo el que presenta mejor ranking promedio en base al error de validación y a la combinación lineal entre el error y el tamaño del conjunto seleccionado. En función de estos resultados, se usa NEHS como heurística de inicialización de las ejecuciones realizadas en la sección 4.2.2, cuya finalidad es determinar aquellas metaheurísticas que se comporten consistentemente mejor que las demás.

ERROR DE VALIDACIÓN			TAMAÑO			ERROR + TAMAÑO		
Inicialización	Rank	Mejor	Inicialización	Rank	Mejor	Inicialización	Rank	Mejor
NEHS	2.06	4	Constante	1.86	6	NEHS	2.00	5
Constante	2.40	2	NEHS	2.26	3	Constante	2.20	2
CNN	2.86	4	FarthestNE	2.86	4	CNN	2.93	4
FarthestNE	2.93	5	CNN	3.60	2	FarthestNE	3.06	4
ClosestNE	4.73	0	ClosestNE	4.40	0	ClosestNE	4.80	0

TABLA 4.8: Ranking de inicializaciones con probabilidades constantes o basadas en heurísticas, según el tamaño y error de validación, considerando los promedios de las ejecuciones de cada algoritmo frente a los conjuntos de datos grandes. Por cada inicialización, se presenta su ranking promedio (*Rank*) y el número de veces que obtuvo los mejores resultados (*Mejor*).

4.2.2. Comparación entre metaheurísticas

Para controlar el proceso de búsqueda, cada metaheurística emplea técnicas diversas para recorrer el espacio de posibles soluciones. Su capacidad para encontrar buenas

soluciones está íntimamente ligado al problema que se desea resolver, la representación de sus soluciones, la función de objetivo, entre otros factores. La distribución del espacio de búsqueda puede alterar significativamente la efectividad de una metaheurística particular. Por esta razón, al estudiar el comportamiento de diferentes metaheurísticas frente al problema de SI, es necesario identificar aquellas metaheurísticas que exhiban mejor capacidad para seleccionar subconjuntos de instancias que cumplan con los objetivos planteados por el problema.

A continuación se presenta un estudio fraccionado del comportamiento de las 5 metaheurísticas, adaptadas en función de los resultados descritos en la sección 4.2.1. Se describen los resultados obtenidos usando los conjuntos de datos caracterizados como pequeños, medianos y grandes. Finalmente, se realiza un análisis estadístico conjunto, con la finalidad de determinar si existen diferencias significativas entre las metaheurísticas.

4.2.2.1. Resultados para conjuntos de datos pequeños

En la tabla 4.9 se presentan los resultados promedio de cada metaheurística, al ser evaluadas usando los conjuntos de datos pequeños. En función del error de clasificación en los conjuntos de entrenamiento y validación, GGA muestra mejores resultados que el resto de las metaheurísticas. Sin embargo, el error de validación alcanzado por las demás metaheurísticas, en particular los resultados de SGA y PBIL, son bastante cercanos. En términos de la reducción lograda, son PBIL y CHC los que seleccionan conjuntos de datos de menor tamaño, mientras que PSO es el que consigue los peores resultados.

Las mayores diferencias se ven reflejadas en los tiempos de ejecución, donde PSO y GGA requieren de un mayor número de segundos para concluir la búsqueda. No obstante, cabe destacar que sobre los conjuntos de datos pequeños, los tiempos de ejecución de estas metaheurísticas no son prohibitivos.

ALGORITMO	TIEMPO	TAMAÑO	% de ERROR	
			Entrenamiento	Validación
GGA	5.51	4.66	10.94	17.89
SGA	0.70	4.85	13.23	18.94
CHC	1.75	3.44	14.15	19.45
PBIL	2.95	3.36	13.32	18.64
PSO	10.59	6.28	18.64	21.90

TABLA 4.9: Resultados promedio de los conjuntos de datos pequeños, usando las metaheurísticas descritas.

La tabla de rankings (Tabla 4.10) ofrece otra perspectiva. PBIL domina los rankings en función del error de validación, el tamaño del conjunto seleccionado y la combinación de ambos objetivos. El ranking promedio de GGA y CHC en términos del error y el tamaño respectivamente, corroboran los resultados descritos anteriormente.

ERROR DE VALIDACIÓN			TAMAÑO			ERROR + TAMAÑO		
Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor
PBIL	2.00	4	PBIL	1.44	5	PBIL	1.55	6
GGA	2.22	4	CHC	1.88	3	GGA	2.44	2
SGA	2.66	1	GGA	3.11	1	CHC	2.88	0
CHC	3.77	0	SGA	3.88	0	SGA	3.22	1
PSO	4.33	0	PSO	4.66	0	PSO	4.88	0

TABLA 4.10: Ranking de metaheurísticas según el tamaño y error de validación, considerando los promedios de las ejecuciones de cada algoritmo frente a los conjuntos de datos pequeños. Por cada algoritmo, se presenta su ranking promedio (*Rank*) y el número de veces que obtuvo los mejores resultados (*Mejor*).

Los datos sugieren que PBIL representa la mejor alternativa para encontrar soluciones a instancias “pequeñas” del problema de SI, ya que logra buenos resultados tanto en términos de precisión de clasificación, como en términos de reducción. Sin embargo, GGA y CHC presentan resultados comparables, por lo que no deben descartarse.

4.2.2.2. Resultados para conjuntos de datos medianos

En la tabla 4.11 se presentan los resultados obtenidos al evaluar las metaheurísticas frente a los conjuntos de datos medianos. Los tiempos de ejecución aumentan notablemente, y las diferencias en tiempo entre los diferentes algoritmos se mantienen: PSO y GGA requieren en promedio más del doble del tiempo que PBIL, y pueden

tardar hasta 9 veces más que SGA y CHC. De nuevo, PBIL y GGA logran los mejores resultados en función del error de clasificación. Sin embargo, los porcentajes de error reportados (sobretodo en el conjunto de validación) no varían considerablemente entre metaheurísticas; a excepción de PSO, que exhibe los mayores errores de clasificación en ambos casos. Por último, el tamaño de los conjuntos encontrados por PBIL es menor que en el resto de algoritmos, aunque CHC también reporta una reducción considerable.

ALGORITMO	TIEMPO	TAMAÑO	% de ERROR	
			Entrenamiento	Validación
GGA	80.49	6.02	5.96	8.58
SGA	8.67	5.80	6.76	8.92
CHC	8.89	4.46	7.51	8.90
PBIL	38.06	2.72	6.06	8.35
PSO	97.98	5.55	10.66	11.83

TABLA 4.11: Resultados promedio de los conjuntos de datos medianos, usando las metaheurísticas descritas.

Estas observaciones son ratificadas con los datos de la tabla 4.12, que muestra el dominio de los resultados dados por PBIL bajo las métricas del tamaño del conjunto seleccionado y el error de validación alcanzado. No obstante, los resultados reportados por CHC son interesantes en función del ranking conjunto, tomando en cuenta la diferencia en tiempo de ejecución existente en comparación con PBIL.

ERROR DE VALIDACIÓN			TAMAÑO			ERROR + TAMAÑO		
Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor
PBIL	1.5	1	PBIL	1.0	2	PBIL	1.0	2
GGA	2.5	1	CHC	2.0	0	CHC	2.0	0
SGA	3.0	0	PSO	3.0	0	GGA	3.5	0
CHC	3.0	0	GGA	4.5	0	SGA	3.5	0
PSO	5.0	0	SGA	4.5	0	PSO	5.0	0

TABLA 4.12: Ranking de metaheurísticas según el tamaño y error de validación, considerando los promedios de las ejecuciones de cada algoritmo frente a los conjuntos de datos medianos. Por cada algoritmo, se presenta su ranking promedio (*Rank*) y el número de veces que obtuvo los mejores resultados (*Mejor*).

4.2.2.3. Resultados para conjuntos de datos grandes

Los resultados de las ejecuciones usando los conjuntos de datos grandes se presentan en la tabla 4.13. El tiempo requerido por las metaheurísticas para solucionar los conjuntos grandes es notablemente mayor que el requerido para conjuntos medianos. Esto es una consecuencia directa del aumento, no solo en el número de instancias, sino en el número de atributos de los datos: el fenómeno conocido como la «*maldición de la dimensionalidad*». Son aumentos en el orden de 24, 22, 17, 12 y 2 veces el tiempo requerido por GGA, PSO, SGA, PBIL y CHC respectivamente, en comparación con los tiempos reportados para los conjuntos medianos.

ALGORITMO	TIEMPO	TAMAÑO	% de ERROR	
			Entrenamiento	Validación
GGA	1938.78	5.58	6.11	7.13
SGA	152.71	5.74	5.42	6.56
CHC	22.37	5.00	7.35	8.08
PBIL	472.63	2.36	4.35	6.19
PSO	2208.22	4.80	7.76	8.43

TABLA 4.13: Resultados promedio de los conjuntos de datos grandes, usando las metaheurísticas descritas.

En función del tamaño de los conjuntos seleccionados, los resultados de PBIL dominan los del resto de las metaheurísticas. Adicionalmente, los mejores resultados en términos del error de clasificación (ambos) son reportados por PBIL, conclusión respaldada por la tabla de rankings 4.14. Sin embargo, los porcentajes de error logrados por SGA y GGA son competitivos frente a PBIL, lo cual se ve reflejado en los rankings promedio del error de validación y el ranking conjunto.

En el trabajo de *Cano et al.* [CHL03] se utilizan estos conjuntos de datos (*i.e.* *Penbased*, *Satimage* y *Thyroid*) para encontrar soluciones al problema de SI usando GGA, SGA, CHC y PBIL. Esto abre la posibilidad de establecer comparaciones en términos del tamaño de las soluciones encontradas y el error de validación alcanzado. En la tabla 4.15 se muestran los resultados promedio obtenidos en el presente trabajo, junto con los datos publicados por *Cano* para los conjuntos de datos mencionados.

Los datos indican que, si bien existe una diferencia en función del error de validación a favor de las soluciones encontradas por *Cano*, existen diferencias significativas en

ERROR DE VALIDACIÓN			TAMAÑO			ERROR + TAMAÑO		
Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor
PBIL	1.00	3	PBIL	1.00	3	PBIL	1.00	3
SGA	2.00	0	PSO	2.00	0	SGA	2.00	0
GGA	3.00	0	CHC	3.00	0	GGA	3.33	0
CHC	4.33	0	GGA	4.33	0	CHC	4.33	0
PSO	4.66	0	SGA	4.66	0	PSO	4.33	0

TABLA 4.14: Ranking de metaheurísticas según el tamaño y error de validación, considerando los promedios de las ejecuciones de cada algoritmo frente a los conjuntos de datos grandes. Por cada algoritmo, se presenta su ranking promedio (*Rank*) y el número de veces que obtuvo los mejores resultados (*Mejor*).

ALGORITMO	TAMAÑO		ERROR DE VALIDACIÓN	
	<i>Flores et al.</i>	<i>Cano et al.</i>	<i>Flores et al.</i>	<i>Cano et al.</i>
GGA	5.58	37.47	7.13	6.15
SGA	5.74	37.09	6.56	6.33
CHC	5.00	0.71	8.08	6.47
PBIL	2.36	26.87	6.19	5.87
<i>Promedio</i>	4.67	25.53	6.99	6.20

TABLA 4.15: Comparación con resultados publicados por *Cano et al.* usando GGA, SGA, CHC y PBIL para los conjuntos *Penbased*, *Satimage* y *Thyroid*.

términos de la reducción del conjunto original. En promedio, las soluciones alcanzadas por nuestra implementación tienen un tamaño 5 veces menor que las encontradas por el trabajo de *Cano*. Sin embargo, deben destacarse los excelentes resultados exhibidos por CHC en dicho estudio y en contraste con los resultados obtenidos en este trabajo. Esta diferencia puede deberse a que en nuestro estudio se usa un menor número de iteraciones y de tamaño de población para la búsqueda realizada por CHC.

4.2.2.4. Análisis de los resultados

De los resultados presentados anteriormente, resulta evidente la existencia de diferencias en el comportamiento de cada metaheurística y en su capacidad para encontrar soluciones que minimicen ambos objetivos del problema de SI. En esta sección se pretende identificar aquellas metaheurísticas que exhiben un comportamiento consistentemente mejor que el resto de los algoritmos, en función de los dos objetivos del problema de SI y con respecto al tiempo de ejecución que requieren.

En la tabla 4.16 se presentan los resultados promedio en error de validación y tamaño de la solución conseguidos por cada metaheurística frente a los conjuntos de datos descritos en la sección 4.1.1.

En promedio, los mejores resultados en términos de error son los alcanzados por GGA, con un 14.25 % de error de clasificación en los conjuntos de validación. Sin embargo, aunque el error promedio de PBIL (14.50 %) es mayor que GGA, PBIL logra los mejores resultados en 8 de los 14 conjuntos de datos, mientras que GGA lo logra en tan solo 5 conjuntos. Esto ubica a PBIL en el primer lugar en el ranking según el error de validación de las metaheurísticas (ver la tabla 4.17).

CONJUNTO	GGA		SGA		CHC		PBIL		PSO	
	Error	Tam.	Error	Tam.	Error	Tam.	Error	Tam.	Error	Tam.
Banana	10.83	5.33	10.60	5.33	10.67	4.68	10.37	1.95	11.80	4.87
Cleveland	45.81	6.02	43.92	5.18	44.81	3.70	43.16	3.21	44.53	5.63
Glass	32.49	10.19	32.85	9.38	36.55	6.31	35.02	6.99	35.38	13.43
Iris	5.11	2.66	4.44	3.00	5.11	2.93	6.22	2.78	4.66	4.86
Led7digit	26.03	4.84	26.89	4.88	26.10	3.10	25.37	3.15	30.25	6.71
Monk	11.16	6.57	18.67	7.26	19.16	4.53	17.88	4.85	29.40	7.49
Penbased	2.46	5.10	1.84	5.52	2.99	4.93	1.59	2.32	3.03	4.86
Pima	27.43	5.08	27.91	5.48	26.35	4.04	26.26	3.68	30.08	4.97
Satimage	11.80	6.49	11.21	6.40	12.92	5.18	10.58	3.04	14.00	4.77
Segmentation	6.33	6.72	7.25	6.26	7.12	4.23	6.34	3.48	11.85	6.24
Thyroid	7.15	5.16	6.63	5.29	8.34	4.90	6.40	1.71	8.26	4.76
WDBC	6.91	2.97	8.43	3.82	8.43	2.64	7.32	2.27	13.23	4.84
Wine	3.17	2.82	4.53	3.03	5.08	2.95	3.93	2.62	6.00	4.76
Wisconsin	2.88	0.81	2.83	1.66	3.46	0.75	2.63	0.68	3.60	3.86
<i>Promedio</i>	14.25	5.05	14.86	5.18	15.51	3.92	14.50	3.05	17.58	5.86

TABLA 4.16: Resultados promedio del error de validación y tamaño de la solución, de cada metaheurística para cada conjunto de datos. Se marcan en **negrita** los mejores resultados en error y tamaño por cada conjunto de datos.

En función del tamaño de las soluciones encontradas, los resultados exhibidos por PBIL dominan al resto de las metaheurísticas. Seleccionando en promedio el 3.05 % de las instancias de los conjuntos de datos iniciales, PBIL exhibe los mejores resultados en función de la reducción de los datos originales. Adicionalmente, PBIL consigue las soluciones de menor tamaño en 10 de los 14 conjuntos de datos.

Sin embargo, los resultados de CHC en términos del tamaño de las soluciones encontradas son interesantes. Presenta un tamaño de soluciones promedio igual a

ERROR DE VALIDACIÓN			TAMAÑO			ERROR + TAMAÑO		
Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor	Algoritmo	Rank	Mejor
PBIL	1.71	8	PBIL	1.28	10	PBIL	1.35	11
GGA	2.42	5	CHC	2.14	3	GGA	2.78	2
SGA	2.57	1	GGA	3.57	1	SGA	3.00	1
CHC	3.78	0	PSO	3.85	0	CHC	3.07	0
PSO	4.50	0	SGA	4.14	0	PSO	4.78	0

TABLA 4.17: Ranking de metaheurísticas según el tamaño y error de validación, considerando los promedios de las ejecuciones de cada algoritmo frente a todos los conjuntos de datos. Por cada algoritmo, se presenta su ranking promedio (*Rank*) y el número de veces que obtuvo los mejores resultados (*Mejor*).

3.92 %, solo 0.88 % más que PBIL. Adicionalmente, CHC consigue los conjuntos de menor tamaño en 3 ocasiones, logrando la segunda posición en dicho ranking.

En la figura 4.3 se muestra la distribución de las soluciones encontradas por cada metaheurística, en función de su error de validación (eje *x*) y su tamaño (eje *y*). Cada punto representa una de las 30 ejecuciones realizadas para cada conjunto de datos, usando el algoritmo correspondiente. Adicionalmente, se grafica una estimación de densidad multivariable con la finalidad de identificar con mayor facilidad, las áreas de concentración de los resultados.

Esta gráfica confirma las conclusiones obtenidas a través de los datos presentados: PBIL muestra los mejores resultados en función de los objetivos del problema de SI. En comparación con las demás metaheurísticas, PBIL exhibe un comportamiento más estable, *i.e.* la mayoría de sus soluciones presentan bajos porcentajes de error y tamaño. Al comparar estos datos con los de GGA, la gráfica de densidad muestra que los resultados de PBIL están mucho más concentrados en un área particular del plano, mientras que GGA presenta mayor variabilidad.

Para poder determinar con seguridad las posibles diferencias entre PBIL y el resto de las metaheurísticas, es necesario aplicar una prueba estadística. Se usa una prueba de rangos con signo de *Wilcoxon* [Wil45] (de cola inferior), con un nivel de significancia del 1 %, para determinar si la mediana de la resultados de PBIL es menor que la de los resultados de las demás metaheurísticas en términos de error de validación o tamaño. A continuación se establecen las hipótesis de la prueba para este caso particular:

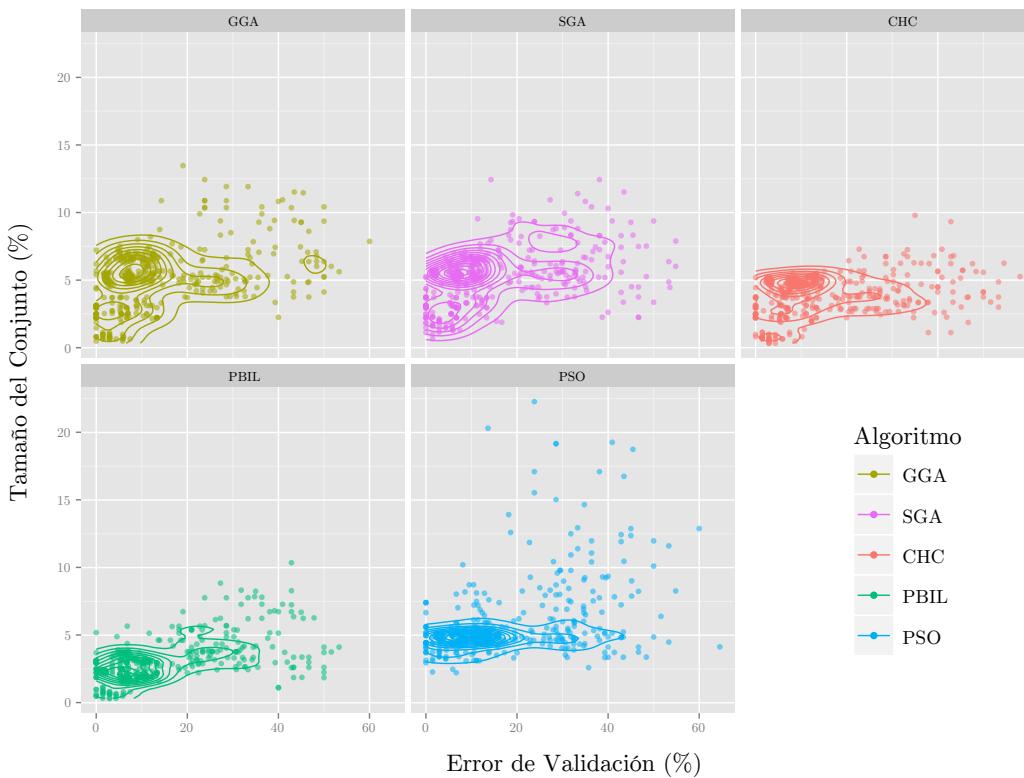


FIGURA 4.3: Distribución de los resultados de cada metaheurística, en función del error de validación y el tamaño de las soluciones.

H_0 Los resultados de PBIL son mayores o iguales que los del algoritmo en cuestión.

H_1 Los resultados de PBIL son menores que los del algoritmo en cuestión.

En función de estas hipótesis, en la tabla 4.18 se presenta el estadístico W y el p -valor que permitirán rechazar o no la hipótesis nula H_0 .

ALGORITMO	ERROR DE VALIDACIÓN		TAMAÑO	
	W	p -valor	W	p -valor
GGA	46	3.574×10^{-1}	1	1.221×10^{-4}
SGA	27	5.945×10^{-2}	0	6.104×10^{-5}
CHC	6	8.545×10^{-4}	14	6.714×10^{-3}
PSO	6	8.545×10^{-4}	0	6.104×10^{-5}

TABLA 4.18: Estadístico W y p -valor de pruebas de rangos con signo de Wilcoxon para determinar si PBIL es mejor que el resto de las metaheurísticas, en función del error de validación y el tamaño de las soluciones encontradas.

Estos resultados permiten afirmar con un nivel de significancia del 1%, que PBIL exhibe mejores resultados que a) todas las metaheurísticas en términos de la reducción

de los datos, y que *b)* CHC y PSO en función al error de validación. En resumidas cuentas, para encontrar soluciones al problema de SI, PBIL resulta la mejor opción en función de los objetivos del problema.

Sin embargo, debido a que el objetivo de estos algoritmos es que se apliquen sobre conjuntos de datos de tamaños grandes (mucho mayores a los conjuntos de datos evaluados en este estudio), la métrica del tiempo de ejecución es particularmente importante para evaluar las capacidades de las metaheurísticas para escalar a conjuntos mayores.

En este sentido, en la figura 4.4 se grafica el aumento en tiempo de ejecución de las metaheurísticas en función del número de instancias de los 14 conjuntos de datos evaluados. Mientras que GGA, SGA, PBIL y PSO exhiben un aumento sustancial en el tiempo de ejecución, resulta evidente que CHC se ve menos afectado por el número de instancias que el resto de las metaheurísticas. Es destacable que CHC logre, en un tiempo significativamente menor, resultados competitivos en función del tamaño de las soluciones encontradas y manteniendo un error de validación aceptable. Esto hace de CHC el candidato perfecto para ser aplicado bajo conjuntos de datos con gran número de instancias.

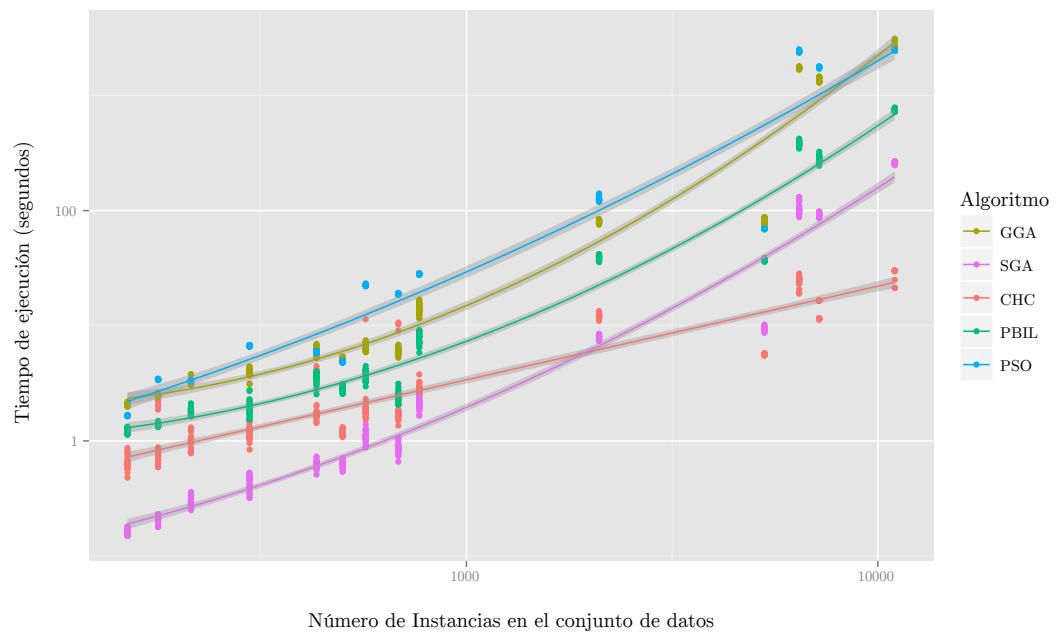


FIGURA 4.4: Tiempo de ejecución reportado por cada metaheurística, usando el tiempo promedio en cada conjunto de datos, ordenándolos en función del número de instancias en cada conjunto de datos. Se usa escala logarítmica en ambos ejes.

Conclusiones y Recomendaciones

En el presente trabajo fueron implementadas 5 metaheurísticas bio-inspiradas (*i.e.* GGA, SGA, CHC, PBIL y PSO) para encontrar soluciones al problema de Selección de Instancias. Para esto, se propusieron (en función de los objetivos del problema) una serie de modificaciones a las estrategias de generación de soluciones iniciales al problema, usadas por las metaheurísticas como punto inicial del proceso de búsqueda. Adicionalmente, se realizó un estudio comparativo entre las metaheurísticas implementadas.

Los aportes de este trabajo se centran en las modificaciones realizadas sobre la generación de soluciones iniciales. En primer lugar, se propuso disminuir la probabilidad de aparición de bits δ (normalmente fijada en 50 %), con la finalidad de reducir la cardinalidad de las soluciones iniciales generadas. Al usar una probabilidad igual al 5 %, se lograron los objetivos planteados en términos del tamaño de las soluciones, además de una disminución considerable en el tiempo de ejecución; todo esto sin afectar de manera significativa en el error de clasificación.

Adicionalmente, se propuso modificar la probabilidad de aparición de cada bit de forma individual, aumentando la probabilidad de aparición de las instancias seleccionadas por algoritmos heurísticos al problema de SI. En este sentido, se evaluó el impacto de usar probabilidades en función de las soluciones generadas por CNN, NEHS, Closest NE y Farthest NE, en contraste con el uso de una probabilidad constante. Contario a lo esperado, esta estrategia no resultó particularmente beneficiosa; únicamente la probabilidad en base a la selección de NEHS logró resultados comparables (y en algunos casos mejores) con el uso de una probabilidad constante.

Una vez determinadas las estrategias a seguir para la generación de soluciones, se procedió a realizar un estudio comparativo entre las metaheurísticas implementadas,

con la finalidad de determinar aquellas metaheurísticas que se comportan consistentemente mejor que las demás. De los resultados obtenidos, se concluye que PBIL encuentra las mejores soluciones en función de los objetivos del problema de SI. Sin embargo, el tiempo de ejecución de PBIL se ve muy afectado por el número de instancias en los conjuntos de datos. En este sentido, los resultados reportados por CHC lo convierten en una opción a considerar al momento de buscar soluciones al problema de SI sobre conjuntos de datos de tamaños mayores a los evaluados en este estudio.

En función del trabajo realizado, existen algunos aspectos relevantes a tomar en cuenta en futuros estudios sobre el problema de SI. A continuación se presentan algunas direcciones para expandir el presente estudio:

- Emplear una evaluación estratificada (como la descrita por [CHL03]) para encontrar soluciones a conjuntos de tamaño mayor que los usados en el presente trabajo.
- Analizar el impacto de generar soluciones iniciales mediante algoritmos heurísticos, en el proceso de búsqueda llevado a cabo por metaheurísticas de trayectoria.
- Evaluar adaptaciones alternativas de PSO para encontrar soluciones con representación binaria, con el objetivo de reducir el impacto que tiene el tamaño de los conjuntos de datos sobre su tiempo de ejecución.
- Profundizar el estudio de NEHS (y algoritmos basados en ideas similares), para determinar su capacidad para conseguir soluciones al problema de SI.

Bibliografía

- [AFL⁺10] J Alcalá, A Fernández, J Luengo, J Derrac, S García, L Sánchez, and F Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic and Soft Computing*, 17:255–287, 2010.
- [AR11] Miloud-Aouidate Amal and Baba-Ali Ahmed Riadh. Survey of nearest neighbor condensing techniques. 2011.
- [Bac96] T. Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.
- [Bal94] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical report, 1994.
- [Bal95] Shumeet Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical report, 1995.
- [BC95] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. pages 38–46. Morgan Kaufmann Publishers, 1995.
- [BDT99] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc., New York, NY, USA, 1999.
- [Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

- [BL97] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2):245–271, 1997.
- [BL13] K. Bache and M. Lichman. UCI machine learning repository, 2013.
- [BS12] S Siva Sathya Binitha S. A survey of bio inspired optimization algorithm. *International Journal of Soft Computing and Engineering*, 2(2):137–151, 2012.
- [BT12] J. Bien and R. Tibshirani. Prototype selection for interpretable classification. *ArXiv e-prints*, February 2012.
- [CF01] Vicente Cerveron and Francesc J Ferri. Another move toward the minimum consistent subset: a tabu search approach to the condensed nearest neighbor rule. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 31(3):408–413, 2001.
- [CH67] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, January 1967.
- [CHL03] José Ramón Cano, Francisco Herrera, and Manuel Lozano. Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study. *Evolutionary Computation, IEEE Transactions on*, 7(6):561–575, 2003.
- [DGH09] Joaquín Derrac, Salvador García, and Francisco Herrera. A first study on the use of coevolutionary algorithms for instance and feature selection. In *Proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems*, HAIS ’09, pages 557–564, Berlin, Heidelberg, 2009. Springer-Verlag.
- [DGKL94] Luc Devroye, Laszlo Gyorfi, Adam Krzyzak, and Gábor Lugosi. On the strong universal consistency of nearest neighbor regression function estimates. *The Annals of Statistics*, pages 1371–1385, 1994.
- [Esh90] Larry J Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. *Foundations of genetic algorithms*, pages 265–283, 1990.

- [FH51] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. *US Air Force School of Aviation Medicine*, Technical Report 4(3):477+, January 1951.
- [FI93] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In Ruzena Bajcsy, editor, *IJCAI*, pages 1022–1029. Morgan Kaufmann, 1993.
- [Gat72] Geoffrey W. Gates. The reduced nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.
- [GCH08] Salvador García, José Ramón Cano, and Francisco Herrera. A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709, 2008.
- [GDCH12] Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):417–435, March 2012.
- [GK14] Lee-Ad Gottlieb and Aryeh Kontorovich. Near-optimal sample compression for nearest neighbors. *CoRR*, abs/1404.3368, 2014.
- [GKK13] Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient classification for metric data. *CoRR*, abs/1306.2547, 2013.
- [GPY08] Roberto Gil-Pita and Xin Yao. Evolving edited k-nearest neighbor classifiers. *International Journal of Neural Systems*, 18(06):459–467, 2008.
- [Har68] P. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Trans. Inf. Theor.*, 14(3):515–516, September 1968.
- [HLL02] Shinn-Ying Ho, Chia-Cheng Liu, and Soundy Liu. Design of an optimal nearest neighbor classifier using an intelligent genetic algorithm. *Pattern Recognition Letters*, 23(13):1495–1503, 2002.
- [Hol75] J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

- [JG] Norbert Jankowski and Marek Grochowski. Comparison of instances selection algorithms i. algorithms survey.
- [KE95] J Kennedy and R Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948. IEEE, 1995.
- [KL04] Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In J. Ian Munro, editor, *SODA*, pages 798–807. SIAM, 2004.
- [KS98] J Kennedy and WM Spears. Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 78–83. IEEE, 1998.
- [KS04] Mohammad Kolahdouzan and Cyrus Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30, VLDB '04*, pages 840–851. VLDB Endowment, 2004.
- [LHTD02] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data Min. Knowl. Discov.*, 6(4):393–423, October 2002.
- [LM98] Huan Liu and Hiroshi Motoda. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.
- [LM02] Huan Liu and Hiroshi Motoda. On issues of instance selection. *Data Min. Knowl. Discov.*, 6(2):115–130, April 2002.
- [MG95] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

- [Muh91] Heinz Muhlenbein. Evolution in time and space - the parallel genetic algorithm. In *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann, 1991.
- [SE98] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 69–73. IEEE, 1998.
- [Ska94] David B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In William W. Cohen and Haym Hirsh, editors, *ICML*, pages 293–301. Morgan Kaufmann, 1994.
- [SLI⁺01] Basilio Sierra, Elena Lazkano, Iñaki Inza, Marisa Merino, Pedro Larrañaga, and Jorge Quiroga. Prototype selection and feature subset selection by estimation of distribution algorithms. a case study in the survival of cirrhotic patients treated with tips. In *Artificial Intelligence in Medicine*, pages 20–29. Springer, 2001.
- [SSBD14] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [Tou02] Godfried T. Toussaint. Open problems in geometric methods for instance-based learning. In Jin Akiyama and Mikio Kano, editors, *JCDCG*, volume 2866 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 2002.
- [Vor08] Georges Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les paralléloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.
- [Wil45] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, pages 80–83, 1945.
- [Wil72] DR Wilson. Asymptotic properties of nearest neighbor rules using edited data. *Institute of Electrical and Electronic Engineers Transactions on Systems, Man and Cybernetics*, 2:408–421, 1972.

- [Wil91] Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, pages 224–233, New York, NY, USA, 1991. ACM.
- [WK88] D. Whitley and J. Kauth. *GENITOR: A Different Genetic Algorithm*. Technical report (Colorado State University. Department of Computer Science). Colorado State University, Department of Computer Science, 1988.
- [WM97] D. Randall Wilson and Tony R. Martinez. Instance pruning techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 403–411, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [Yan08] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.
- [ZS02] Hongbin Zhang and Guangyu Sun. Optimal reference subset selection for nearest neighbor classification by tabu search. *Pattern Recognition*, 35(7):1481–1490, 2002.
- [Zuk10] A. V. Zikhba. Np-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recognit. Image Anal.*, 20(4):484–494, December 2010.

Apéndice A

Entonación de las metaheurísticas

En este apéndice se describe la metodología seguida durante el proceso de entonación de las metaheurísticas descritas en el Capítulo 2. Por cada metaheurística se describen los valores evaluados para cada parámetro y se presentan los resultados obtenidos de la experimentación. Estos experimentos fueron realizados usando *Banana* como conjunto de datos de entonación, realizando 30 ejecuciones por cada configuración (3 repeticiones por cada *fold*).

Cabe destacar, que para las metaheurísticas GGA, SGA y PBIL se fijó el valor de la probabilidad de mutación en 0.001. Este debe ser un valor pequeño para no convertir el proceso de búsqueda en uno aleatorio, y el valor seleccionado ha sido usado en otros estudios [CHL03]. El resto de parámetros de cada metaheurística fue entonado, y sus resultados se presentan a continuación.

A.1. Entonación de GGA

Los parámetros de los que depende GGA son: el número de iteraciones, el tamaño de la población y la probabilidad de cruce (la prob. de mutación ya está fijada). En la tabla A.1 se presentan los diferentes valores evaluados para cada metaheurística. En este sentido, se realizó un estudio factorial para determinar el impacto de cada valor por separado y la interacción entre valores de diferentes parámetros.

En la figura A.1 se presentan los resultados obtenidos de este estudio. Resulta evidente la existencia de un impacto inversamente proporcional entre el tamaño de

PARÁMETRO	VALORES EVALUADOS				
Iteraciones	100 1000 10000				
Población	10 20 30 40 50				
Prob. de Cruce	0.5	0.6	0.75	0.9	1.0

TABLA A.1: Valores evaluados para los parámetros de GGA.
En negrita los valores fijados por cada parámetro.

población y el error de validación. Con una población de tamaño 50, GGA logra los mejores resultados. Adicionalmente, los mejores resultados se obtienen mediante el uso de una probabilidad de cruce igual a 1.0 o 0.9, favoreciendo ligeramente a la selección de una probabilidad igual a 0.9. Sin embargo, los resultados en función del número de iteraciones no son concluyentes, puesto que los resultados exhiben un comportamiento errático. Se selecciona un número de iteraciones igual a 1000.

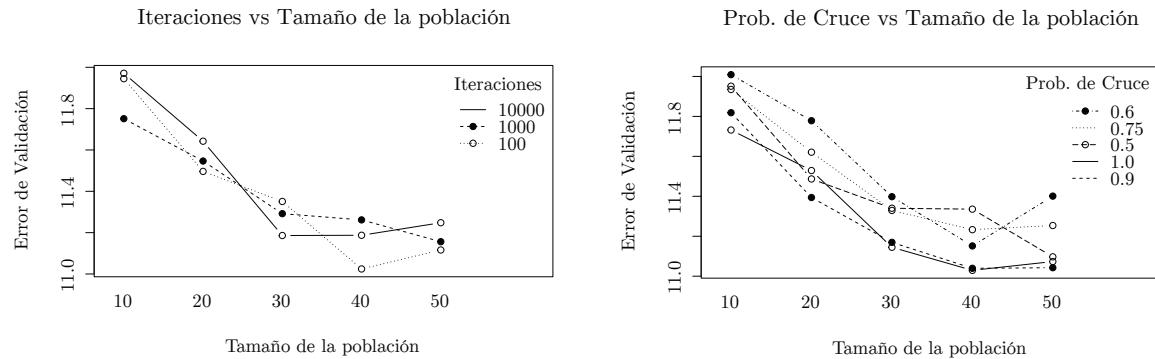


FIGURA A.1: Resultados de entonación de GGA

A.2. Entonación de SGA

SGA depende de los mismos parámetros que GGA. Por esta razón, se evalúan los mismos valores, descritos en la tabla A.2. A partir de estos niveles de evaluación, se realiza un estudio factorial entre todos los parámetros.

Al comparar los resultados presentados en la figura A.2, resulta evidente el efecto positivo de aumentar el número de iteraciones en SGA. Sin embargo, a pesar de que los resultados favorecen ampliamente el uso de 10000 iteraciones, esto implica un aumento de un orden de magnitud en el tiempo de ejecución de la metaheurística. Esto lleva

PARÁMETRO	VALORES EVALUADOS				
Iteraciones	100 1000 10000				
Población	10 20 30 40 50				
Prob. de Cruce	0.5	0.6	0.75	0.9	1.0

TABLA A.2: Valores evaluados para los parámetros de SGA.
En negrita los valores fijados por cada parámetro.

a la selección de un número de iteraciones intermedio (1000) que genere soluciones aceptables sin incurrir en altos costos de cómputo.

Adicionalmente, los resultados presentados en función del tamaño de la población usada, llevan a concluir que este parámetro no influye significativamente en el proceso de búsqueda de SGA. Se selecciona el nivel medio, igual a 30 individuos.

Por último, el efecto la probabilidad de cruce es poco claro. Se decide seleccionar una probabilidad de cruce igual a 1.0 para asegurar la generación de nuevos individuos en cada iteración.

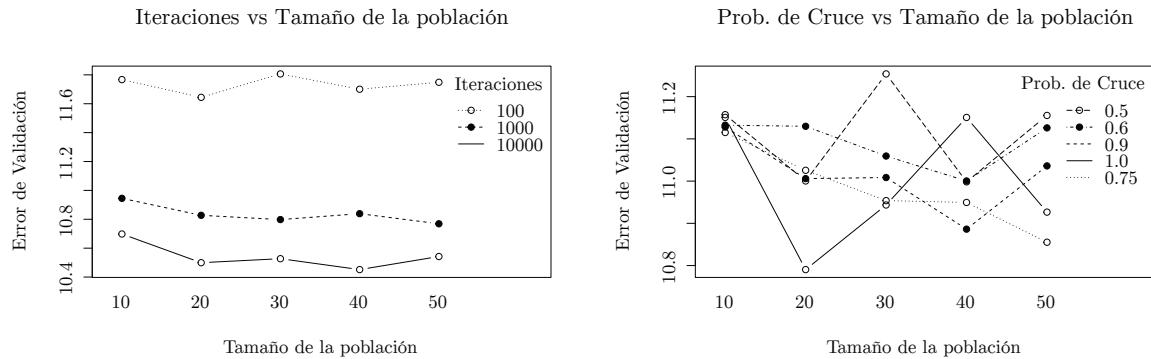


FIGURA A.2: Resultados de entonación de SGA

A.3. Entonación de CHC

CHC es la metaheurística con menos parámetros que entonar. Tan solo depende del número de iteraciones y el tamaño de la población. Los valores evaluados son descritos en la tabla A.3.

Los resultados presentados en la figura A.3 son claros y las conclusiones evidentes. El comportamiento de la metaheurística luego de 1000 o 10000 iteraciones es muy

PARÁMETRO	VALORES EVALUADOS			
Iteraciones	100	1000	10000	
Población	10	20	30	40

TABLA A.3: Valores evaluados para los parámetros de CHC.
En negrita los valores fijados por cada parámetro.

similar, y es poco dependiente del tamaño de la población. Se escoge un tamaño de población medio (igual a 30) y un número de iteraciones igual 1000 para reducir el tiempo de ejecución.

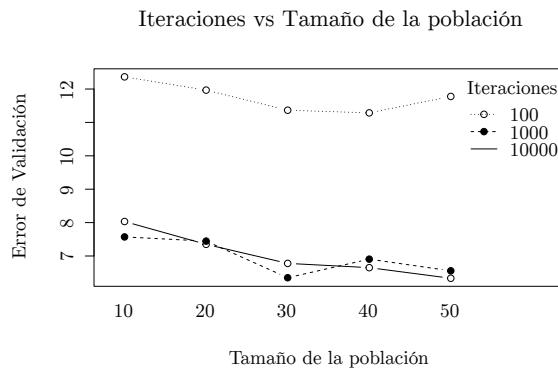


FIGURA A.3: Resultados de entonación de CHC

A.4. Entonación de PBIL

La entonación de PBIL resulta más complicada, debido a que depende de un mayor número de parámetros. En la tabla A.4 se describen los valores evaluados para cada parámetro. Se excluye de este estudio el valor de *Mutation Shift*, que se mantiene en un valor bajo (igual a 0.01). Con los niveles descritos por cada parámetro, se realiza un estudio factorial.

PARÁMETRO	VALORES EVALUADOS			
Iteraciones	100	1000	10000	
Población	10	20	30	40
Learning Rate	0.1	0.05	0.01	
Negative Learning Rate	0.075	0.05	0.01	

TABLA A.4: Valores evaluados para los parámetros de PBIL.
En negrita los valores fijados por cada parámetro.

De los resultados obtenidos (ver figura A.4), se derivan las siguientes conclusiones:

- El tamaño de la población y el número de iteraciones influyen significativamente en la disminución del error de validación. Aplicando 1000 iteraciones, los mejores resultados son obtenidos con poblaciones de tamaño 40.
- Existe una relación importante entre los valores de *Learning Rate* y el *Negative Learning Rate*. La combinación de 0.1 y 0.01 respectivamente, prueba ser la mejor alternativa. Esta conclusión se ratifica al evaluar la interacción entre ambos parámetros y el número de iteraciones.

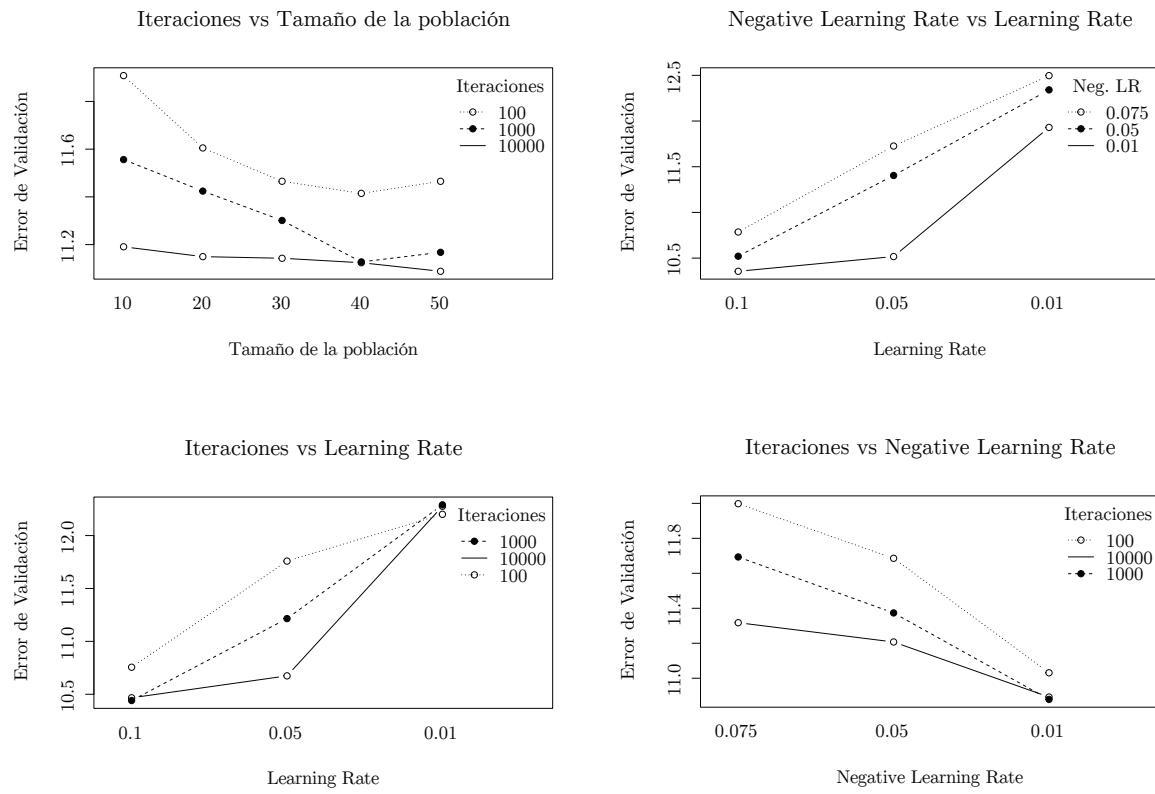


FIGURA A.4: Resultados de entonación de PBIL

A.5. Entonación de PSO

En la tabla A.5 se describen los valores utilizados para evaluar el impacto de algunos de los parámetros de los que depende PSO. El resto de los parámetros fueron

fijados con la finalidad de limitar el impacto en tiempo sobre el algoritmo. PSO se vio especialmente afectado en tiempo de ejecución por el número de iteraciones, por lo que se decidió fijar este parámetro en 1000 al igual que el resto de las metaheurísticas. Adicionalmente, debido a que el número de soluciones generadas por PSO en cada iteración es igual al número de partículas por el tamaño de la población, se fijó este último parámetro en 5 individuos para evitar un aumento considerable en el tiempo de ejecución y evaluar el impacto del aumento de las partículas. Por último, las constantes c_1 y c_2 se fijan en función de la velocidad máxima (*i.e.* $\frac{v_{\max}}{2}$) pues no tiene sentido usar valores más altos dado que el vector de velocidad está acotado por v_{\max} .

PARÁMETRO	VALORES EVALUADOS		
Partículas	5	10	15
Inercia (w)	0.9	0.6	0.3
Velocidad Máxima (v_{\max})	0.5	0.2	0.05

TABLA A.5: Valores evaluados para los parámetros de PSO.
En negrita los valores fijados por cada parámetro.

Para los valores de Inercia y Velocidad Máxima, se realizó un estudio factorial, y los resultados son presentados en la figura A.5. Resulta clara la interacción entre diferentes valores de estos parámetros, donde la combinación de una inercia igual a 0.9 y una velocidad máxima de 0.2 logran los mejores resultados.

El número de partículas fue evaluado de forma independiente. Los resultados obtenidos muestran que no existen diferencias en el uso de diferentes números de partículas. Este parámetro se fija en 5 partículas con la finalidad de disminuir el tiempo de ejecución del algoritmo.

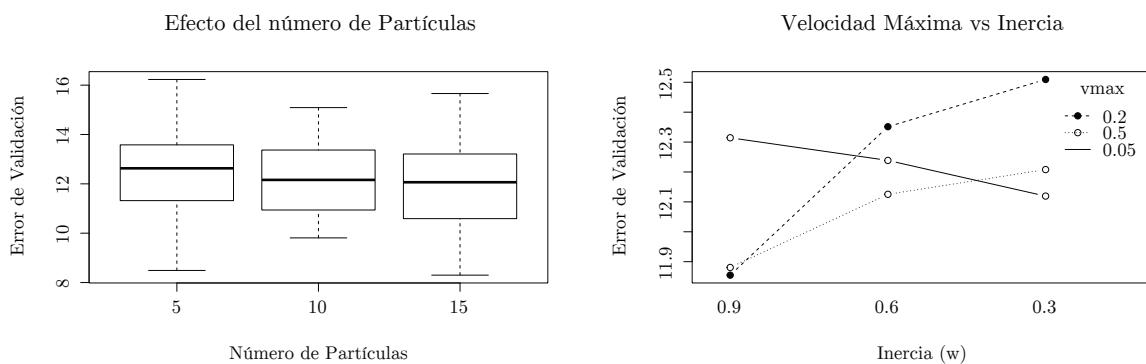


FIGURA A.5: Resultados de entonación de PSO