

# Problema de selección de prototipo (IS/PS): Un enfoque con metaheurísticas

Juan Carlos Arocha Ovalles

Matteo José Ferrando Briceño

## Abstract

El uso de clasificadores para determinar la clase de algún *documento* está intimamente relacionado con el conjuntos de datos *TR* con el que son entrenados. Un *TR* que represente adecuadamente a la muestra contiene potencialmente una gran cantidad de instancias, lo que genera un entrenamiento y una clasificación lenta. El objetivo de este estudio es encontrar, haciendo uso de metaheurísticas, un subconjunto de *TR* considerablemente más pequeño que mantenga el mismo desempeño de clasificación.

Entre las metaheurísticas probadas, tenemos Búsqueda Local, Búsqueda Local Iterada, GRASP y algoritmos evolutivos como GGA (Algoritmo Genético Generacional) y SGA (Algoritmo Genético Estacionario), y dentro de ellas, se hizo uso de una función objetivo exponencial que prioriza la clasificación sobre la reducción para comparar soluciones, y de una perturbación inteligente, la cual prioriza aquellos puntos que potencialmente mejoren más la solución.

Se hicieron una serie de experimentos sobre las metaheurísticas antes mencionadas donde se concluyó que para las metaheurísticas de trayectoria, GRASP fue superior mostrando una buena reducción para conjuntos de datos pequeños pero cierta inestabilidad con conjunto de datos grandes, y para las metaheurísticas poblacionales, GGA mostró superioridad reduciendo al mismo nivel que GRASP pero con estabilidad y menores tiempos de ejecución.

## 1. Problema de selección de prototipo

En el campo de clasificación de documentos, la calidad de clasificadores como *K-NN* depende del *TR* con el cual sean entrenados. Un problema que se encuentra a la hora de elegir el *TR* es la selección de las instancias que lo conforman. Una gran cantidad instancias que representen las diferentes clases, pueden llevar a una buena clasificación, pero a su vez llevan a un entrenamiento y a un desempeño lento por parte de los clasificadores.

Es por ello que se ha intentado reducir este conjunto para disminuir el tiempo de entrenamiento y clasificación, pero manteniendo la calidad de los resultados que se obtengan.

Este problema es bastante común en diversas áreas de

ciencias de la computación, por ejemplo, procesamiento y etiquetado de imágenes, análisis de *big data*, análisis de lenguaje natural, *Data Mining*, entre otros por lo que ha sido bastante estudiado y existen diversas soluciones propuestas.

## 2. Trabajos anteriores

Entre las soluciones que se estudiaron en el estado del arte, en [5] Toussaint describe soluciones de tipo *Greedy* como CNN, RNN y MCNN, las cuales demuestran una baja complejidad de tiempo y cuyos resultados son aceptables pero no optimales. Cano y Herrera en [2], proponen utilizar algoritmos evolutivos solucionar el problema de selección de instancias para extracción de conocimientos en bases de datos (KDD), presentan una función de evaluación de calidad que se utilizaron para las pruebas; finalmente, concluyen que para KDD, los algoritmos evolutivos mejoran tanto la reducción del subconjunto, como la presición. Luego García y Derrac, con apoyo de Cano y Herrera, en [3] hacen un análisis extenso de los diversos algoritmos basados en el algoritmo *K-NN*.

## 3. Metaheurísticas

Como se vió en [2], una forma de resolver el problema IS/PS es usando metaheurísticas. Una metaheurística es un método genérico de solución de problemas computacionales, que no toma en cuenta el enunciado del problema sino la representación de sus soluciones, para luego mejorarlas en un tiempo eficiente pero que no asegura la optimalidad.

La mayoría de las metaheurísticas parten de una solución inicial, y a través de métodos iterativos, mejoran dicha solución, manteniendo la mejor de todas vista hasta el momento. La condición de parada de las mismas suelen variar entre un límite de tiempo, de iteraciones o que el espacio de búsqueda no provea mejores soluciones.

Para poder realizar pruebas usando metaheurísticas, se necesita una representación de la solución del problema planteado. En este caso, IS/PS fue representado con un conjunto **SP** que corresponde a las instancias que se encuentran dentro del subconjunto de *TR* seleccionado y el resto de los puntos se encuentra en el conjunto **UP**.

Por otro lado, también es necesario tener una función que sea capaz de evaluar una solución y que de este modo dos soluciones sean comparables. Como se vió en la sección 2, la calidad del subconjunto de TR seleccionado depende de la calidad de clasificación y del tamaño del mismo. La calidad se puede representar como el porcentaje de documentos bien clasificados, y el tamaño como el porcentaje de reducción de TR, lo que genera una función con dos variables. Para los experimentos se propusieron tres funciones:

$$f(cl, rd) = \alpha \times cl + (1 - \alpha) \times rd \quad (1)$$

$$f(cl, rd) = (cl + (1 - \alpha))^2 \times (rd + \alpha)^2 \quad (2)$$

$$f(cl, rd) = \beta^{(cl+(1-\alpha)) \times (rd+\alpha)} \quad (3)$$

Donde  $cl$  corresponde al porcentaje de clasificación correcta,  $rd$  al porcentaje de reducción de TR,  $\alpha$  al peso en porcentaje (entre 0 y 1) que tiene  $cl$  con respecto a  $rd$  y  $\beta$  a un número real arbitrario. La razón por la cual de toman estas funciones es porque se busca penalizar la puntuación obtenida por las mismas cuando cualquier de los dos atributos, sea  $cl$  o  $rd$ , tengan valores muy bajos.

Luego de hacer experimentos, cuyos resultados se pueden ver en 1, utilizando búsqueda local, probando las tres funciones contra un el conjunto de datos con el que se entrenó, se decidió utilizar la *función exponencial*, específicamente con  $\beta = e$  y  $\alpha = 0,9$ , dando un mayor peso a la clasificación que a la reducción.

Función	$f(cl, rd)$	$cl$	$rd$	seg
planar	0.81	1	0.62	76.1
cuadrática	0.42	0.98	0.65	113.11
exponencial	0.52	0.99	0.64	93.62

Cuadro 1: Comparación de resultados de funciones objetivo

Por otra parte, también es necesario definir un operador de vecindad, el cual consiste en una función capaz de generar nuevas soluciones a partir de cualquier solución. El mismo depende de la metaheurística aplicada a problema.

En los experimentos realizados en este estudio, se consideró el uso de búsqueda local como metaheurística lineal y búsqueda local iterada y GRASP como metaheurísticas de trayectoria.

#### 4. Metaheurísticas a implementar

En esta sección se hablará de las metaheurísticas que fueron implementadas con sus respectivas modificaciones para tratar el problema de selección de instancia.

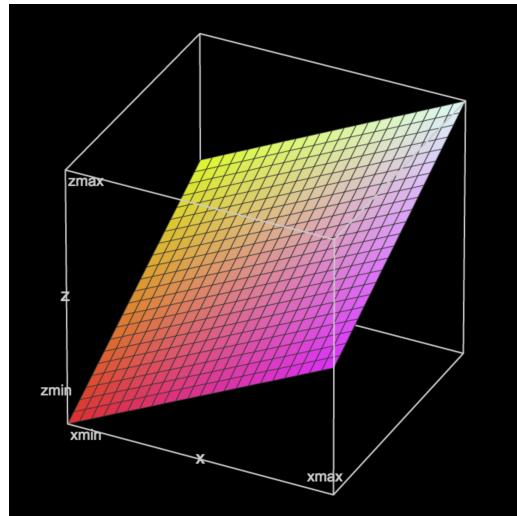


Figura 1: Función planar, ecuación 1

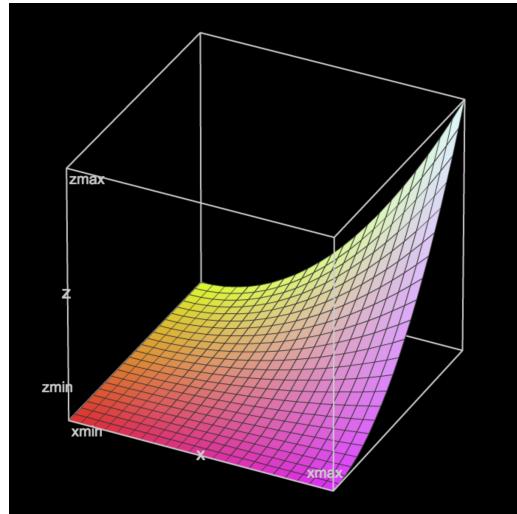


Figura 2: Función cuadrática, ecuación 2

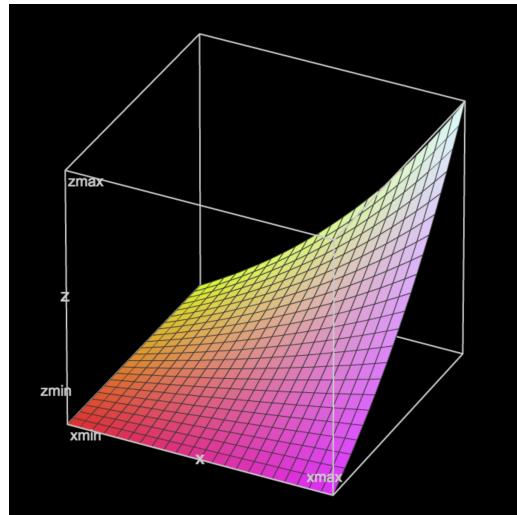


Figura 3: Función exponencial, ecuación 3

A continuación se hablará de la solución inicial a la cual será aplicada la metaheurística, y luego de las metaheurísticas búsqueda local (LS), búsqueda local iterada (ILS), GRASP, y algoritmos evolutivos como GGA y SGA.

#### 4.1. Solución inicial

En este trabajo se consideraron varias soluciones iniciales a implementar:

- Selección de todos los puntos: Se inicia de una solución que contenga todos los puntos del conjunto TR.
- Selección aleatoria: Se inicia de una solución que contenga un porcentaje de los puntos de TR seleccionados de forma aleatoria.
- CNN: Se aplica el algoritmo CNN para obtener la solución inicial.
- MCNN: Se aplica el algoritmo MCNN para obtener la solución inicial.

De todas estas se decidió implementar MCNN puesto que la misma es una solución bastante buena y balanceada en cuanto a las clases que contiene. El único inconveniente es que puede generar una solución cercana al óptimo, lo que impide que la metaheurística realice una reducción considerable con respecto al original. Es por esto que se añadió un *deterioro* a esta solución, añadiendo de forma aleatoria un 10% de los puntos del conjunto total a la solución obtenida.

#### 4.2. Búsqueda local (LS)

La búsqueda local consiste en partir de una solución inicial, que va a ser modificada por un operador de vecindad que genera otra solución factible. Luego, la solución actual se compara con la solución generada a través de una función de evaluación de calidad, y se desecha la peor de ambas para recurrir en el mismo proceso con la restante.

Como fue descrito en la sección 3, se tomó la representación de **SPy UP** para las soluciones y las funciones de evaluación se tomaron las funciones 2, 1 y 3. .

Para 3,  $\beta$  tomó el valor del número de *Euler* por la forma en que la función exponencial crece. Como se puede ver en la figura 3, la misma penaliza a los valores más bajos, pues su crecimiento en ese punto es lento y a medida que los valores suben, el valor de la función crece de forma más violenta. En las figuras 2 y 1 se puede notar que 2 y 1 tienen un comportamiento parecido a 3 pero menos pronunciado.

Por otra parte, el operador de vecindad que se definió consiste en una función que intercambia  $K$  instancias entre **SP** y **UP** para generar nuevas soluciones, donde

$K$  es un parámetro a calibrar dependiendo del problema de clasificación.

Para la selección de las instancias a intercambiar, se tomaron en cuenta dos estrategias:

- **Selección aleatoria:** Se seleccionan puntos de forma aleatoria de alguno de los dos conjuntos **SP** o **UP**.
- **Selección en base a costos:** Las instancias a intercambiar entre **SP** y **UP** son seleccionadas dependiendo de cuánto mejoren o empeoren la solución actual. Para determinar dicha medida, se define el *costo incremental*  $C(e)$  como una estimación del porcentaje de mejora a la solución al insertar la instancia  $e$  a la misma y el *costo decremental*  $D(e)$  como una estimación de mejora a la solución al extraer la instancia  $e$  de la misma.

Para el caso específico de IS/PS, se tomaron dos definiciones de las función de costo incremental y decremental.

- **Prueba por fuerza bruta:** En el caso de la función incremental  $C(e)$ , el valor se obtiene con la diferencia entre la evaluación de la calidad de la solución que contiene al punto  $e$  y la evaluación de la calidad de la solución que no contiene al punto  $e$ . Y en el caso de la función decremental  $D(e) = -C(e)$ .
- **Variación de posición del centroide:** En este caso se define la función incremental  $C(e)$  como la variación de la posición del centroide de la clase de  $e$ , cuyo valor se considera mejor si es mayor. Y definimos la función decremental  $D(e)$  igual que  $C(e)$  pero en este caso su valor se considera mejor mientras sea menor.

La estrategia por fuerza bruta es la más convencional de las dos, puesto que obtiene el costo a partir de la función de evaluación definida para la búsqueda local y por ende asegura que la perturbación mejore la solución anterior. Pero por otro lado, se trata de un algoritmo con un orden de ejecución alto para la cantidad de veces que se podría repetir.

La estrategia de la distancia al centroide se trata de insertar puntos a la solución que modifiquen de una manera significativa la posición del centroide actual. Esto es porque una instancia que no modifique la posición del centroide de forma significativa, es una instancia cuyo valor no influye tanto a la hora de clasificar.

#### 4.3. Búsqueda local iterada (ILS)

La búsqueda local iterada se trata de realizar una búsqueda local sobre una solución inicial para obtener

**Algorithm 1:** Operador de vecindad con selección aleatoria

**Input:** SP, UP : Conjunto de puntos, K : Entero  
**Output:** SP', UP'

```
1 foreach 1 ... K do
2   sacar ← booleano_al_azar()
3   if sacar then
4     punto ← punto_al_azar(SP)
5     SP' ← SP \ {punto}
6     UP' ← UP ∪ {punto}
7   else
8     punto ← punto_al_azar(UP)
9     UP' ← UP \ {punto}
10    SP' ← SP ∪ {punto}
```

**Algorithm 2:** Operador de vecindad con selección en base a costos

**Input:** SP, UP : Conjunto de puntos  
**Output:** SP', UP'

```
1 foreach 1 ... K do
2   sacar ← booleano_al_azar()
3   if sacar then
4     punto = argmin{C(e)|e ∈ SP}
5     SP' ← SP \ {punto}
6     UP' ← UP ∪ {punto}
7   else
8     punto = argmax{C(e)|e ∈ UP}
9     UP' ← UP \ {punto}
10    SP' ← SP ∪ {punto}
```

una mejor solución, la cual se perturba para generar una nueva solución inicial que se utiliza para realizar el mismo proceso. La solución perturbada puede ser mejor o peor que la solución generada por la búsqueda local, dado que lo que se quiere obtener al realizar dicha perturbación es expandir el espacio de búsqueda (lo que puede generar una solución peor pero que al realizar la búsqueda local sobre ella, lleve a un valor mejor que el encontrado) o mejorar aún más la solución actual.

#### 4.4. Greedy Randomized Adaptive Search Procedures (GRASP)

GRASP (por sus siglas en inglés *Greedy Randomized Adaptive Search Procedures*) es una metaheurística de trayectoria que hace uso de un algoritmo *Greedy* aleatorio para generar soluciones diversas y de esta forma recorrer la mayor cantidad de vecindades del espacio de búsqueda. En cada iteración se hace uso de dicho algoritmo para generar una solución aleatoria y luego a esta se le aplica una búsqueda local para intentar mejorarlala (Véase el algoritmo 3). Cabe destacar que toda solución es generada desde cero y no depende de ninguna solución anterior, por lo que se pueden explorar distintas vecindades y llegar a un mejor resultado.

**Algorithm 3:** Algoritmo GRASP

**Input:** N, Semilla : Entero  
**Output:**  $S_{best}$

```
1 fbest ← ∞
2 foreach 1 ... N do
3   S ← AlgoritmoGreedyAleatorio(Semilla)
4   S ← BusquedaLocal(S)
5   if f(S) < fbest then
6     Sbest ← S
7     fbest ← f(S)
```

La implementación en general es simple, pero lo que hace que sea efectivo o no es la generación de las soluciones aleatorias. La función que realiza esta tarea define un costo incremental para cada punto dentro del conjunto **UP** para decidir cuál es el mejor candidato para ser añadido a la solución **SP**. En el caso de IS/PS, se tomó la función *Variación de posición del centroide* definida en la sección 4.2. Véase el algoritmo 4 para más información.

#### 4.5. Algoritmos evolutivos

Los algoritmos genéticos son metaheurísticas que hacen alusión a la metáfora de la evolución biológica natural. Hacen uso del concepto de población de individuos (los cuales representan soluciones potenciales al problema) que luego son sometidos a operadores como la selec-

**Algorithm 4:** Algoritmo Greedy Aleatorio

---

```

Input: Semilla : Entero
Output:  $S_{best}$ 
1  $Candidatos \leftarrow ConjuntoAleatorio()$ 
2  $EvaluarCostoIncremental(e) \forall e \in Candidatos$ 
3 while  $Candidatos \neq \emptyset$  do
4    $C_{min} \leftarrow \min\{C(e) | e \in C\}$ 
5    $C_{max} \leftarrow \max\{C(e) | e \in C\}$ 
6    $RCL \leftarrow \{e \in C | C(e) \leq C_{min} + \alpha(C_{max} - C_{min})\}$ 
7    $s \leftarrow Aleatorio(RCL)$ 
8    $S_{best} \leftarrow S_{best} \cup \{s\}$ 
9    $Candidatos \leftarrow Candidatos \setminus \{s\}$ 
10   $EvaluarCostoIncremental(e) \forall e \in Candidatos$ 

```

---

ción, la recombinación y la mutación para evolucionar los individuos de dicha población, es decir, mejorar las soluciones que estos representan. Más específicamente:

- **Mutación:** Se trata de un operador cuya función principal es aportar nueva información a la población al modificar los genes de las soluciones intermedias y de esta manera poder explorar diferentes espacios de búsqueda. El mismo simula la aparición de nuevos genes en la población que aumentan la posibilidad de supervivencia.
- **Recombinación/Cruce:** Este operador permite el intercambio de información entre dos o más individuos de la población para generar unos o más individuos hijos. Simula la reproducción de los individuos.
- **Selección:** Este operador permite definir qué individuos van a participar en la recombinación o cruce y a su vez, los genes que pasarán a las siguientes generaciones. Simula el proceso de selección natural.

En nuestro caso particular, la mutación consiste en seleccionar de forma aleatoria (con misma probabilidad) si se va a insertar o quitar de la solución un punto cualquiera, repitiendo este proceso para un 5 % del conjunto total de individuos, la selección es completamente aleatoria, y el algoritmo de recombinación/cruce utilizado es el de recombinación en un punto; tomamos dos individuos como *padres*, se elige aleatoriamente un punto de corte en la longitud de ambos padres; los *hijos* resultan de combinar la sección izquierda del corte de un padre con la sección derecha del corte del otro, y viceversa.

El esquema más tradicional que define el uso de estos operadores son los *Algoritmos Genéticos*. A continuación se definen dos algoritmos: GGA y SGA

**4.5.1. Algoritmo Genético Generacional (GGA)**

Este algoritmo es el más apegado a la metáfora evolutiva. El mismo parte de una población de tamaño  $n$  para generar una población descendente tomando el mejor individuo entre ellas como solución. Esto lo hace seleccionando dos individuos de la población original para cruzarlos y generar dos hijos que formarán parte de la población hija, repitiendo este proceso hasta que la misma alcance el mismo tamaño que su predecesora. Ver algoritmo 5

**Algorithm 5:** Algoritmo Genético Generacional

---

```

Input: n tamaño de la población, cp probabilidad de cruce, mp probabilidad de mutación
Output: Solución al problema
1  $P \leftarrow$  Generar solución aleatoria de  $n$  individuos
2  $s^* \leftarrow$  mejor individuo de  $P$ 
3 while  $\neg$  Condición de parada do
4    $P' \leftarrow \emptyset$ 
5   while  $|P'| < n$  do
6      $p1 \leftarrow$  Seleccionar un individuo en  $P$ 
7      $p2 \leftarrow$  Seleccionar un individuo en  $P$ 
8      $c1, c2 \leftarrow$  recombinar  $p1$  y  $p2$  con probabilidad cp
9     Mutar  $c1$  y  $c2$  con probabilidad mp
10     $P' \leftarrow P' \cup \{c1, c2\}$ 
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  $s^*$ 

```

---

**4.5.2. Algoritmo Genético Estacionario (GGA)**

Como se dice en [6], SGA (*Steady-State Genetic Algorithm*) es un enfoque distinto en el esquema de los algoritmos evolutivos dado que el mismo no sigue un esquema generacional en el que se genera una nueva población en cada paso del algoritmo. SGA comienza con una población de tamaño  $n$  y en cada iteración se produce un máximo de dos nuevos individuos, y no una población completa como lo hace GGA.

En cada iteración se eligen dos individuos padres mediante el operador de selección, se generan uno o dos hijos mediante el cruce, se mutan a través del operador de mutación y finalmente se sigue una estrategia de selección para reemplazar uno o dos individuos de la población actual por la nueva descendencia, lo que mantiene el tamaño  $n$  de la población. Ver algoritmo 6

**Algorithm 6:** Algoritmo Genético Estacionario

---

**Input:** n tamaño de la población, cp probabilidad de cruce, mp probabilidad de mutación  
**Output:** Solución al problema

- 1  $P \leftarrow$  Generar solución aleatoria de  $n$  individuos
- 2  $s^* \leftarrow$  mejor individuo de  $P$
- 3 **while**  $\neg$  Condición de parada **do**
- 4   |  $p1 \leftarrow$  Seleccionar un individuo en  $P$
- 5   |  $p2 \leftarrow$  Seleccionar un individuo en  $P$
- 6   |  $c1, c2 \leftarrow$  recombinar  $p1$  y  $p2$  con probabilidad cp
- 7   | Mutar  $c1$  y  $c2$  con probabilidad mp
- 8   | Reemplazar dos individuos en  $P$  por  $c1$  y  $c2$  según algún criterio
- 9   | **if** El mejor individuo en  $P$  es mejor que  $s^*$
- 10   | **then**
- 11   |   |  $s^* \leftarrow$  el mejor individuo en  $P$
- 12 **return**  $s^*$

---

## 5. Evaluación experimental

En esta sección se hablará de la configuración usada sobre los experimentos para probar y comparar las metaheurísticas búsquedas local, búsquedas local iteradas, GRASP, GGA y SGA y de los resultados arrojados por las mismas.

### 5.1. Ambiente de prueba

Es importante definir un ambiente de pruebas igual para todos los experimentos que se vayan a realizar para que sea posible la comparación entre ellos, sobre todo en términos de tiempo. En este trabajo se realizaron pruebas en una máquina que corre sobre Debian Wheezy con procesador Intel Core i7 de 3.4 Ghz y 8 GB de memoria RAM.

### 5.2. Conjunto de datos

Un factor importante para la evaluación de las diferentes metaheurísticas es el conjunto de datos a utilizar, la distribución de los datos, el número de instancias, el número de atributos de cada una de ellas y la cantidad de datos ruidosos ya que los mismos influyen en el espacio de búsqueda y por ende en la efectividad del algoritmo.

Los conjuntos de datos usados para estos experimentos pertenecen originalmente a *UCI Machine Learning Repository* [4] pero se obtuvo una modificación de los mismos basada en la estrategia de validación cruzada en 10 grupos (*10-fold cross-validation*) del *KEEL Data-Mining Software Tool* [1]. Se usaron 6 conjuntos para realizar dichas evaluaciones, separando los datos en tres

grupos según el número de instancias. En el cuadro 2 se puede ver con más detalle los conjuntos de datos pequeños, en el cuadro 3 los medianos y en el cuadro 4 los grandes.

Conjunto	Instancias	Atributos	Clases
Glass	214	9	7
Iris	150	4	3

Cuadro 2: Conjuntos de datos pequeños

Conjunto	Instancias	Atributos	Clases
WDBC	596	30	2
Pima	768	8	2

Cuadro 3: Conjuntos de datos medianos

Conjunto	Instancias	Atributos	Clases
Segment	2310	19	7

Cuadro 4: Conjunto de datos grande

### 5.3. Parámetros

En las diferentes metaheurísticas hay parámetros que regulan el proceso de búsqueda que realizan. Los valores dados a cada parámetro y la combinación entre ellos determinará el comportamiento y el desempeño de cada algoritmo y por lo tanto estos son altamente dependientes del problema a resolver. Esto hace que la regulación de estos parámetros no sea un proceso trivial y que conlleve un análisis con pruebas sobre los mismos para asegurar el mejor desempeño.

Para el caso de la evaluación de las metaheurísticas descritas anteriormente, se realizó una entonación de los parámetros. En la tabla 5 se muestra los parámetros usados.

Parámetro	LS	ILS	GRASP	GGA	SGA
Iteraciones	1000	1000	1000	100	100
Iteraciones LS	-	100	100	-	-
Classification	0.9	0.9	0.9	0.9	0.9
Greediness	-	-	0.5	-	-
Población	-	-	-	50	50
Prob. cruce	-	-	-	1.0	1.0
Prob. mutación	-	-	-	0.5	0.5
% perturbaciones	-	-	-	0.1	0.1

Cuadro 5: Parámetros entonados para las metaheurísticas

*Classification* se refiere al peso de la clasificación sobre la reducción, es decir,  $\alpha$  en la ecuación 3, *Greediness* es qué tan *Greedy* es el algoritmo *Greedy* aleatorio

usado en GRASP (mientras se acerca a 1, se hace más *Greedy*), y el % de perturbaciones es el porcentaje sobre el número total de puntos del conjunto que define cuantas perturbaciones se harán para modificar los descendientes generados con el operador de Mutación.

#### 5.4. Resultados

En esta sección se muestran los resultados obtenidos luego de hacer la evaluación experimental. Se harán comparaciones de las metaheurísticas en cada uno de los conjunto de datos utilizados por separado.

Se tomaron 10 ejecuciones de los 10 grupos de validación cruzada y se promediaron en las tablas 6 , 7 , 8 , 9 y 10 tomando en cuenta porcentaje de clasificación, porcentaje de reducción y tiempo de ejecución.

Conjunto	Clasificación	Reducción	Tiempo (seg)
Glass	0.64	0.58	1.4
Iris	0.94	0.89	0.2
WDBC	0.92	0.87	69.4
Pima	0.64	0.56	151.3
Segment	0.96	0.77	214.8

Cuadro 6: Resultados Búsqueda Local

Conjunto	Clasificación	Reducción	Tiempo (seg)
Glass	0.68	0.65	56
Iris	0.97	0.96	11
WDBC	0.92	0.97	196
Pima	0.65	0.59	661
Segment	0.95	0.9	786.2

Cuadro 7: Resultados Búsqueda Local Iterada

Conjunto	Clasificación	Reducción	Tiempo (seg)
Glass	0.68	0.93	56
Iris	0.95	0.96	24
WDBC	0.93	0.99	72
Pima	0.73	0.99	46.03
Segment	0.85	0.88	335.6

Cuadro 8: Resultados GRASP

Conjunto	Clasificación	Reducción	Tiempo (seg)
Glass	0.65	0.9	2.17
Iris	0.93	0.97	1.23
WDBC	0.92	0.98	16.3
Pima	0.74	0.99	13.2
Segment	0.87	0.9	122.486

Cuadro 9: Resultados GGA

Se puede ver una tendencia a mantener el porcentaje de clasificación para LS, ILS y GRASP lo que se puede

Conjunto	Clasificación	Reducción	Tiempo (seg)
Glass	0.6	0.78	6.66
Iris	0.89	0.89	2.71
WDBC	0.91	0.92	76.83
Pima	0.67	0.88	74.13
Segment	0.91	0.85	592.08

Cuadro 10: Resultados SGA

deber al 0.9 que se le da de peso a la clasificación en la función objetivo. Pero por otro lado, en general, GRASP muestra una reducción muy superior con respecto a LS e ILS en los conjuntos de datos pequeños (Glass, Iris, Pima y WDBC), lo que sugiere que la función *Greedy* aleatoria está haciendo una buena selección de los candidatos a utilizar, por lo que se puede concluir que la función de costo incremental basada en la distancia del centroide es una buena opción para esta metaheurística con conjuntos pequeños.

En cuanto a los tiempos de ejecución, ILS es más lento que LS y GRASP a medida que el conjunto de datos aumenta el tamaño, y estas dos últimos tienen tiempos de ejecución aceptables.

Con respecto a las metaheurísticas poblacionales, GGA arrojó resultados favorables. Tiene una reducción mucho mayor que LS e ILS y más estabilidad que GRASP cuando se usan conjuntos de datos grandes. Por otra parte, siendo GRASP la única metaheurística de trayectoria comparable en términos de resultados, la misma se ejecuta en un tiempo mucho mayor que GGA, lo que indica que GGA es una mejor opción para resolver el problema de selección de instancia.

Por otro lado, en las figuras 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, , podemos observar una comparación de las metaheurísticas por conjunto de datos. En la clasificación, se ve que siempre se solapan las cajas en el diagrama, lo que indica cierta similitud en los resultados, pero si se detalla la figura 12, GRASP no parece ser muy estable pues la clasificación es muy dispersa en comparación de LS e ILS.

Con respecto a la reducción, se puede notar una superioridad de GRASP sobre LS e ILS, pero de nuevo siendo poco estable en el conjunto segment. Las cajas en los diagramas de GRASP abarcan pocos valores y casi no hay *outliers*.

#### Conclusiones

Podemos concluir que Búsqueda Local es inferior como metaheurística a la hora de resolver el problema de selección de instancias con respecto a las metaheurísticas de trayectoria ILS y GRASP, y las poblacionales GGA y SGA. GRASP arrojó mejores resultados que ILS, pero menor estabilidad, a diferencia de ILS que arrojó resultados más estables pero que se hacían muy

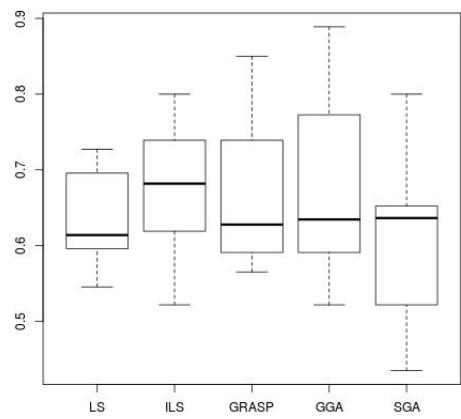


Figura 4: Clasificación para *Glass*

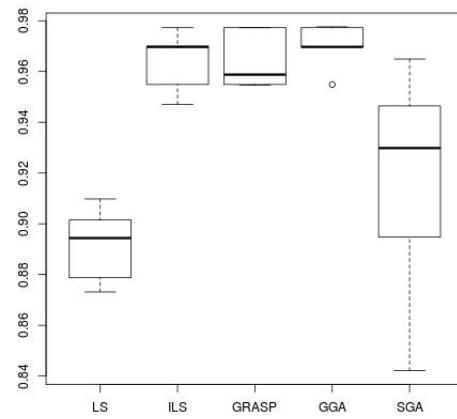


Figura 7: Reducción para *Iris*

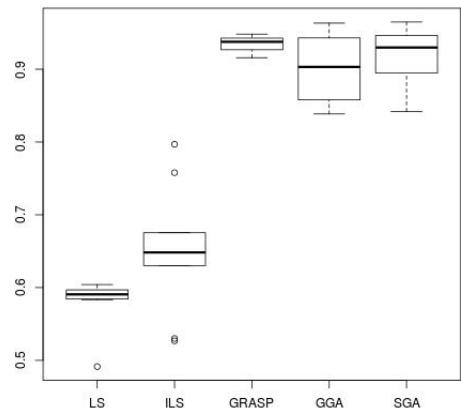


Figura 5: Reducción para *Glass*

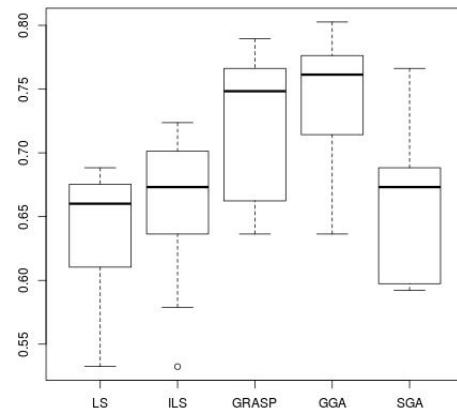


Figura 8: Clasificación para *Pima*

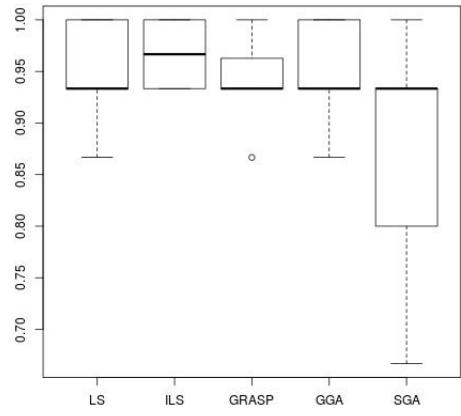


Figura 6: Clasificación para *Iris*

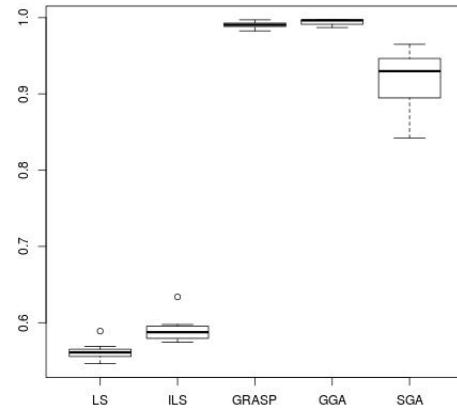
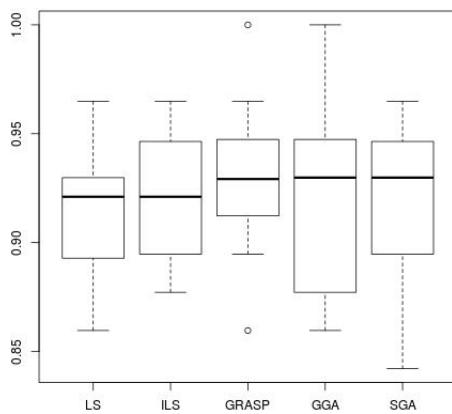
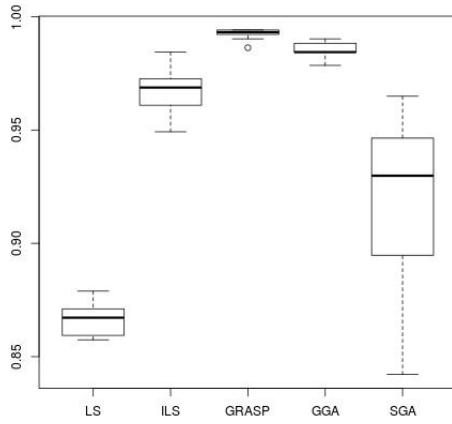
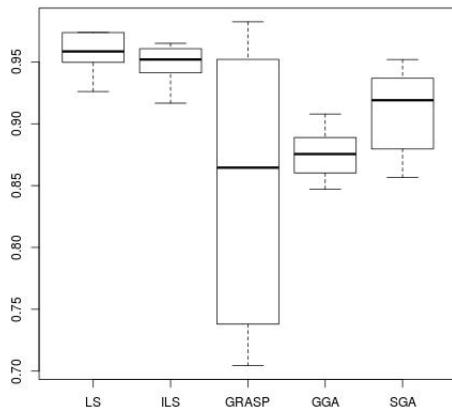
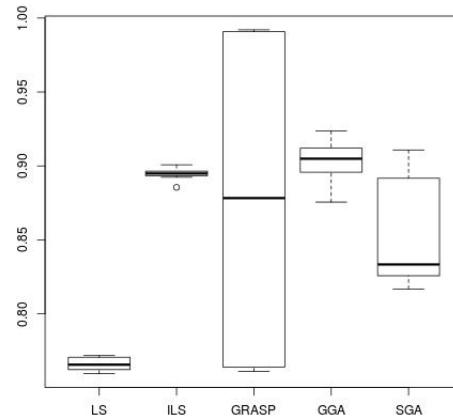


Figura 9: Reducción para *Pima*

Figura 10: Clasificación para *WDBC*Figura 11: Reducción para *WDBC*Figura 12: Reducción para *Segment*Figura 13: Reducción para *Segment*

lentos a medida que el conjunto de datos aumentaba su tamaño. GGA por su parte, arrojó resultados favorables manteniendo estabilidad y tiempos de ejecución muy superiores a GRASP.

Por otra parte, darle un peso mayor a la clasificación que a la reducción mostró resultados favorables dado que las metaheurísticas mantienen el porcentaje de clasificación luego de haber reducido el número de puntos. Finalmente, la función de costo incremental basada en la distancia del centroide, resultó ser positiva para la *perturbación inteligente* de la búsqueda local y el algoritmo *Greedy aleatorio* de GRASP para conjuntos de datos pequeños.

## Referencias

- [1] 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(2-3):255–287, 2010.
- [2] J. Cano, F. Herrera, and M. Lozano. Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study. *Evolutionary Computation, IEEE Transactions on*, 7(6):561–575, Dec 2003.
- [3] S. Garcia, J. Derrac, J. R. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):417–435, 2012.
- [4] M. Lichman. UCI machine learning repository, 2013.
- [5] G. T. Toussaint. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface*, 34, 2002.
- [6] D. Whitley, J. Kauth, and C. S. U. D. of Computer Science. *GENITOR: A Different Genetic Algorithm*. Technical report (Colorado State University. Department of

Computer Science). Colorado State University, Department of Computer Science, 1988.