Escuela Técnica Superior de Ingenierías Informática y de Telecomunicaciones

Máster Oficial en Ingeniería Informática

Curso 2020/2021

CLARANS

Tratamiento Inteligente de Datos

Breve descripción

Explicación del algoritmo de clustering CLARANS

Autor

Álvaro de la Flor Bonilla y Antonio Manuel Salvat Pérez

RESUMEN

Tal y como se expone en el enunciado del problema, los objetivos de esta memoria son dar respuestas a las siguientes cuestiones:

1. Explicar la motivación del diseño del algoritmo CLARANS dada por los autores.
2. Explicación clara y detallada del algoritmo.
3. Explicación del ajuste de parámetros.
4. Agrupación de objetos poligonales convexos
5. Comparación con otros algoritmos.
6. Pequeño ejemplo muestra de la ejecución del algoritmo.

1 ÍNDICE

ÍNDICE DE ILUSTRACIONES

1 MOTIVACIÓN DEL DISEÑO

Este algoritmo ha sido creado por los investigadores Raymond T. Ng y Jiawei Han.

En primer lugar, cabe destacar que el funcionamiento de *CLARA* se basa en la aplicación del algoritmo *PAM* en pequeñas muestras del grafo que se está analizando. Es decir, toma estas pequeñas muestras del grafo de las que hemos comentado anteriormente, ejecuta *PAM* sobre ellas y luego realiza una comparación de los resultados obtenidos para obtener el clustering general.

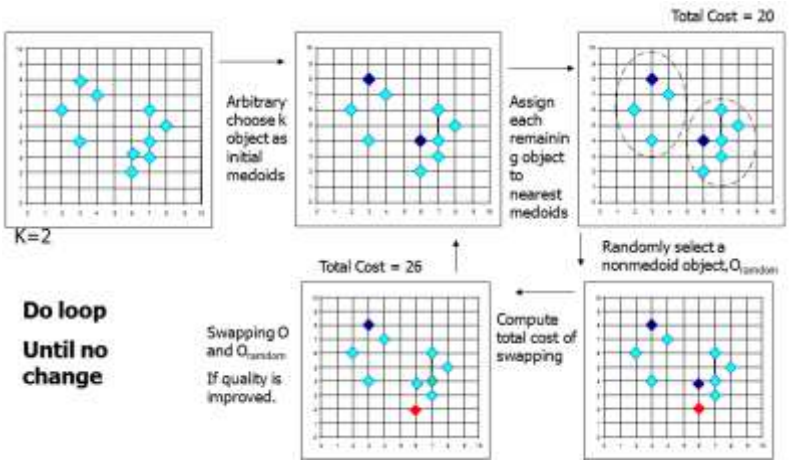

Ilustración 1 - Funcionamiento de PAM

Tras lo anterior, en un primer momento puede parecer solventado el gran problema que lastra a *PAM* que es que en cada paso o iteración se examinan todos los vecinos del nodo actual (recordemos que en cada iteración se actualizan los meloides, se evalúan y se vuelven a recorrer todos los vecinos hasta encontrar la solución final) lo cual conlleva una gran penalización en los casos donde el volumen a analizar es muy grande.

El problema que acarrea *CLARA* es que a pesar de que intenta examinar menos vecinos y restringe la búsqueda en subgrafos que son mucho más pequeños en tamaño que el grafo original, construye estos subgrafos a partir de muestras. Es decir, en primera instancia se construye un primer mapa de subgrafos invariable que se utilizará durante toda la ejecución del algoritmo.

Aunque *CLARA* examine a fondo todos estos subgrafos (mediante *PAM*) el problema radica en que la búsqueda está completamente restringida a esta primera elección de muestreo que se realizó inicialmente.

Como ejemplo, si el nodo mínimo (llamémoslo *M*) no está incluido en uno de los nodos muestreo que se ha generado, nunca se encontrará este en las sucesivas iteraciones, sin importar lo profunda que sea la búsqueda. Habría que recoger muchas muestras y procesarlas.

En resumen, lo que queremos decir es que a pesar de que *CLARANS* mejore muchísimo el rendimiento de *PAM*, restringe la búsqueda a un área localizada, es aquí donde se centra el desarrollo del algoritmo que será expuesto a continuación.

2 EXPLICACIÓN DEL ALGORITMO

CLARANS es un algoritmo de agrupamiento de individuos de un conjunto que tienen características similares entre sí. Este proceso se realiza en base a reglas de asociación por cercanía a un medoide.

En *CLARANS* tenemos dos parámetros:

1. ***maxNeighbor***: número máximo de vecinos que han sido examinados.
2. ***numLocal***: número de mínimos locales.

Mientras más alto sea el valor "*maxNeighbor"* más larga será casa búsqueda de un mínimo local. Pero la calidad de los mínimos locales será mayor y necesitaremos menos de ellos.

El algoritmo busca encontrar cierta aleatoriedad en los pasos de su búsqueda, el grupo obtenido después de sustituirlo a un medoide se llama el vecino del agrupamiento actual. Si durante el proceso el individuo encuentra un vecino mejor, el algoritmo lo traspasa al nodo vecino y comienza de nuevo, si no lo encontrara pararía el agrupamiento que estuviera llevando a cabo y se produciría un óptimo local o Clúster.

2.1 Pasos del algoritmo *CLARANS*

1. Parámetros de entrada "*numLocal"* y *"maxNeighbor"*, inicializamos *"i"* a 1 y *"mincost"* lo establecemos como un número muy grande.
2. Establecemos "*current"* a un nodo arbitrario en $G_{n;k}$.
3. Establecer *"j"* a 1.
4. Considerar un vecino aleatorio S de "*current"* que esté basado en 5 y calcular el coste diferencial de los nodos.
5. Si S tuviera un coste menor, fijaremos *"current"* a S y volveríamos al paso 3.
6. Si el coste fuera mayor o igual incrementamos j en 1. Si $j \leq maxneighbor$ volvemos al paso 4
7. Si no lo fuese, si $j > maxneighbor$, comparamos el costo del current con mincost. Si fuera menor que *"mincost"*, fijamos a mincost el coste del current y establecemos al parámetro "*bestnode"* a *"current"*.
8. Incrementamos i en 1 y si $i > numlocal$, el resultado sería *"bestnode"* y pararíamos, si no lo fuera volvemos al paso 2.

Los pasos del 3 al 6 tienen el objetivo de buscar nodos que vayan costando menos progresivamente. Si tras comparar el nodo actual con el máximo número de vecinos de ese nodo el coste sigue siendo menos, ese nodo será declarado el mínimo local.

En el paso 7 comparamos el coste del mínimo local se compara con el coste menor que hayamos obtenido hasta ese momento. Tras esto *CLARANS* repite lo anterior hasta encontrar otro mínimo local y así sucesivamente hasta que encuentre el *"numLocal"* de mínimos locales.

En la parte de arriba podemos ver ciertas características para compararlo con el algoritmo *PAM*, en CLARANS podemos ver que tenemos dos parámetros *"maxNeighbor"* y *"numLocal"*. Mientras mayor sea el valor de "*maxNeighbor"*, más se acerca este algoritmo a *PAM* y más tiempo se tardará en la búsqueda de un

mínimo local, sin embargo, la calidad de este mínimo será mayor y habrá que encontrar menos mínimos locales.

3 AJUSTE DE PARÁMETROS

El algoritmo CLARANS tiene dos parámetros:

1. ***MAXNEIGHBOR***. Número máximo de vecinos examinados.
2. ***NUMLOCAL***. Número de mínimos locales obtenidos, es decir, la cantidad de iteraciones para resolver el problema.

3.1 Determinación de *MAXNEIGHBOR*

Para realizar el ajuste del parámetro de número de vecinos examinados se realizó el siguiente experimento.

En primer lugar, se utilizaron dos conjuntos de datos, R_{n-k} y T_{n-k}, siendo *"n"* el número de datos del dataset con valores que van desde 100 a 3000 y *"k"* el número de clústeres donde se agrupan estos puntos con valores que van desde 5 a 20.

Cuando en la experimentación se fijó el valor de parámetro a 10 000, la calidad del clustering generado era exactamente igual a la producida por *"PAM"* por lo que *"MAXNEIGHBOR = k(n-k)"*.

Evidentemente, tras la experimentación realizada se comprobó que a cuanto menor fuera el valor de este parámetro produce una agrupación de menor la calidad. La cuestión es el análisis de que valor podría ser válido para obtener una agrupación que cuente con una calidad aceptable.

Tras los sucesivos experimentos se detectó que este valor crítico que se busca atiende a la fórmula *"MAXNEIGHBOR = k(n-k)"*, por lo que para localizar el valor mínimo se utilizó el siguiente razonamiento siendo *"MINMAXNEIGHBOR"* un mínimo definido por el usuario configurando *"MAXNEIGHBOR"*:

1. Si *"k(n-k) <= MINMAXNEIGHBOR"* entonces "*MAXNEIGHBOR = k(n-k)"*
2. En cualquier otro caso *"MAXNEIGHBOR"* es igual al mayor valor de *p*% entre *k(n-k)* y *MINMAXNEIGHBOR*.

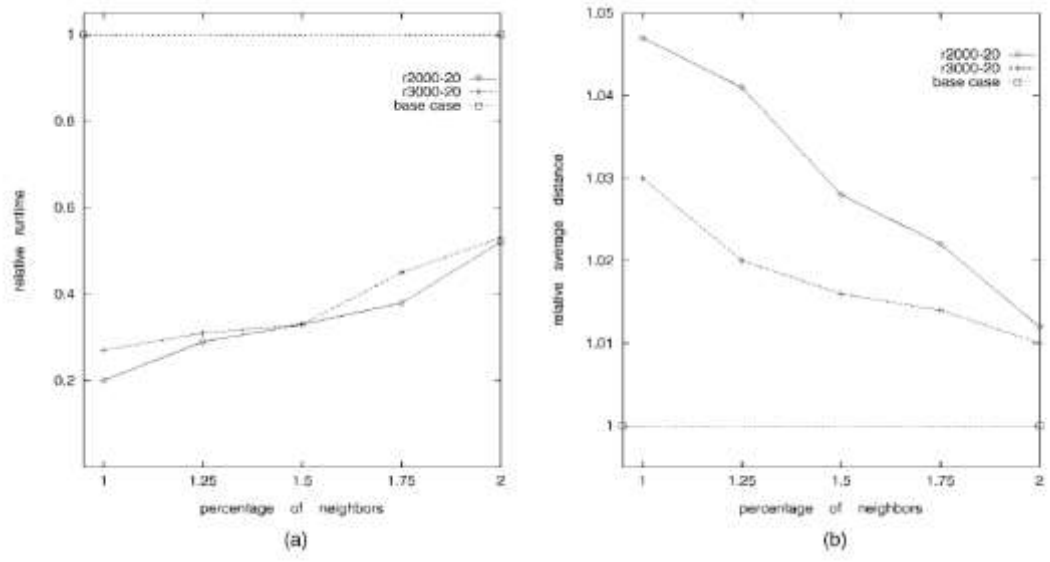

Ilustración 2 – Rendimiento

La imagen anterior es muy representativa ya que se muestra el tiempo de ejecución relativo y la calidad del algoritmo. A medida que se reduce el porcentaje de vecinos visitados menor es el tiempo de ejecución que se necesita y consecuentemente menor es la calidad lograda. Sin embargo, lo que sorprende mucho es que a pesar de que se reduce en un gran número la cantidad de vecinos visitados es apenas apreciable la reducción de calidad consecuente.

De hecho, se comprueba que visitar hasta el 98,5% de más vecinos produce una mejora de calidad marginal, por lo que se establece como valor estándar para todas las experimentaciones la visita del 1.5% o 1.25% de vecinos.

Para entender mejor lo que ocurre utilizaremos el siguiente ejemplo. Considere un nodo *"N"* y dos vecinos *"S_1"* y *"S_2"*. Si en la criba de visitas no se tiene en cuenta el segundo vecino existe una alta posibilidad de que se vuelva a acceder a este nodo vecino a través de las conexiones que puede tener con otro nodo.

3.2 Determinación de *NUMLOCAL*

Como es lógico, debido a que el *NUMLOCAL* determina la cantidad de iteraciones que se van a realizar también penaliza en tiempo, es decir, los tiempos de ejecución son proporcionales al valor de *NUMLOCAL* que se ha utilizado.

Sin embargo, a pesar de que es cierto que cuanto mayor sea el valor de esta variable más penalización en tiempo existe, viene acompañada por una mejora en cuanto a la calidad relativa. Utilicemos los ejemplos dados por los autores para ello.

Las pruebas realizadas por los autores se basaron en utilizar únicamente tres valores para este parámetro: 1, 2 y 3.

Como claramente fue demostrado, hay una mejora sustancial del uso de *NUMLOCAL* con valor 1 y con valor 2. Con esto, se muestra que realizar una segunda iteración y por tanto una segunda búsqueda de mínimo local permite reducir el impacto de la aleatoriedad desafortunada que hablamos anteriormente en el punto 1, es decir aquella aleatoriedad de toma de muestra que lastraba al algoritmo CLARA en el caso de que no fueran escogidas correctamente en primera instancia.

TABLE 1
Relative Runtime and Quality for the Data Set r2000-20

numlocal	1	2	3	4	5
relative runtime	0.19	0.38	0.6	0.78	1
relative average distance	1.029	1.009	1	1	1

Ilustración 3 - Relación coste-calidad del algoritmo

Por otro lado, escoger un valor para *NUMLOCAL* superior a 2 en razón a la experimentación realizada por los autores no es rentable ya que a pesar de que hay un aumento en la calidad de los resultados, es mínima y por tanto no rentable en razón al aumento de coste computacional que supone iniciar nuevas iteraciones. Muestra de ello es la imagen superior, en la que como puede comprobar el coste computacional de pasar del valor 2 al 3 se acerca al doble, sin embargo, la mejora claramente es mínima.

De hecho, tal y como se indica en la documentación de la descripción y justificación del algoritmo, para todos los experimentos que han estudiado para CLARANS han utilizado 2 como valor estándar para el parámetro *NUMLOCAL*.

4 AGRUPANDO OBJETOS POLIGONALES CONVEXOS

Hasta ahora hemos supuesto que todos los objetos se representan como un punto, por lo que hemos podido usar métricas de distancia estándar como la distancia de Manhattan y la euclidiana para calcular la distancia entre objetos, sin embargo, hay numerosos objetos espaciales que podríamos querer agrupar, como edificios, parques, etc. Por lo que la principal cuestión sería como calcular esta distancia de manera eficiente.

Para ello estudiaremos tres enfoques distintos, el primero se basa en calcular la distancia de separación exacta entre dos objetos poligonales, el segundo se basa en usar la distancia mínima entre vértices para aproximar la distancia exacta y el tercero aproxima la distancia usando la distancia de separación entre rectángulos isotéticos.

4.1 Calculo de la distancia de separación exacta

En geometría la distancia entre un punto P y una línea L se define como la distancia mínima, es decir,$\min\{d(P,Q)|\ Q\ es\ un\ punto\ de\ L\}$. Dados dos polígonos A y B podemos decir que la distancia mínima entre cualquier par de puntos sería $\min\{d(P,Q)|P,Q\ son\ puntos\ de\ A\ y\ B\ respectivamente\}$. Esto se llama distancia de separación entre dos polígonos.

Para realizarlo entre polígonos convexos necesitamos dos pasos, el primero sería determinar si tienen alguna intersección. Esto se puede hacer con $O(\log n + \log m)$ Donde n y m determinan el número de vértices de los polígonos A y B, donde un vértice de un polígono es un punto de intersección entre dos bordes de este. Si los polígonos se intersecan la distancia será cero. Si no habría que calcular la distancia entre los dos límites con $O(\log n + \log m)$

El algoritmo elegido para calcular la distancia de separación elegido identifica dos cadenas de vértices y segmentos de línea consecutivos (una cadena de cada polígono) que se “enfrentan” en el sentido de que un vértice de una cadena puede “ver” al menos otro vértice de la otra. Si P es un vértice de A y Q de B, Si P y Q se ven decimos que se ven si la línea del segmento que une P y Q no interseca el interior de los dos polígonos.

Por lo que la distancia de separación de dos polígonos será la distancia mínima entre cualquier par de puntos en las dos cadenas.

4.2 Aproximación por de la distancia mínima entre vértices

Otra forma de aproximar la distancia de separación exacta entre dos polígonos es encontrar la distancia mínima entre los vértices de los polígonos, es decir, $\min\{d(P,Q)|P\ y\ Q\ son\ vértices\ de\ A\ y\ B\ respectivamente\}$ A esta aproximación nos referiremos como MV-Aproximación.

Aunque posea una complejidad superior que el algoritmo descrito antes lo supera generalmente a no ser que el número de n y m sea elevado. La distancia de separación entre dos polígonos no tiene porque ser igual a la distancia mínima entre vértices, sin embargo, la distancia de separación no puede superar la distancia mínima entre vértices por lo que la MV-Aproximación tiende a sobreestimar la distancia de separación real.

Con esta aproximación no es difícil encontrar casos en los que realice una aproximación inexacta, sin embargo, en el estudio realizado en la documentación con numerosos objetos encontrados en mapas reales se da que se realiza una aproximación razonable y que no es muy susceptible a variaciones entre tamaños y formas comparado con la aproximación entre la distancia de los centros.

4.3 Aproximación por la distancia de separación entre rectángulos isotéticos

Otra forma de aproximar la distancia de separación exacta entre dos polígonos A y B es calcular los rectángulos isotéticos y calcular la distancia entre ellos. Dado un polígono A, su rectángulo isotético será el rectángulo mas pequeño que contiene dicho polígono y cuyas aristas son paralelas a los ejes x o y. Nos referiremos a esta aproximación como IR-Aproximación.

Un rectángulo isotético se puede encontrar fácilmente encontrando el máximo y el mínimo del conjunto $\{v \mid v\ is\ la\ coordenada\ x\ del\ vértice\ de\ un\ poígono\}$ y $\{w \mid w\ es\ la\ coordenada\ y\ del\ vértice\ de\ un\ polígono\}$.

El cálculo de la distancia entre dos rectángulos isotéticos requiere dos pasos. El primero será donde se verifica una posible intersección y el siguiente será donde se calcula la distancia de separación real.

Esta aproximación nos otorga una gran ventaja y es que no requiere que el polígono original sea convexo ya que la definición de rectángulo isotético se aplica igual de bien tanto a polígonos convexos como no convexos por lo que no restringiría al algoritmo CLARANS al uso exclusivo en polígonos convexos.

4.4 Eficiencia: distancia exacta vs MV-Aproximación vs IR-Aproximación

Para este estudio se usan polígonos de diferentes vértices y se registra la cantidad promedio de tiempo de ejecución de tiempo de ejecución para calcular la distancia de separación exacta, su MV-Aproximación y su IR-Aproximación.

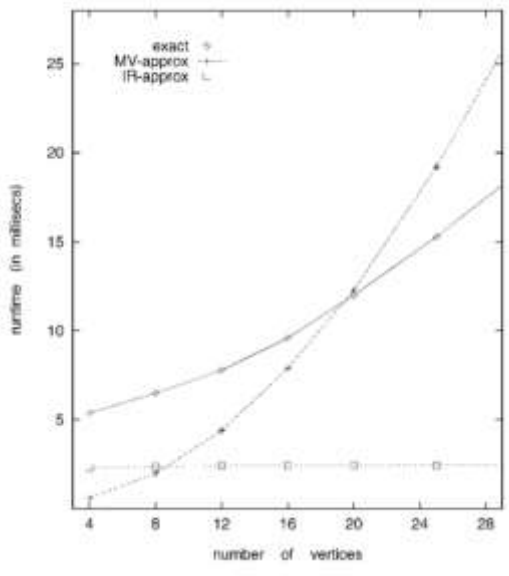

Ilustración 4 - Gráfico de Eficiencia Distancia Exacta - MV-Aproximación - IR-Aproximación

Como vemos el método de IR-Aproximación es un claro ganador ya que supera a los otros dos enfoques ya que destaca mucho su eficiencia cuando el número de vértices aumenta y mantiene una competitividad cuando el número de vértices es pequeño

4.5 Efectividad de agrupamiento: distancia exacta vs IR-Aproximación vs MV-Aproximación

Para ver la efectividad usamos CLARANS con los tres métodos para calcular la distancia entre polígonos.

La calidad de las agrupaciones dadas por la IR-Aproximación y la MV-Aproximación es casi idéntica a la calidad de la agrupación producida utilizando la distancia de separación exacta. Esto muestra claramente que las dos aproximaciones son muy efectivas, ya que subestiman o sobreestiman las distancias reales de manera tan consistente que los grupos se conservan. Por lo tanto, se justifica el uso de la aproximación IR y la aproximación MV como formas de optimizar el rendimiento.

4.6 Eficiencia de agrupamiento: distancia exacta versus aproximación IR versus aproximación MV

Ahora consideraremos la eficiencia de las dos aproximaciones en relación con el enfoque de distancia exacta. La Figura siguiente muestra los tiempos que necesita CLARANS para agrupar un número variable de polígonos que tienen 10 aristas y un número variable de polígonos que tienen entre 4 y 20 aristas. En los dos casos, la IR-Aproximación y la MV-Aproximación superan el enfoque de distancia de separación exacta.

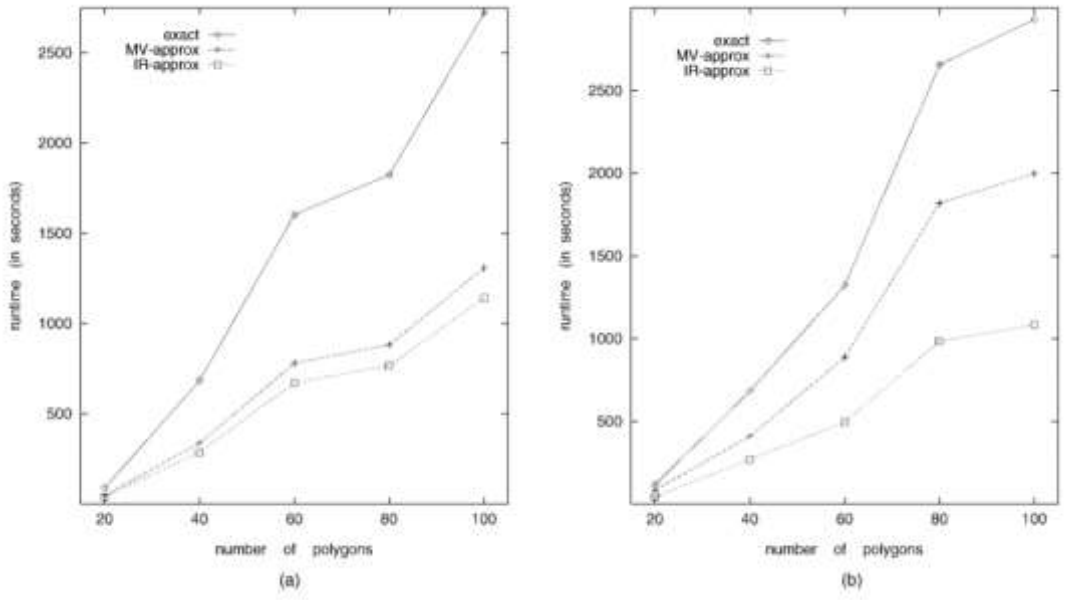

Ilustración 5 - Gráfico de eficiencia de agrupamiento

De hecho, la IR-Aproximación es la más eficiente, ya que solo necesita entre el 30 y el 40 por ciento del tiempo necesario para la aproximación de distancia exacta. Por lo tanto, dado que la aproximación IR es capaz de generar agrupaciones de calidad casi idéntica a las producidas por el enfoque exacto, la aproximación IR es la elección definitiva para CLARANS.

5 COMPARACIÓN CON OTROS ALGORITMOS

Para realizar esta comparativa se tomarán los dos algoritmos previos en los que se basa *"CLARANS"* que son *"PAM"* y *"CLARA"*.

5.1 *CLARANS* vs *PAM*

Como anteriormente expresamos en los apartados anteriores *"PAM"* es completamente ineficiente en las situaciones donde se le obliga a actuar sobre conjunto de datos extensos, de gran tamaño. Lo que si resulta más interesante es realizar esta comparación (entre ambos algoritmos) en pequeños conjuntos de datos.

En el estudio realizado por los autores realzaron la comparativa utilizando para ambos algoritmos conjunto de datos con 40, 60, 80 y 100 puntos en cinco grupos.

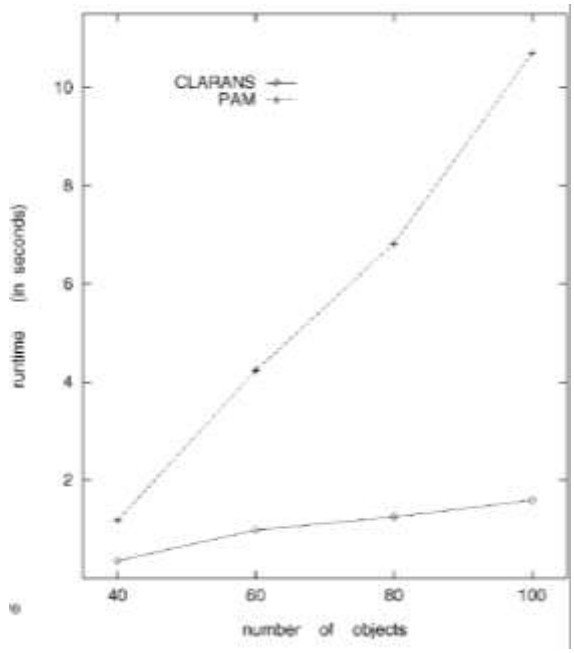

Ilustración 6 - Comparativa CLARANS vs PAM

La figura de arriba muestra el rendimiento de ambos algoritmos usando para ello la comparativa entre el tiempo de ejecución y la calidad de las agrupaciones.

Lo curioso que pueden observar es como, a pesar de que para ambos algoritmos la calidad de los resultados generados es la misma, claramente la mejora en rendimiento (tiempo empleado) por CLARANS es sumamente superior. Esta brecha de rendimiento crece proporcionalmente a medida que el tamaño del conjunto de análisis crece también.

5.2 *CLARANS* vs *CLARA*

En cuanto a esta comparativa, tal y como comentamos en la sección introductoria *CLARA* es una modificación del algoritmo de *PAM* utilizado para en este caso sí, ser utilizado sobre grandes conjuntos de datos.

En el caso de esta experimentación, en la inmensa mayoría de los casos *CLARANS* muestra una mejor calidad de agrupaciones que *CLARA*, sin embargo, también es cierto que en muchas ocasiones *CLARA* es mucho más veloz mostrando sus resultados

finales que *CLARANS*. Por ello, los autores extendieron este estudio a un análisis más exhaustivo a nivel de tiempo, dotando a ambos algoritmos de un mismo período para completar sus cálculos.

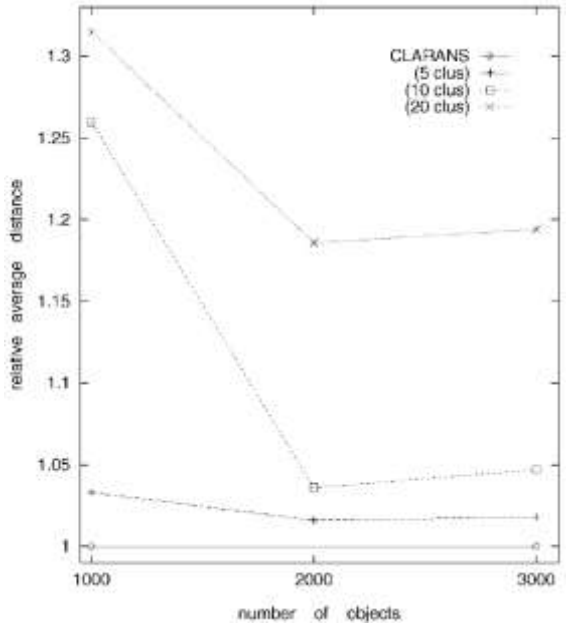

Ilustración 7 - Comparativa *CLARANS* vs *CLARA*

La imagen superior muestra la calidad de los agrupamientos que es capaz de encontrar *CLARA*, normalizados por el valor correspondiente producido por *CLARANS*. En una misma cantidad de tiempo, *CLARANS* es capaz de superar en todas las pruebas a *CLARA*, de hecho, esta brecha diferencial aumenta a medida que se incrementan los agrupamientos utilizados (*"k"*).

El coste de cada iteración de *CLARA* viene dado por la fórmula $O(k^3 + nk)$. Por otro lado, el coste de cada iteración de *CLARANS* es linealmente proporcional al número de objetos. Es por ello por lo que, tras lo anterior, podemos asegurar que un incremento en el número de grupos penaliza en mucha más cuantía a *CLARA* que ha *CLARANS*.

Además, la explicación de la complejidad anterior también justifica el porque si mantenemos un número fijo de clústeres, cuanto mayor es el número de objetos, más estrecha es la brecha entre *CLARANS* y *CLARA*.

6 EJEMPLO PRÁCTICO

6.1 Ejemplo aplicando la metodología

Ahora explicaremos un pequeño ejemplo que hemos obtenido del documento realizado por Carlos Agustín Salvador Alfador donde realiza una descripción del algoritmo en http://repositorio.lamolina.edu.pe/bitstream/handle/UNALM/3345/salvador-alfaro-carlos-agustin.pdf?sequence=3&isAllowed=y

Para este pequeño ejemplo contamos con las notas obtenidas por 5 alumnos y nuestro objetivo será agruparlos según sus resultados.

Comenzaremos con el valor de mínimo coste a 1 y como mejor nodo 0.

minCost = 1

Mejor nodo = 0

datos originales

S0

i	Nota
1	10
2	12
3	15
4	18
5	18

Ilustración 8 - Datos originales ejemplo práctico

Estos serían los datos originales donde el medoide sería 15.

Ahora realizaremos una serie de particiones que se corresponderían a una serie de muestras con un reemplazo dado.

Partición : Muestras con reemplazo

S1

i	Nota
1	10
1	10
2	12
5	18
4	18

S2

i	Nota
2	12
3	15
4	18
5	18
5	18

S3

i	Nota
2	12
2	12
3	15
5	18
1	18

S4

i	Nota
1	18
1	10
3	15
4	18
5	18

Ilustración 9 - Muestras con reemplazo

De aquí obtenemos los medoides siguientes:

- MedoideS1: 12

- MedoideS2: 18
- MedoideS3: 15
- MedoideS4: 18

Ahora realizaremos la diferencia de los absolutos.

	s0	s1	s2	s3	s4
s0	0				
s1	3	0			
s2	3	6	0		
s3	0	3	3	0	
s4	3	6	0	3	0

Ilustración 10 - Diferencia de los absolutos de los conjuntos

Estos resultados salen de la diferencia absoluta entre los medoides de cada muestra, por ejemplo, la diferencia absoluta entre s0 y s2 sería 3 ya que |MedoideS0 – MedoideS2| = |15 – 18| = 3

Como vemos en las diferencias resaltadas tenemos una diferencia de 0 entre s3-s0 y s2-s4. Por lo que sacaríamos los nodos madre de estos dos conjuntos.

	S0∩S3	d	d
2	12	3	3
3	15	0	0
5	18	3	3
1	10	5	5

Ilustración 12 - S0US3

	S4∩S2	d	d
4	18	0	0
5	18	0	0

Ilustración 11 - S4US2

Los nodos hijos serían los siguientes:

		dif
2	12	0
3	15	3
1	10	2

mediana	12

Ilustración 14 - Nodo hijo 1

		dif
4	18	0
5	18	0

mediana	18

Ilustración 13 - Nodo hijo 2

Como vemos se han formado 2 cluster, el primero agruparía los individuos 1,2 y 3 en un nodo hijo con medoide 12 y el otro agruparía a los individuos 4 y 5 con medoide igual a 18.

Por lo tanto, se crearon dos nodos, en el caso de que la diferencia respecto a su medoide sea menor que 1 se finaliza el proceso como pasa en el nodo número dos, sin embargo, con el nodo 1 tendremos que realizar otra vez el paso número 3.

	nodo hijo		S'1		S'2		S'3		S'4	
	2	12	2	12	3	15	1	10	2	12
	3	15	2	12	3	15	2	12	1	10
	1	10	3	15	1	10	3	15	1	10
medoides		12		12		15		12		10

Ilustración 15 - Muestras de reemplazo 2

Diferencia Abs

	S'0	S'1	S'2	S'3	S'4
s0	0				
s1	0	0			
s2	3	3	0		
s3	0	0	3	0	
s4	2	2	5	2	0

Ilustración 16 - Diferencia de absolutos 2

	S0∩S		S3∩S		S3∩S
	1		1		0
2	12	1	10	1	10
3	15	2	12	2	12
		3	15	3	15
mediana	13.5	median a	12	median a	12

Ilustración 17 - Conjuntos finales

Al ser una cantidad de datos pequeña tendremos el mismo resultado por lo que el clúster estará compuesto de los individuos 1,2 y 3.

Por lo que tenemos como resultado un cluster formado por los individuos 1,2 y 3 y otro cluster formado por los individuos 4 y 5. El proceso finaliza hasta que todos los individuos tengan un costo menor frente a su óptimo local.

6.2 Ejemplo en python

Para la realización de un ejemplo práctico vamos a realizar una demostración del algoritmo CLARANS usando el lenguaje de programación Python, a su vez vamos a realizar también una demostración del algoritmo K-Medoids para comparar los resultados de ambos.

Tanto para CLARANS como para K-Medoids hemos usado las librerías pyclustering y sklearn de Python, la primera sería una librería dedicada a la minería de datos mientras que la segunda la usaremos para el tratamiento del dataset. En ambos algoritmos trataremos con el dataset iris que nos proporciona la segunda librería.

Para Clarans contamos con dos funciones, la primera que se llamaría 'execute' sería la encargada de ejecutar el algoritmo y la segunda se llamaría 'porcentaje' y nos proporciona el porcentaje de individuos que tiene cada clúster.

El primer paso sería importar el dataset de iris y obtener sus datos.

```
iris = datasets.load_iris()
data = iris.data
```

Ilustración 18 - Importar y obtener dataset Iris

Tras esto debemos convertir estos datos que vienen en forma de array a una list, ya que la librería pyclustering nos requiere este formato.

```
data = data.tolist()
```

Ilustración 19 - Convertir array a list

Ahora procederemos a instanciar el algoritmo CLARANS de la siguiente manera.

```
clarans_instance = clarans(data, 3, 6, 4)
```

Ilustración 20 - Instanciación de CLARANS

El primer parámetro que se le pasa al método sería el dataset, el segundo correspondería al número de clústers que queremos, el tercer parámetro será para el número de mínimos locales obtenidos y el cuarto para el número máximo de vecinos examinados. Posteriormente procedemos a iniciar el proceso y asignarle a un objeto los clústers obtenidos y los medoides correspondientes.

```
(ticks, result) = timedcall(clarans_instance.process)
clusters = clarans_instance.get_clusters()
medoids = clarans_instance.get_medoids()
```

Ilustración 21 - Obtener clusters y medoides

Y ya lo siguiente serían los 'prints' correspondientes para visualizar el resultado del algoritmo.

```
print("Index of the points that are in a cluster : ", clusters)
print("The target class of each datapoint : ", iris.target)
print("The index of medoids that algorithm found to be best : ", medoids)
print("Porcentaje: " + str(porcentaje(clusters)))
```

Ilustración 22 - Mostrar resultados

Para la función 'porcentaje' tenemos el siguiente código.

```
def porcentaje(clusters):
    res = []
    i = 0
    total = 0
    for cluster in clusters:
        size = len(cluster)
        total += size
        res.append([i, size])
        i += 1
    for aux in res:
        aux.append(aux[1]*100/total)
    return res
```

Ilustración 23 - Función porcentaje

Este código lo que hace es contar el número de individuos de cada clúster y ver qué porcentaje de individuos totales representa.

Ahora comenzaremos viendo los resultados del algoritmo Clarans, en nuestro caso el índice de los medoides escogidos son el 53,43 y 63.

```
The index of medoids that algorithm found to be best :  [53, 43, 63]
```

Ilustración 24 - Índices de los medoides encontrados

A su vez tras el cálculo de número de individuos en cada clúster vemos que en el primer clúster tenemos el 14% de los individuos, en el segundo tenemos 33,3% y en el tercero tenemos el 52,66% restante.

```
Porcentaje: [[0, 21, 14.0], [1, 50, 33.333333333333336], [2, 79, 52.666666666666664]]
```

Ilustración 25 - Porcentaje de individuos por clusters.

Para K-Medoids contamos también con las mismas funciones, la primera(execute) es la encargada de ejecutar el algoritmo y la segunda(porcentaje) nos proporciona el porcentaje de individuos que tiene cada clúster.

De hecho, la única modificación sería a la hora de instanciar el algoritmo que sería de la siguiente manera.

```
kmedoid_instance = kmedoids(data, [8, 20, 100])
```

Ilustración 26 - Instanciación de K-Medoide

En el caso del algoritmo k-medoids obtenemos los siguientes índices sobre los medoides escogidos.

```
The index of medoids that algorithm found to be best :  [78, 7, 143]
```

Ilustración 27 - Índices de los medoides encontrados en K-Medoide

En el cálculo de porcentajes para cada clúster obtenemos que el primer clúster tiene el 43,3% de individuos agrupados, el segundo tiene el 33,3% y el tercero cuenta con el 23,3%.

```
Porcentaje: [[0, 65, 43.333333333333336], [1, 50, 33.333333333333336], [2, 35, 23.333333333333332]]
```

Ilustración 28 - Porcentaje de Individuos encontrados por clusters en K-Medoide

Como resultado tendríamos que verificar la calidad de los agrupamientos dados, sin embargo, se sabe que K-Medoide tiene un mejor rendimiento en conjunto de datos pequeños, pero la diferenciación que hace que Clarans destaque sobre el es la mejor complejidad que presenta que hace que podamos trabajar con conjuntos de datos superiores.