Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ciencias Físico Matemáticas

Mejorando la técnica de permutaciones usando grupos para búsquedas por similitud

T E S I S

que para obtener el título de:

licenciada en ciencias físico matemáticas

P R E S E N T A :

Brenda Daniela Medina Aguilar

TUTORA

Dra. Karina Mariela Figueroa Mora

Ciudad Universitaria, Morelia, Michoacán.
Noviembre 2021

Resumen

La búsqueda de similitud consiste en recuperar los elementos más similares en una base de datos. Esto es una tarea central en muchas aplicaciones reales, por ejemplo, reconocimiento de patrones, búsqueda multimedia, etc. Esto se vuelve muy costoso (en tiempo y/o recursos) cuando se utiliza en grandes bases de datos. En vista de estos desafíos, una propuesta eficiente es la construcción de un índice basado en permutantes pero a pesar de tener un muy buen rendimiento en dimensiones altas, existen algunos casos cuando la técnica no reporta elementos relevantes en la consulta que de encontrarlos rápidamente se mejoraría la técnica. Una forma para solucionar este problema es usando una técnica de agrupamiento la cual consiste en usar un conjunto de subconjuntos de permutantes (clases de permutantes) con el fin de tener una mejor precisión a la hora de las consultas. Este trabajo consiste en proponer un un nuevo criterio para la técnica de clases de permutantes con el fin de obtener consultas más eficientes.

Conceptos clave: Búsquedas por similitud, espacios métricos, algoritmo basado en permutaciones.

Abstract

The search for similarity consists of retrieving the most similar elements in a database. This is a central task in many real applications, for example, pattern recognition, multimedia search, etc. In view of these challenges, an efficient proposal is the construction of an index based on permutants, but despite having a very good performance in high dimensions, there are some cases when the technique does not report relevant elements in the query that, if found quickly, would improve the technique. One way to solve this problem is using a grouping technique which consists of using a family of sets of permutants (groups of permutants) in order to have better precision when it comes to queries. This work consists of proposing a new criterion for the technique of groups of permutants in order to obtain more efficient queries.

Key concepts: Similarity searches, metric spaces, algorithm based on permutations.

Dedicatorias

Mi dedicación especial en el presente trabajo de tesis está dirigida a mi madre María de los Angeles Aguilar Arreola quien me dio la vida y ha estado conmigo en todo momento. Gracias por todo mamá, por ser mi cómplice todo el tiempo.

También dedico este trabajo a mi tía Leovigilda Aguilar Arreola y a mi abuela Isabel Arreola Aguilar por apoyarme.

Así como a Eliezer Lozano Trejo y mis grandes amistades que estuvieron conmigo en los momentos buenos y malos.

A mis maestros quienes nunca desistieron al enseñarme, a los sinodales quienes estudiaron mi tesis y la aprobaron. Gracias por su valioso tiempo y dedicación.

A todos los que me apoyaron para escribir y concluir esta tesis.

Sinceramente

BRENDA DANIELA MEDINA AGUILAR

Agradecimientos

En primer lugar deseo agradecer de todo corazón a mi asesora de tesis la Doctora Karina Mariela Figueroa Mora, por su gran esfuerzo y dedicación, muchas gracias por compartir conmigo parte de sus conocimientos.

Agradezco de forma muy especial a mis sinodales, quienes me ayudaron con las correcciones en mi trabajo de tesis, gracias por su paciencia y perseverancia.

De igual forma deseo agradecer a mi madre por su gran apoyo incondicional para cumplir todos mis sueños y metas.

Atentamente

BRENDA DANIELA MEDINA AGUILAR

Índice general

Índice de figuras

Índice de tablas

Lista de Algoritmos

Capítulo 1

Introducción

La necesidad de procesar conjuntos de datos para obtener información ha estado presente durante toda la historia de la humanidad. Por lo cual en 1963 nació el concepto de bases de datos, y una década después el modelo relacional para organizarlas. Este modelo promueve organizar la información en forma de tuplas o registros con una "clave", lo cual serviría posteriormente para recuperar ese registro mediante una búsqueda por igualdad de la clave (o una parte de ésta). Este esquema continúa utilizándose en las bases de datos tradicionales, donde es posible estructurar los datos en tuplas y después hacer búsquedas por igualdad.

Pero a lo largo de los años hemos sido testigos de la aparición de nuevas aplicaciones y de la gran velocidad con la que crecen los tipos de datos, los cuales implican nuevos desafíos, que incluyen cómo procesarlos, y cómo recuperarlos. Entre los nuevos tipos de datos podemos mencionar los multimedia como: imágenes, audio, vídeos, huellas digitales, etc. Nótese que los datos multimedia ya no pueden ser estructurados en claves y registros como se haría en una base de datos tradicional (por ejemplo, los almacenados por el registro civil: nombre de la persona y los padres, ciudad, etc.). Esto ha generado un nuevo problema relacionado con la forma en que se almacena y se consulta la información.

En las *bases de datos multimedia* (BDM) las búsquedas por igualdad ya no son posibles o simplemente no tendrían sentido porque difícilmente los objetos serán iguales, por lo tanto, una alternativa son la búsquedas por similaridad o proximidad. Este tipo

de búsquedas consiste en recuperar de una base de datos los objetos más semejantes o relevantes a una de consulta dada.

Cuando tenemos casos donde se trata miles o millones de objetos y/o la comparación entre ellos es muy costosa (en tiempo y/o recursos), una alternativa ante esto son los índices. Los índices tratan de localizar rápidamente los elementos relevantes.

Un claro ejemplo de la utilidad de un índice es un diccionario físico, donde se tienen miles de palabras que sería muy lento de encontrar si estuvieran desordenadas, en lugar de estar ordenadas alfabéticamente. Lo mismo ocurre en las bases de datos en general, aunque construir un índice no sea tan simple como ordenar lexicográficamente.

Cabe mencionar que las bases de datos están presentes en nuestra vida diaria y realizar búsquedas es una tarea natural y un problema fundamental de la ciencia de la computación.

En vista de estos desafíos, la propuesta aquí planteada para realizar consultas de similitud o proximidad que consiste en mapear el problema a un espacio métrico que está constituido por un conjunto de objetos y una función de distancia. Con este mapeo, el algoritmo aquí planteado, se puede aplicar a cualquier campo donde se defina una medida de similitud entre los elementos, como la función de distancia qué determine que tan similares son dos elementos de ese tipo. Por supuesto, esta medida debe ser definida por un experto en el dominio de la aplicación en específico y puede usarse como una caja negra. Con frecuencia, este tipo de funciones de distancia son muy costosas de calcular, por lo que, nuestro objetivo es reducir el número de cálculos de distancias para resolver cada consulta.

En general, el proceso completo se divide en dos partes: crear un índice, que es un proceso fuera de línea, y realizar consultas sobre el índice. El rendimiento de este tipo de índices depende de la dimensión intrínseca (IDim) de los datos [8]; en la práctica, cuando el IDim es demasiado alto, el rendimiento del índice podría ser inútil y revisar toda la base de datos para llevar acabo la consulta sería muy costoso.

En este trabajo, se propone una novedosa mejora a uno de los algoritmos imbatibles en alta dimensión [9] y cuyo trabajo previo relacionado se encuentra en [11],

además de tener un índice pequeño que se puede mantener en la memoria principal.

1.1. Antecedentes

Existen distintas propuestas para tratar a los objetos de una BDM, en su mayoría están basadas en extraer la información importante de los objetos y representarla formalmente usando vectores de alta dimensión. Esto se conoce como extracción de características. Definimos la *extracción de características* δ como una transformación de un objeto (Obj) a un vector de dimensión1 m, esto es, $\delta : Obj \rightarrow \mathbb{R}^m$, que son almacenados en la BDM.

Una vez procesados los objetos de una BDM, se tienen los vectores asociados a cada objeto, la búsqueda por similaridad consiste en encontrar los vectores con mayor semejanza entre éstos y uno de la consulta.

En algunas bases de datos se emplean índices que aprovechan las coordenadas de cada vector (llamados índices para espacios vectoriales), además aprovechan el hecho de que la similaridad se interpreta geométricamente. Los índices presentan un buen rendimiento en dimensiones bajas ($<$16). A medida que las dimensiones aumentan, su rendimiento se degrada rápidamente. Esto se conoce como la *maldición de la dimensión* [7].

Algunos algoritmos para espacios vectoriales son conocidos como *Spatial Access Met-hods (SAM)* [5]. Entre los más populares están *KD-Tree* [2, 1], *R-Tree* [12] y *X-Tree* [3], por mencionar algunos [13, 4]. Este tipo de técnicas hacen uso exhaustivo de la información de las coordenadas para agrupar y clasificar los objetos en el espacio. Una alternativa para este tipo de base de datos es modelarlos como un espacio métrico (definido formalmente en el capitulo 2).

En este trabajo que se presenta una contribución al artículo *Efficient Group of Permutants for Similary Searching* [11], la idea del trabajo es revisar los diferentes tipos de selección de permutantes que conformaran los grupos y distintos tipos de criterios de distancia, que permiten mejorar el rendimiento del trabajo previo.

1.2. Metodología

La metodología en este trabajo será seguir un análisis experimental de un algoritmo de grupos de permutaciones. Las etapas de este trabajo consistirán en estudiar las propuestas, y evaluar si se obtienen mejor desempeño en la idea de los algoritmos basados en grupos de permutantes.

El rendimiento del trabajo se evaluará en dos tipos de bases de datos: sintéticas y reales. Las bases de datos sintéticas se usarán para controlar la dimensión del espacio y poder estudiar los parámetros del algoritmo. Las bases de datos reales se usarán para demostrar la efectividad de la propuesta.

1.3. Hipótesis

Mejorar el algoritmo de grupos de permutaciones para la búsqueda por similaridad a partir de obtener buenos criterios de selección de permutantes que conformarán a los grupos y distintos tipos de criterios de distancia.

1.4. Propuesta

El trabajo presentado en [11] se utilizó como base de esta propuesta. En dicho trabajo el enfoque es la construcción de un índice utilizando técnicas de agrupamiento, de modo que consultas posteriores pueden ser respondidas de manera eficiente. El objetivo es evitar comparar la consulta con toda la base de datos.

En este trabajo, en particular se proponen distintas estrategias que permiten mejorar el rendimiento del trabajo previo. Ampliamos la forma de reducir el tamaño de la permutación en comparación con el necesario para permutantes aislados, sin reducir la precisión de recuperación.

Capítulo 2

Conceptos Básicos

En una base de datos multimedia, la búsqueda de similitud es la única forma significativa de recuperar los objetos más similares a una consulta determinada. En este capítulo se explican algunos conceptos básicos utilizados a los largo de la tesis.

2.1. Espacio Métrico

Un espacio métrico es definido de la siguiente manera. Sea $\mathbb{X}$ un universo de objetos válidos, donde el subconjunto $\mathbb{U} \subseteq \mathbb{X}$ es una base de datos real de tamaño finito $n = |\mathbb{U}|$ y d es una función de distancia con valor real no negativo $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}^+$ definida entre los elementos $\mathbb{X}$. Las funciones de distancia a la vez deben satisfacer los siguientes axiomas:

- Reflexiva: $d(x, x) = 0$
- Estrictamente positiva: $x \neq y \Longrightarrow d(x, y) > 0$
- Simetría: $d(x, y) = d(y, x)$
- Desigualdad del triángulo: $d(x, z) \leq d(x, y) + d(y, z)$.

Nótese que la idea es trasladar el espacio de búsqueda a un espacio métrico, donde la función de distancia es una aproximación de semejanza entre los objetos. En general se construye un índice, donde sea posible establecer un orden entre los datos, y poder

estimar las distancias entre los objetos. Además, aprovechar que la desigualdad del triángulo pueda descartar objetos sin ser comparados directamente contra la consulta, esto permitirá evitar cálculos de distancia. La desigualdad del triángulo es la única propiedad que podría permitir descartar elementos.

2.2. Consultas por proximidad

Existen dos tipos básicos de consultas por similitud, estos dos tipos de consultas son definidas a continuación. Sea $q \in \mathbb{X}$ el elemento de la consulta, se tiene:

- La consulta de *rango (r):* recupera los objetos dentro de una región centrada en un objeto de consulta determinada q con radio r. Formalmente, $(q, r)_d = \{u \in \mathbb{U} \mid d(u, q) \leq r\}$. En la siguiente Figura 2.1 se muestra un ejemplo de una consulta de rango sobre q.

- La consulta de *K-vecinos más cercanos* o *KNN(q)*: recupera los K elementos de $\mathbb{U}$ que son más cercanos a q, es decir, el resultado es un subconjunto $A \subseteq \mathbb{U}$ tal que $\forall u \in A, \forall v \in (\mathbb{U} - A)$, $d(q, u) \leq d(q, v)$, $K = |A|$. En la siguiente Figura 2.2 se muestra un ejemplo de una consulta de *K-vecinos mas cercanos* sobre q.

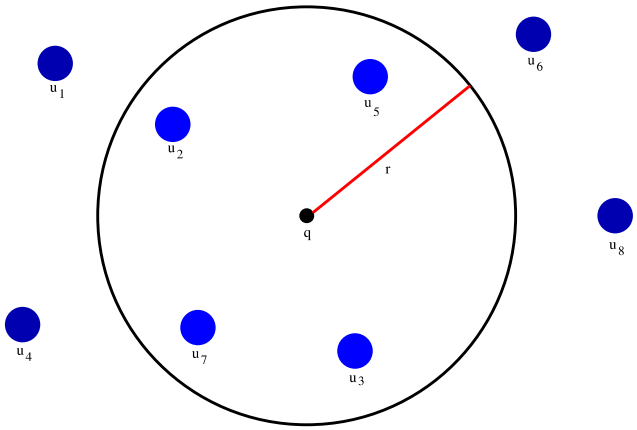


Figura 2.1: Los elementos más cercanos a q mediante la consulta de rango son: u_2, u_3, u_5, u_7.

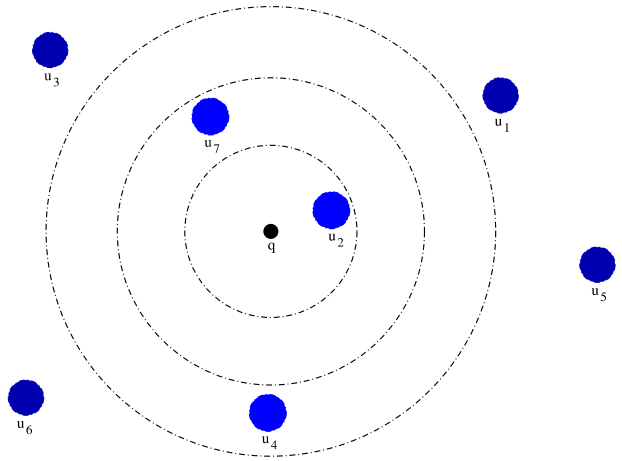


Figura 2.2: Los K elementos más cercanos a q mediante la consulta de *K-vecinos más cercanos* son: con K=1 es u_2, con K=2 son u_2 y u_7, con K=3 son u_2, u_7 y u_4.

2.3. Estado del Arte

Como se menciona en el capítulo anterior los algoritmos que resuelven consultas por similitud implementan índices dentro de la base de datos, para obtener una respuesta más rápida y sin la necesidad de comparar toda la base de datos.

Este tipo de algoritmos resuelven consultas por proximidad, por lo general tienen dos fases que son: el pre-procesamiento y la consulta. En la primera fase del pre-procesamiento se proyecta la base de datos en un nuevo espacio que permite construir un índice, que posteriormente se almacena. En la parte de la construcción del índice se seleccionan algunos elementos de la base de datos, los cuales funcionarán como elementos clave, éstos se comparan contra el resto de objetos de la base de datos, a esto se le conoce como *distancias internas*, guardando como índice el orden en el que se encuentran los elementos clave con respecto a los demás objetos de la base de datos.

En la segunda fase de la consulta se recorre el índice para obtener una *lista de candidatos* los cuales serán comparados directamente contra la consulta, a esto se le conoce como *distancias externas*. Para este tipo de algoritmos basados en espacios métricos, básicamente existen tres principales familias de algoritmos de búsqueda por similitud:

- Algoritmos basados en particiones compactas.
- Algoritmos basados en pivotes.
- Algoritmos basados en permutantes.

2.4. Algoritmos basados en particiones compactas

Un índice basado en particiones (PaBI), divide el espacio en zonas. La idea consiste en seleccionar un conjunto de objetos $p_1, p_2, p_3, ..., p_k \subseteq \mathbb{U}$ y se divide el espacio, es decir, el resto de los elementos se distribuyen entre las k zonas pertenecientes a cada p_i. El índice se compone de los p_i llamados *centros* de su zona, los elementos que pertenecen a cada zona, y en algunos casos, información adicional sobre las distancias [7].

En estos algoritmos suelen clasificarse de acuerdo a su procedimiento de búsqueda:

- Los que utilizan un radio de cobertura r_c, que es la distancia máxima del centro de la zona a los elementos en ella.
- Los que utilizan hiperplano, que es la frontera entre dos zonas cuando cada elemento se asigna a su centro más cercano.
- Los que usan ambos criterios.

Usando el criterio de radio de cobertura durante la búsqueda, es posible acotar la distancia entre una consulta y los elementos en la base de datos de la siguiente manera.

Lema 1 *Dados tres objetos* u, $p \in \mathbb{U}$, $q \in \mathbb{X}$, donde $\mathbb{X} \in \mathbb{U}$, *si* p *es el centro de la zona a la que pertenece* u, *con radio de cobertura* r_c, *sabemos que* $d(q,u)$ *está acotada por* $máx\{d(q,p) - r_c, 0\} \leq d(q,u) \leq d(q,p) + r_c$.

Demostración Ambos límites se obtienen de la desigualdad del triángulo y la cota superior:

$$d(p,q) \leq d(p,u) + d(u,q),$$
$$d(p,u) \leq r_c.$$

Usando estas dos ecuaciones tenemos

$$d(p,q) - d(q,u) \leq d(p,u) \leq r_c,$$

esto implica que:

$$d(p,q) - r_c \leq d(q,u).$$

Por otro lado, sabemos que $d(q,u) \geq 0$, por lo tanto el límite inferior es :

$$máx\{d(q,p) - r_c, 0\} \leq d(u,q).$$

El límite superior lo podemos obtener de la desigualdad del triángulo:

$$d(q,u) \leq d(q,p) + d(p,u) \leq d(q,p) + r_c.$$

Durante una consulta por rango $(q,r)_d$, esta familia de algoritmos descarta aquellas zonas de centro p tales que $d(p,q) - r_c > r$, pues usando el Lema 1 sabemos que cualquier elemento u en esa zona cumple con $d(q,u) > r$. En estas zonas no descartadas se repite el proceso hasta que se llega a zonas sin divisiones. En éstas los elementos deben revisarse secuencialmente. En una búsqueda usando el criterio de los hiperplanos, es posible acotar $d(q,u)$.

Lema 2 *Sea $u \in \mathbb{U}$ un objeto más cercano a p_1 que a p_2 o que equidista, es decir, $d(p_1,u) \leq d(p_2,u)$. Dados $d(q,p_1)$ y $d(q,p_2)$, podemos establecer la cota $máx\{\frac{d(q,p_1)-d(q,p_2)}{2}, 0\} \leq d(q,u)$.*

Demostración De la desigualdad del triángulo sabemos que:

$$d(q, p_1) \leq d(q, u) + d(p_1, u),$$

lo que implica que:

$$d(q, p_1) - d(q, u) \leq d(p_1, u).$$

De la misma forma tenemos que:

$$d(p_2, u) \leq d(q, p_2) + d(q, u),$$

y por hipótesis tenemos

$$d(p_1, u) \leq d(p_2, u).$$

Combinando las ecuaciones anteriores obtenemos

$$d(q, p_1) - d(q, u) \leq d(p_1, u) \leq d(p_2, u) \leq d(q, p_2) + d(q, u),$$

$$d(q, p_1) - d(q, u) \leq d(q, p_2) + d(q, u),$$

$$d(q, p_1) - d(q, p_2) \leq 2d(q, u),$$

Ya que $d(q, u) \geq 0$, la cota inferior es

$$\text{máx}\{\frac{d(q, p_1) - d(q, p_2)}{2}, 0\} \leq d(q, u).$$

Utilizando sólo el criterio de hiperplanos no es posible determinar una cota superior, dado que los objetos pueden estar muy cerca o muy lejos de p_1 y p_2. En particular, una zona p_j debe ser revisada si $\frac{d(q,p_j)-d(q,p_i)}{2} \leq r$, y q pertenece a la zona de p_i. Para poder decidir visitar o descartar las otras zonas de la base de datos, se repite el mismo cálculo.

2.5. Algoritmos basados en pivotes

Un índice basado en pivotes (PiBI) elige un conjunto de elementos $\mathbb{P} = \{p_1, p_2p_3, ..., p_k\} \subseteq \mathbb{U}$, de tamaño $k = |\mathbb{P}|$, a estos elementos los llamaremos *pivotes*. Para cada uno de los objetos restantes de la base de datos se calcula su distancia hacia los pivotes en $\mathbb{P}$. Este conjunto de distancias $\{d(p_1, u), d(p_2, u), d(p_3, u), ..., d(p_k, u)\}$, para cada $u \in \mathbb{U}$ conformará finalmente el índice.

Dada una consulta q, se calcula la distancia contra cada pivote $d(p_i, q)$. Con esto se puede acotar la distancia entre q y cualquier $u \in \mathbb{U}$ tal como se demuestra en el siguiente lema.

Lema 3 Dados tres objetos $q \in \mathbb{X}$, $u \in \mathbb{U}$, $p \in \mathbb{P}$, sabemos que $|d(q, p) - d(p, u)| \leq d(q, u) \leq d(q, p) + d(p, u)$.

Demostración El límite superior se obtiene directamente de la desigualdad del triángulo,

$$d(q, u) \leq d(q, p) + d(p, u).$$

En el caso del límite inferior, de acuerdo a la desigualdad del triángulo, se tiene que:

$$d(p, u) \leq d(p, q) + d(q, u),$$

$$d(p, q) \leq d(p, u) + d(u, q).$$

Estas desigualdades implican:

$$d(p, u) - d(p, q) \leq d(q, u),$$

$$d(p, q) - d(p, u) \leq d(u, q),$$

combinando estas desigualdades y usando la simetría, obtenemos que:

$$|d(p,u) - d(p,q)| \leq d(u,q).$$

Este tipo de algoritmos funcionan haciendo una consulta por rango y usando el Lema 3 podemos descartar todos los elementos de la base $u \in \mathbb{U}$ tales que $\exists p \in \mathbb{P}, |d(q,p) - d(p,u)| > r$, si esto sucede entonces es claro que $d(q,u) > r$. Los elementos no descartados se comparan directamente contra la consulta.

2.6. Algoritmos basados en permutantes

Los algoritmos basados en permutaciones (PeBI) seleccionan un conjunto de elementos de la base de datos $\mathbb{P} = \{p_1, p_2, p_3, ..., p_k\} \subseteq \mathbb{U}$, donde $|\mathbb{P}| = k$, estos elementos los llamaremos *permutantes*. Para cada elemento restante de la base de datos $u \in \mathbb{U}$ se calcula su distancia a los permutantes, es decir, $D_u = \{d(u,p_1), d(u,p_2), d(u,p_3), ..., d(u,p_k)\}$. Una vez calculadas las distancias de cada elemento u hacia los permutantes, se ordenan, en este caso es de manera ascendente, una vez ordenados guardamos el orden en que va cada permutante según su distancia con respecto a cada elemento u, a este orden de cada elemento lo definiremos como una permutación Π_u, teniendo así una lista de permutaciones la cual conformara nuestro indice, de manera que para todo $1 \leq i < k$ se mantiene $d(p_{\Pi_{u(i)}}, u) < d(p_{\Pi_{u(i+1)}}, u)$ o $d(p_{\Pi_{u(i)}}, u) = d(p_{\Pi_{u(i+1)}}, u)$ y $\Pi_u(i) < \Pi_u(i+1)$, donde $p_{\Pi_{u(i)}}$ es el permutante p en la posición *i-ésima*, y $\Pi_{u(i)}$ es un permutante en la posición i de esa permutación.

La idea está basada en que al tener dos objetos idénticos se debe tener la misma permutación, mientras que dos objetos similares, deberían tener permutaciones similares. De manera que el objetivo de esta técnica es identificar en el indice a los elementos más cercanos entre sí usando la semejanza entre las permutaciones.

Los elementos de $\mathbb{P}$ se seleccionaron de manera aleatoria, en [10] se explican otras formas de seleccionar los permutantes que más adelante se mencionarán. Las permutaciones inversas están denotadas como Π_u^{-1}, las cuales nos indican en qué posición se encuentra un permutante con respecto de u. En la Figura 2.3 tenemos un ejemplo de cómo se vería el elemento u con respecto a los permutantes, donde

Posición(i)	1	2	3	4	5	6
Permutante	3	4	2	6	5	1

Tabla 2.1: Permutación de u (Π_u)

Permutante	1	2	3	4	5	6
Posición(i)	6	3	1	2	5	4

Tabla 2.2: Permutación inversa de u (Π_u^{-1})

$d(u, p_1) = 34$, $d(u, p_2) = 19$, $d(u, p_3) = 9$, $d(u, p_4) = 12$, $d(u, p_5) = 31$ y $d(u, p_6) = 21$. Teniendo así que $D_u = \{34, 19, 9, 12, 31, 21\}$, por lo que la permutación de u sería $(3, 4, 2, 6, 5, 1)$. Para la permutación inversa note que el permutante p_1 se encuentra en la posición 6, el permutante p_2 se encuentra en la posición 3, el permutante p_3 se encuentra en la posición 1, y así sucesivamente, de aquí que la permutación inversa sea $\Pi_u^{-1} = (6, 3, 1, 2, 5, 4)$. A continuación se muestran en la segunda fila de cada tabla 2.1 y 2.2 las permutaciones de Π_u y de Π_u^{-1} respectivamente.

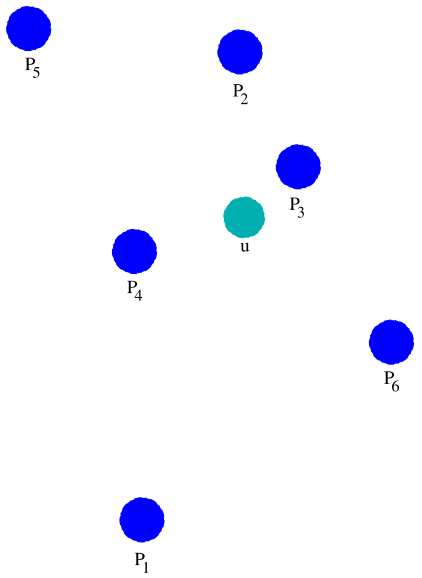


Figura 2.3: Posición de u con respecto a los permutantes, donde $d(u, p_1) = 34$, $d(u, p_2) = 19$, $d(u, p_3) = 9$, $d(u, p_4) = 12$, $d(u, p_5) = 31$ y $d(u, p_6) = 21$..

Existen algunas medidas de distancia entre permutaciones para conocer qué tan parecidas son entre ellas, los autores en [9] presentaron varias de estas distancias. Pero la más simple y con un desempeño competitivo fue la similitud de Spearman

Footrule (S_f), que está definida como:

$$S_f(\Pi_u, \Pi_q) = \sum_{1 \leq i \leq k} |\Pi_u^{-1}(i) - \Pi_q^{-1}(i)|$$

Donde Π_u^{-1} y Π_q^{-1} son las permutaciones inversas del objeto u y de una consulta q respectivamente.

Tomando como ejemplo a $\Pi_u = (3, 4, 2, 6, 5, 1)$ y con su permutación inversa como $\Pi_u^{-1} = (6, 3, 1, 2, 5, 4)$, para una consulta q cuya permutación sea $\Pi_q = (4, 3, 2, 5, 6, 1)$ y con su permutación inversa sea $\Pi_q^{-1} = (6, 3, 2, 1, 4, 5)$. Cada elemento en particular de la permutación Π_u se encuentra a una posición con respecto a la posición en Π_q. Y utilizando Spearman Footrule tenemos: 6-6, 3-3, 1-2, 2-1, 5-4, 4-5, estos resultados son sumados los valores absolutos obteniendo $S_f(\Pi_u, \Pi_q) = 4$. El resultado de $S_f(\Pi_u, \Pi_q)$ indica qué tan similares son dos permutaciones, entre más pequeño es el resultado entonces son más similares.

Los PeBI son algoritmos que tienen un buen rendimiento en grandes bases de datos, pero existen algunos falsos negativos que de encontrarlos rápidamente se mejoraría la técnica. Uno de los problemas de los PeBI, es que tienen una frontera entre los elementos cercanos a un permutante y a otro. La línea de negra que denotaremos como "frontera", delimita la mitad entre dos permutantes (p_1 y p_2), observe la Figura 2.4. A pesar de que q y u son cercanos, tienen una permutación invertida entre ellos, idealmente las permutaciones de u y q deberían ser iguales por su cercanía, esto convierte al elemento u en un falso negativo, pues podría no ser reportado relevante respecto a los primeros elementos comparados.

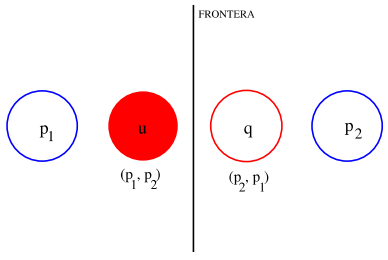


Figura 2.4: Problema de cercanía entre permutantes.

La propuesta de esta tesis consiste en mejorar la heurística propuesta en [11], para

tratar de reducir la problemática explicada en el párrafo anterior. En el capítulo 2.7 se explica en detalle cómo esta técnica basada en agrupamiento (clases) de permutantes puede contribuir a mejorar el desempeño de este algoritmo.

2.7. Clases de Permutantes

La metodología de los algoritmos de búsqueda en permutaciones (GPeBI) se explicará en este capítulo y aparece la propuesta previa de este trabajo en "Efficient group of permutants for proximity searching" [11].

La idea de implementar Clases de permutaciones (Π_G) (dado que el concepto de *clases* es el término correcto, el resto del trabajo se hablará de *clases*), está inspirada en la problemática de los elementos que quedan cercanos a dos permutantes descrita en la sección 2.6. Al usar clases de permutantes se realiza un intento para obtener una mayor precisión a la hora de las consultas, difuminando la línea que representa la frontera. Esta técnica propone que en lugar de tener un conjunto de permutantes, se tenga un conjunto de conjuntos de permutantes que denotaremos como clases de permutantes, obteniendo que la frontera no esté determinada por un único elemento. A continuación se definirá formalmente esta técnica.

2.7.1. Definición

El algoritmo de GPeBI consiste en utilizar clases de permutantes, en lugar de permutantes aislados, de esta manera, usamos la misma cantidad de espacio que los algoritmos PeBI, pero tenemos mayor precisión.

Formalmente el conjunto de clases de permutantes está definido de la siguiente manera. Sea $\mathbb{G} = \{G_1, G_2, ..., G_k\}$ un conjunto de permutantes agrupados en conjuntos más pequeños, donde $k = |\mathbb{G}|$. Y cada clase está compuesto por m permutantes, por lo que tendríamos $G_1 = \{p_1, ..., p_m\}$, $G_2 = \{p_{m+1}, ..., p_{2*m}\}$, . . . , $G_k = \{p_{(k-1*m}, ..., p_{k\times m}\}$, donde $\{p_1, p_2, p_3, ..., p_{kxm}\} \subseteq \mathbb{U}$. En la sección 2.7.3 tenemos distintas maneras para conformar los clases.

Para la etapa de preprocesamiento, se calcula la distancia para todo $u \in \mathbb{U}$ y

$u \notin \cup\mathbb{G}$ hacia los permutantes de cada clase. Pero nótese que ahora para obtener la permutación de u se necesita un criterio de distancia que dependa de los permutantes de cada clase, en la siguiente sección definiremos algunos de estos criterios. Una vez definido el criterio de distancia podemos obtener las distancias de u contra los clases $D'_i(u, G_i)$, $1 \leq i \leq k$, y ordenamos D'_i por proximidad a u, teniando así la permutación de u. Básicamente, la propuesta es que ahora las permutaciones se hacen respecto a los clases, a diferencia de los $PeBI$ comunes donde se hacen respecto a un sólo permutante.

2.7.2. Criterios de distancia a un clase

Por cada clase tenemos m elementos, por lo que se pueden definir varios criterios para calcular las distancias de los clases. Consideremos estas cuatro opciones:

1. *Mínima (min):* es la distancia mínima a todos los objetos del clase. Formalmente, $D_{min}(u, G_i) = min_{p \in G_i} d(p, u)$.

2. *Maxima (max):* es la mayor distancia a todos los objetos del clase. Formalmente, $D_{max}(u, G_i) = max_{p \in G_i} d(p, u)$.

3. *Promedio (prom):* es el promedio a todas las distancias de los objetos del clase. Formalmente, $D_{prom}(u, G_i) = \sum_{p \in G_i} \frac{d(p,u)}{m}$.

4. *Cabecera de Clase (cabG):* distancia al elemento considerado cabecera de clase.

2.7.3. Selección de clases de permutantes

En la idea original se escogen de manera aleatoria los permutantes que conformaron cada clase. En este caso tenemos $k * m$ permutantes seleccionados ya que cada clase de permutantes está conformado por m elementos y tenemos k clases. Ahora nos es posible definir distintas estrategias de selección de permutantes que conformarán los clases propuestas por los autores en [11]. Los permutantes se eligen utilizando los mismos criterios distancia para todos. Consideremos estas cinco opciones:

1. *Aleatorios (rand)*: Este criterio consiste en elegir elemento en el clase al azar.

2. *Cercanos dentro del clase (cerc):* selecciona el primer elemento de cada clase al azar y agrega al clase sus $m-1$ permutantes más cercanos.

3. *Lejanos dentro del clase (lej):* selecciona el primer elemento de cada clase al azar y agrega al clase sus $m-1$ permutantes más lejanos.

4. *Pares Cercanos (paresC):* Selecciona por cada clase, un par de vecinos más cercanos mutuos y los $m-2$ permutantes que minimizan la suma de distancias a las anteriores en el clase.

5. *Lejanos dentro del clase (SSS):* Selecciona k jefes de clase siguiendo el método en [6] para elegir objetos esparcidos en el espacio. Donde M es la distancia máxima entre todos los objetos pares. Los jefes de cada clase tienen al menos una distancia $0{,}4 \times M$ entre ellos. Los clases se completan con el elemento más cercano a cada jefe de clase.

Capítulo 3

Propuesta

Anteriormente en los capítulos pasados hemos explicado el funcionamiento de tener clases de permutantes. Es una forma novedosa de reducir el tamaño de la permutación en comparación con el necesario para PeBI estándar, sin reducir la precisión de recuperación. La propuesta en este trabajo será poner a prueba los distintos criterios de selección de permutantes y criterios de distancia con los mismos parámetros, con el fin de tener una vista más clara del costo de cada criterio de selección y de distancia obteniEndo como resultados que tipo de criterio funcionan de manera más óptima según los parámetros que tengamos.

Además en este capítulo se propone un nuevo criterio de distancia. El cual será puesto a prueba con el resto de los distintos criterios de selección de permutantes y criterios de distancia con los mismos parámetros.

La propuesta del nuevo criterio de distancia: Es D_{av+min} que consiste en la suma del promedio de distancias de todos los objetos del clase D_{prom} más la distancia mínima de todos los objetos del clase D_{min}. Formalmente $D_{av+min}(u, G_i) = D_{av}(u, G_i) + D_{min}(u, G_i)$.

3.1. Construcción de índice

La implementación consiste en crear el índice para resolver las consultas por similaridad, usando el pseudocódigo del algoritmo 1. Este algoritmo recibe la base de

datos $\mathbb{U}$, el número de permutantes por clase m, el número de clases k, y la función regresará el índice I. El primer ciclo asigna los permutantes que conFormarán los clases, se puede observar que usa $k \times m$ elementos para conFormarlos. En la línea 6 se utiliza *CalcularD* para calcular la distancia de un elemento u hacia los clases y cambia según el criterio elegido. *Dis* es una estructura que contiene tuplas con las distancias hacia cada clase $D_i(u, G_i)$ e id es el identificador de cada clase. Estas distancias se ordenan de menor a mayor con el propósito de Formar la permutación de cada elemento (el método de ordenamiento puede ser *qsort* dado su buen desempeño) y en caso de que las distancias sean iguales, se toma en cuenta el id del clase de permutantes para romper el empate. Para todos los ordenamientos que se llevan a cabo a lo largo del algoritmo se implementó el Quick Sort (qsort) de la librería estándar del lenguaje C cuya complejidad promedio es $O(nlogn)$. Estas permutaciones se almacenan en un índice I además de otros parámetros importantes como $\mathbb{G}$, m y k.

Algorithm 1 Construir el índice

```
1:  U, m, k, I
2:  Sea Dis un arreglo con tuplas id y d
3:  Para i = 1 ⟶ k * m hacer
4:      G[i] ← tipo-de-selección(i)
5:  Para u ∈ U y u ∉ G hacer
6:      Para i = 1 → k hacer
7:          Dis[i].d ← CalcularDistancia − al − clase(G_i, u, m)
8:          Dis[i].id ← i
9:      qsort(Dis)
10:     Para i = 1 ⟶ k hacer
11:         Π_u[i] ← Dis[i].id
    return I
```

3.2. Consultas

Una vez construido el índice se puede proceder a hacer las consultas ya sea por radio o por *KNN*. En esta fase se utilizará el *Algoritmo* 2 para la búsqueda por radio y el *Algoritmo* 3 para la búsqueda de *KNN*, para ambos algoritmos primero se calculan las distancias de q hacia los permutantes, así conseguiremos la permutación inversa

de q, es decir, Π_q^{-1} pues será útil para calcular *Spearman Footrule*. El *Algoritmo 2* regresa el total de elementos dentro del radio r. Primero se calcula la cercanía de Π_q hacia todos los Π_u, esta cercanía se calcula usando Spearman Footrule de acuerdo a la ecuación2.6.

Una vez que se calcularon las distancias de *Spearman Footrule* Π_q al conjunto de elementos Formado por $\mathbb{U} - \mathbb{G}$, éstos se ordenan de menor a mayor, debido a que las que tienen menor distancia son más similares. Con este orden establecido se evalúa la distancia real de los elementos en la base de datos, con respecto a la consulta q, esto es *d(u,q)*.

Observe que sólo se comparará un porcentaje de elementos α en la línea 9, si este candidato propuesto se encuentra dentro del radio r, entonces se reporta el objeto. *Aprox.id* contiene la posición cada elemento de $\mathbb{U}$, así sabemos a qué objeto nos referimos. En la sección de pruebas se explicará por qué sólo se utiliza un porcentaje de elementos. El *Algoritmo 3* regresa la distancia a la que se encuentra el *K-ésimo* vecino más cercano. Este algoritmo utiliza un arreglo ordenado de tamaño K. Al igual que en el *Algoritmo 2*, se calcula la cercanía entre Π_q. y todos los Π_u esto es $S_F(\Pi_u, \Pi_q)$. También se ordenan de menor a mayor y así se establecen los elementos más relevantes.

Después se realizan las comparaciones externas para encontrar a los *KNN*. Si un elemento tiene menor distancia que cualquiera de los que están en el arreglo, entonces se reemplaza el elemento más alejado (mayor distancia hacia q) por el nuevo elemento. La función radio regresa la distancia hacia el *K-ésimo* vecino más cercano.

Los criterios utilizados para conocer la cercanía entre un elemento u y una clase pueden ser *min*, *max*, *prom*, *paresC*, *av+min* recuerde que estos permiten construir una permutación, de la misma Forma que se hizo para cada elemento en la base de datos (en el *Algoritmo 1*). Estos criterios serán sometidos a pruebas para evaluar su efectividad.

En la tabla siguiente se hace un resumen de todas las abrevaciones que se usarán en el siguiente capítulo de experimentos.

Algorithm 2 Búsqueda por radio

```
1: I, q, r, cont
2: Sea Aprox un arreglo con tuplas (entero, entero)
3: cont ← 0
4: Π_q^{-1} ← CalculaQInv(I, q)
5: Para u ∈ 𝕌 y u ∉ 𝔾 hacer
6:     Aprox.d ← SpearmanFootrule (Π_q^{-1}, Π_u)
7:     Aprox.id ← u
8: Aprox ← qsort(Aprox)
9: Para i to α hacer
10:    d ← d(q, Aprox[i].id)
11:    Si d ≤ r entonces
12:        reportar Aprox[i].id
13:        cont ← cont + 1
   return cont
```

Algorithm 3 Búsqueda de KNN

```
1: I, q, K
2: r
3: r ← −1
4: KNN ← 0
5: Sea C un arreglo de tamaño k
6: Sea Aprox un arreglo con tuplas (entero, entero)
7: cont ← 0
8: Π_q^{-1} ← CalculaQInv(I, q)
9: Para u ∈ 𝕌 y u ∉ 𝔾 hacer
10:    Aprox.d ← SpearmanFootrule (Π_q^{-1}, Π_u)
11:    Aprox.id ← u
12: Aprox ← qsort(Aprox)
13: Para i to α hacer
14:    d ← d(q, Aprox[i].id)
15:    Si d ≤ r OR r = −1 entonces
16:        KNN ← push(C, Aprox[i].id, d)
17:        Si i ≥ K entonces
18:            r ← radio(C)
   return r
```

Abreviatura	Descripción
k	Número de clases de permutantes
m	Número de elementos por clase
S_F	Métrica de Spearman Footrule
r	Consulta por rango
KNN	Consulta por K-vecinos más cercanos
BD	Base de Datos
BDS	Base de Datos Sintética
BDR	Base de Datos Reales
Selección de elementos por clase	
rand	Selección de elementos de manera aleatoria
cerc	Selección de elementos más cercanos a un elemento (cabecera)
lej	Selección de elementos más lejanos a un elemento (cabecera)
paresC	Selección de elementos más cercanos a un par
SSS	Técnica de selección espacial dispersa
Distancia a una clase	
prom	Distancia promedio a un elemento para una clase
max	Distancia máxima del clase
min	Distancia mínima del clase
cabG	Distancia al elemento cabecera de clase
av+min	Distancia media a un elemento para una clase (prom+min)

Tabla 3.1: Palabras abreviadas utilizadas en este trabajo.

Capítulo 4

Experimentación

En este capítulo se mostrará el rendimiento del algoritmo de clases de permutaciones con sus distintos tipos de selección de permutantes y de criterios de distancia. El rendimiento del algoritmo de clases de permutantes se midió en dos tipos de bases de datos como se mencionaba anteriormente que son:

- Las bases de datos sintéticas (BDS). Éstas están conformadas por, vectores aleatorios distribuidos de manera uniforme en el cubo unitario.
- Las bases de datos reales (BDR). En estas bases de datos se emplearon 3 tipos: NASA, Colors y diccionarios de palabras (uno en Español y otro en inglés). En la sección 4.2 se explican detalladamente las características de cada una.
    - NASA
    - Colors
    - Diccionarios

4.1. Base de Datos Sintéticas

Las BDS son vectores aleatorios distribuidos de manera uniforme en el cubo unitario, de esta manera es posible controlar la dimensión intrínseca de los datos y la cantidad de datos; este tipo de BDS puede usarse como un espacio métrico abstracto.

Una diferencia que tienen las pruebas de una BDS con respecto a las pruebas en una BDR se debe a que esta última no podemos controlar la dimensión intrínseca de los objetos que componen las bases de datos a diferencia de las BDS.

En el momento de las pruebas se trabajó con 3 dimensiones distintas que fueron *dim = 16,32,64*. En cada una de las dimensiones se trabajó con 4 conformaciones distintas de clases que son $k = \{8, 16, 32, 64\}$, con 3 cantidades diferentes de elementos por clase que son $m = \{2,3,4\}$ en combinación de 5 tipos de selección de permutantes (selectP) y 6 tipos de distancias (critDist).

Se muestran en el cuadro 4.1 todos distintos parámetros a considerarse en este experimento. Lo que al menos nos garantiza que se hicieron 900 experimentos sólo para las BDS. De todas estas combinaciones se obtuvieron varias gráficas; sin embargo, en este capítulo se mostrarán solo algunas. El resto de las mejores gráficas las puede consultar en el apéndice A.

dim	numG	m	selectP	critDist
16	8	2	rand	prom
32	16	3	cerc	max
64	32	4	lej	min
–	64	–	SSS	cabG
–	–	–	paresC	av+min

Tabla 4.1: Tipos de pruebas para las BDS.

Para cada una de las gráficas mostradas a continuación se muestran las distancias totales, es decir, distancias internas más las distancias externas.

Para el caso de las distancias internas se considera el valor de m por el número de clases escogidos, más el número de distancias externas.

4.1.1. Dimensiones 32 variando m

Dado que las dimensiones trabajadas fueron de 16, 32 y 64 en este capítulo solo mostraremos la dimensión 32 pues es el valor medio.

La organización de las gráficas se realizó de la siguiente manera: primero se mostrará el desempeño considerando m=2 (Figura 4.1), para $m = 3$ véase la Figura 4.2.

Para $m = 4$ se obtuvieron resultados semejantes a los anteriores; éstas gráficas pueden consultarse en el apéndice A.

En cada una de las figuras se organizó la información de la siguiente manera (de izquierda a derecha y de arriba a abajo):

- la 1a gráfica tiene el criterio de distancia de av+min y promedio, para el criterio de selección de aleatorio, SSS y pares cercanos.
- la 2a gráfica mantienen los criterios de distancia de av+min y promedio, y cambia los criterios de selección de cercanos y lejanos.
- la 3a gráfica tiene los criterios de distancia: cabecera de clase, máxima y mínima para los criterios de selección de aleatorio, SSS y pares cercanos.
- la 4a gráfica usa los criterios de distancia de cabecera clase, máxima y mínima, con los criterios de selección de cercanos y lejanos.

Discusión de la Figura 4.1

En estas gráficas se usaron solo 2 permutantes por clase.

De la 1a gráfica se aprecia que la técnica de permutaciones originales es mejor a medida que el número de clases aumenta, sin embargo, cuando se usan pocos clases el criterio de *paresC* y la distancia de *prom* tiene un mejor desempeño (con 16 clases).

Para la 2a gráfica tenemos que nuevamente el criterio de *prom* es una buena opción. También es de remarcarse que la selección de *lejanos* tiene un buen desempeño.

Para la gráfica 3, nuevamente el criterio de *paresC* muestra su buen desempeño al combinarse con otra distancia como *mínimo* y *cabecera de clase*.

En la gráfica 4, nuevamente la opción de *lejanos* y *cabecera de clase* tienen un buen desempeño. El resto de las opciones no son tan competitivas.

Cómo puede observarse en estas gráficas, el desempeño no es tan distinto entre algunos criterios; recordemos que se está trabajando con bases de datos sintéticas cuya distribución de los datos es uniforme.

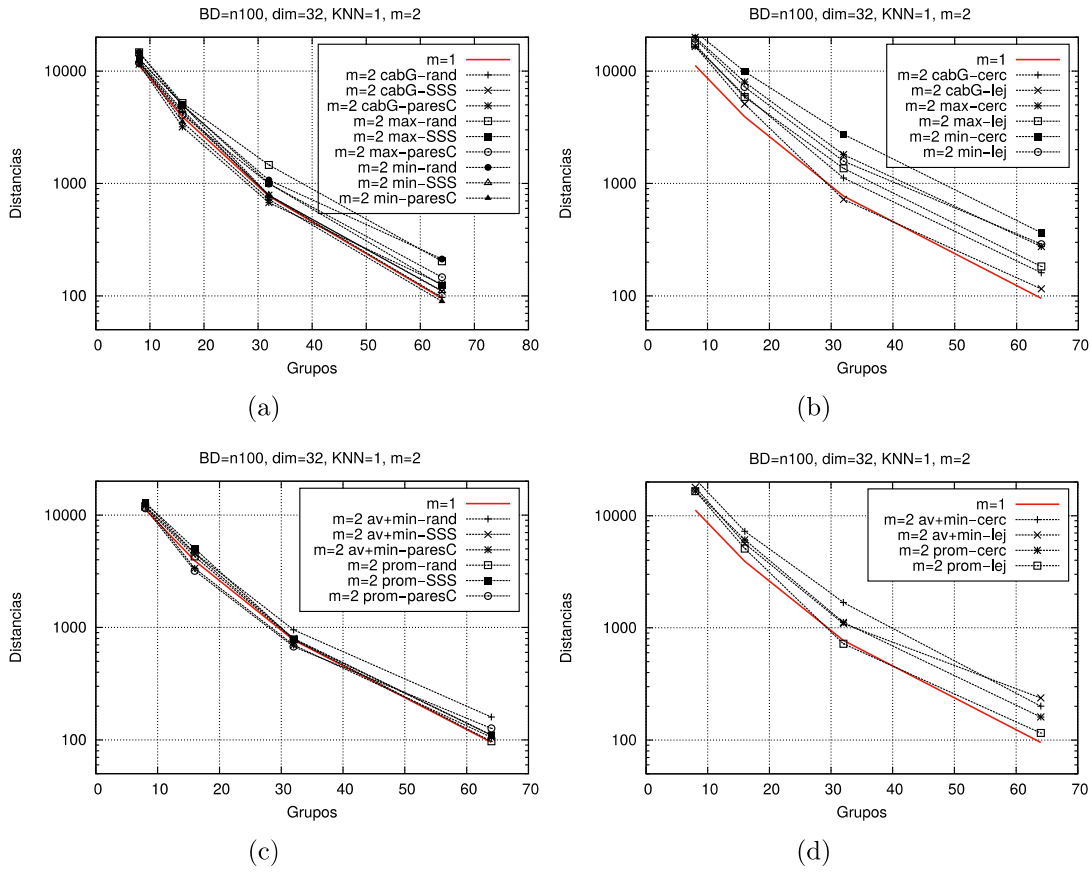


Figura 4.1: Dimensión 32 con m =2, k ={8,16,32,64}

Discusión de la Figura 4.2

Para las gráficas con m=3 (4.2), tenemos el mismo desempeño que usando $m = 2$. Aunque empeora un poco el número de distancias totales puesto que el valor de m aumenta, por lo que, el número de distancias totales aumenta.

Como conclusión, los parámetros con mejor desempeño son: *av+min* y *SSS*; *promedio* y *SSS*; *mínimo* y *lejanos*; y *mínimo* y *pares cercanos*.

4.1.2. Dimensión 32 variando KNN

En las siguientes gráficas con m={2,3} (4.3) solo se considerarán los parámetros que tuvieron el mejor desempeño. En este caso se variará el número de vecinos mas cercanos a recuperarse: KNN=1,2,4,8. Los parámetros utilizados en las siguientes gráficas son: *av+min* y *SSS*; *promedio* y *SSS*; *mínimo* y *lejanos*; y *mínimo* y *pares*

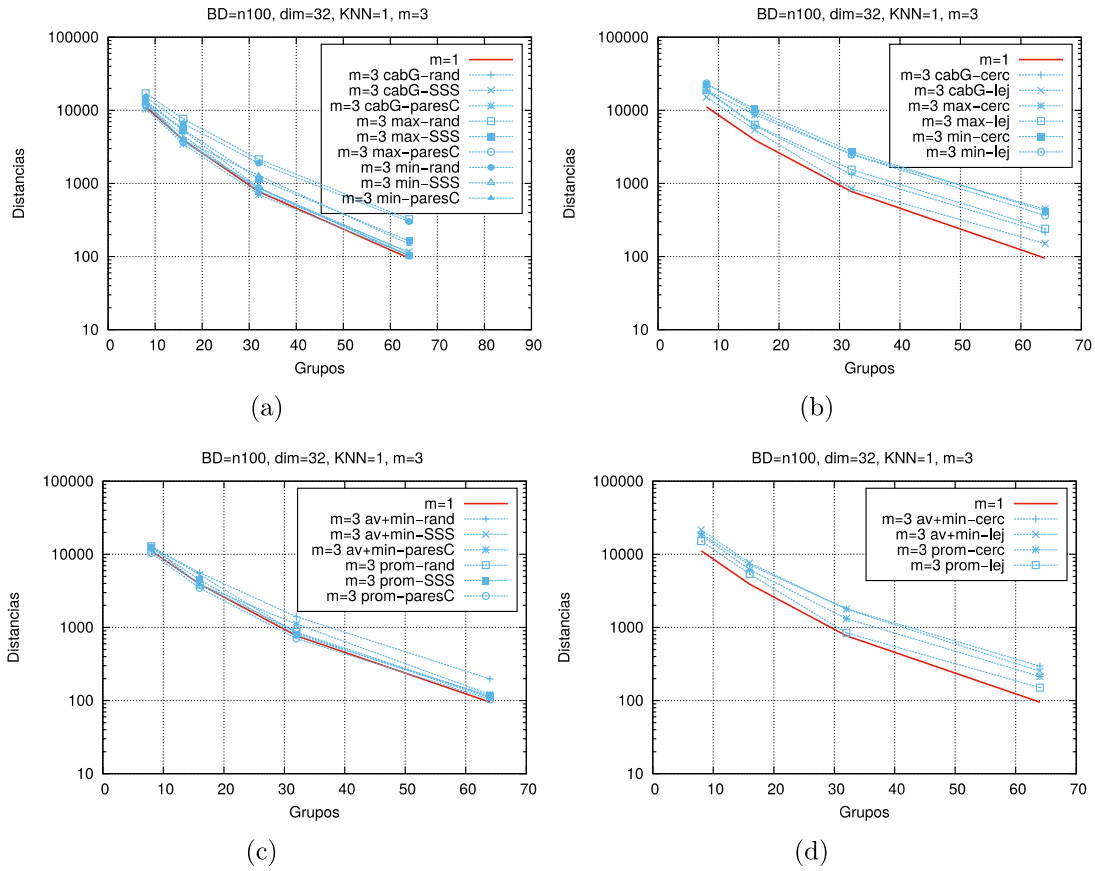


Figura 4.2: Dimensión 32 con m =3, k ={8,16,32,64}

cercanos.

4.2. Base de Datos Reales

Para el caso de las base de datos reales se trabajó en 4 base de datos distintas, 2 fueron de diccionarios y 2 fueron de imágenes. En cada una de las base datos reales se trabajó con 4 conformaciones distintas de clases y con 3 cantidades diferentes de elementos por clase, comparados contra la técnica de permutantes original, en combinación 5 tipos de selección de permutantes contra 6 distancias diferentes.

Las bases de datos reales utilizadas para realizar las pruebas correspondientes y viendo su efectividad en datos reales (BDR), estas BDR fueron *English*, *Spanish*, *Colors* y *Nasa*.

La base de datos de objetos reales English, es un conjunto de 69069 histogramas

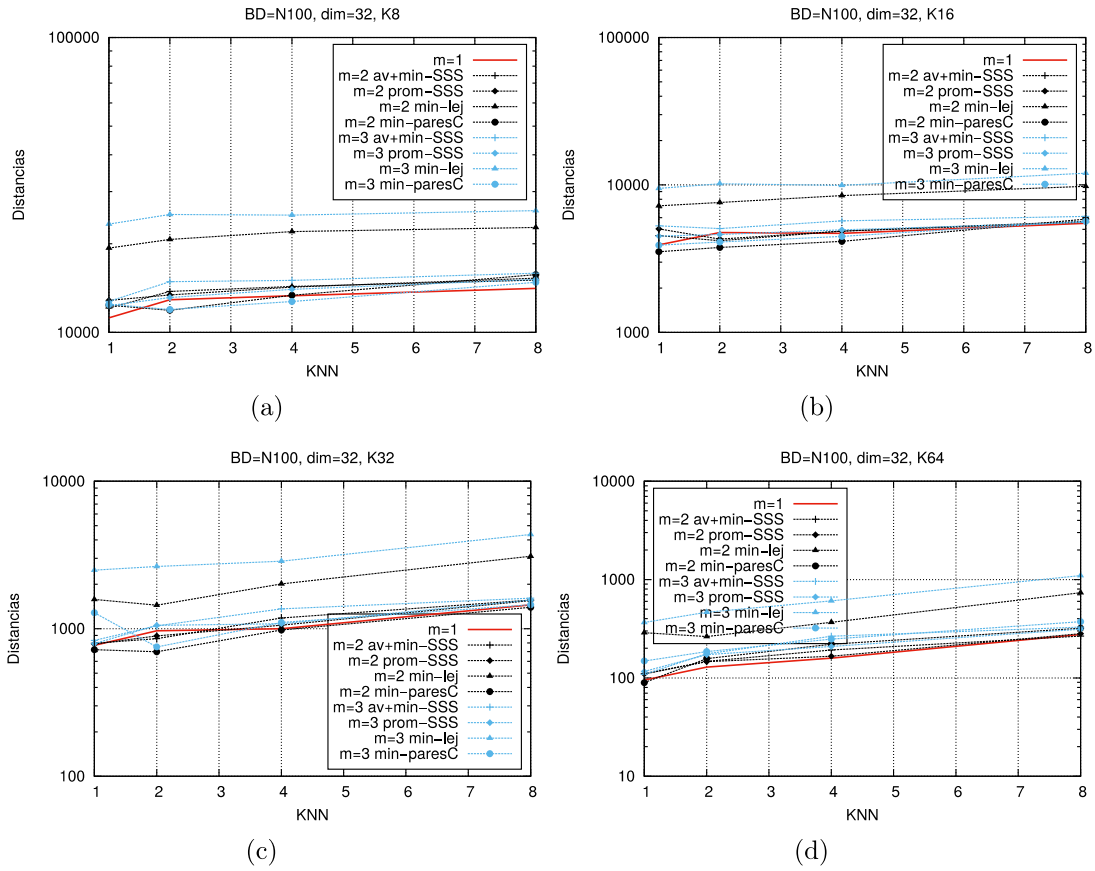


Figura 4.3: Dimensión 32 con m =3, k ={8,16,32,64}

de strings (vectores de dimensión) de una base de datos de palabras.

La base de datos de objetos reales Spanish, es un conjunto de 86061 histogramas de strings (vectores de dimensión) de una base de datos de palabras. La distancia usada entre objetos de esta base de datos fue la distancia de edición.

La base de datos de objetos reales Colors, es un conjunto de 112,544 histogramas de colores (vectores de dimensión 112) de una base de datos de imágenes. La distancia usada entre objetos de esta base de datos fue la Euclidiana.

La base de datos de objetos reales Nasa es un conjunto de 40,150 vectores (de dimensión 20), generados de imágenes descargadas de la NASA. La distancia usada entre objetos de esta base de datos también fue la Euclidiana.

El número de clases usados para los experimentos en BDR fue k = 8, 16, 32, 64. Donde cada clase se conformó por m = 2, 3, 4 elementos por clase.

BD	numG	m	selectP	critDist
colors	8	2	rand	prom
nasa	16	3	cerc	max
–	32	4	lej	min
–	64	–	SSS	cabG
–	–	–	paresC	av+min

Tabla 4.2: Tipos de pruebas para la BDR para *Colors* y *NASA*

BD	numG	m	selectP	critDist
English	8	2	rand	prom
Spanish	16	3	SSS	max
–	32	4	paresC	min
–	64	–	–	cabG
–	–	–	–	av+min

Tabla 4.3: Tipos de pruebas para la BDR para *English* y *Spanish*

4.2.1. Base de Datos de *Colors*

Variando m

En las Figuras 4.4 y 4.5 se agrupan las gráficas correspondientes a la base de datos *Colors*, variando m =2,3 se prueban los diferentes tipos de criterios de distancia y selección de permutantes. Nuevamente se puede observar que al aumentar el número de clases es necesario una menor cantidad de distancias requeridas. A la vez puede observarse en el cuadro 4.4 las combinaciones que tienen una leve mejoría o igualan el rendimiento de las permutaciones originales.

Variando K

Otro factor importante en este tipo de búsquedas es el valor del número de vecinos a recuperarse. En la Figura 4.6 se muestra el desempeño al variar este parámetro. Para esta base de datos el mejor desempeño se tiene

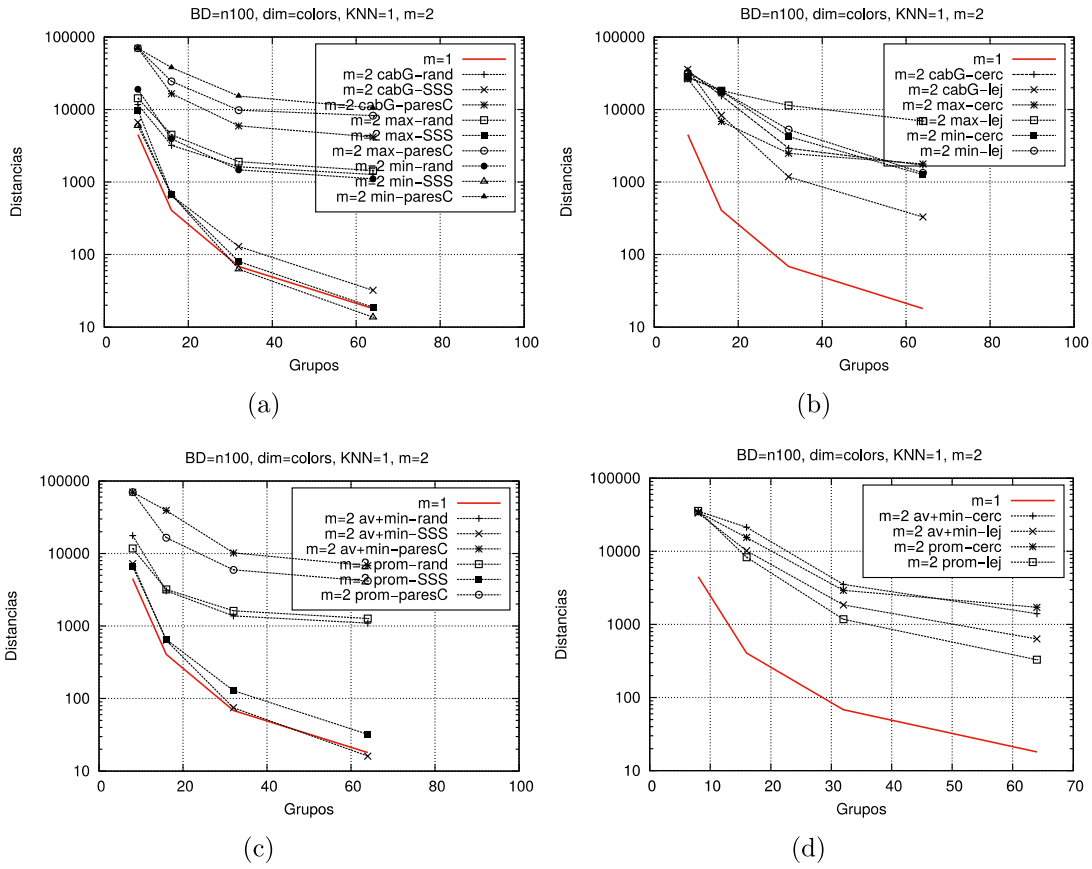


Figura 4.4: Base de datos *Colors* con m =2, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	cabG	SSS		3	av+min	SSS
2	max	SSS		3	prom	SSS
2	min	SSS		4	cabG	SSS
2	av+min	SSS		4	max	SSS
2	prom	SSS		4	min	SSS
3	cabG	SSS		4	av+min	SSS
3	max	SSS		4	prom	SSS
3	min	SSS				

Tabla 4.4: Se muestran los combinaciones con mejor rendimiento para la base de datos *Colors*

Discusión

De las gráficas mostradas se observa que el mejor desempeño se tiene con la propuesta presentada en esta tesis: *av+min* y *SSS*. Consideremos que se están mostrando

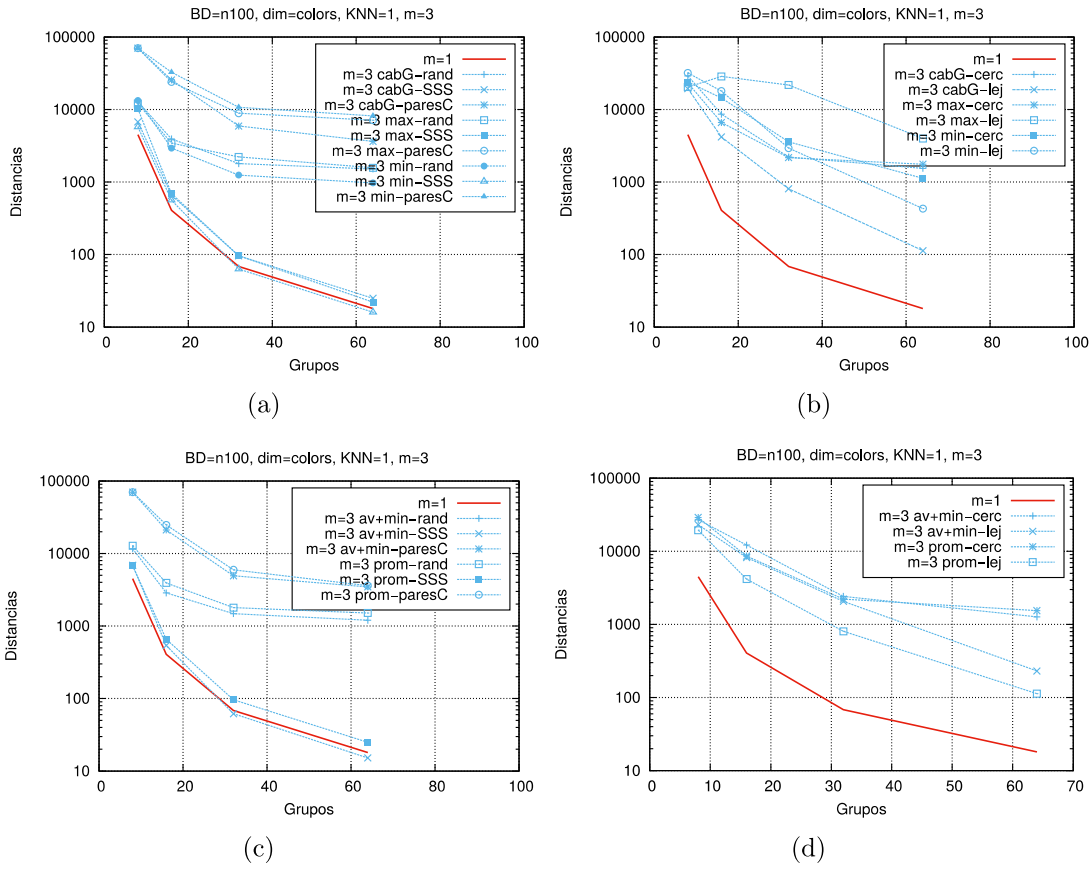


Figura 4.5: Base de datos *Colors* con m =3, k ={8,16,32,64}

los resultados de las distancias totales. En segundo lugar, el criterio de *prom* y *SSS*. El resto de los criterios no tiene un desempeño competitivo, nótese que esto no era tan claro en las bases de datos sintéticas.

4.2.2. Base de Datos de *NASA*

Variando m

Para la base de datos de NASA se agrupan las gráficas en las Figuras 4.7 y 4.8 variando m =2,3. Se prueban los diferentes tipos de criterios de distancia y selección de permutantes.

Nuevamente se puede observar que al aumentar el número de clases es necesario una menor cantidad de distancias requeridas para recuperar los elementos.

En el cuadro 4.5 se presentan las combinaciones que tienen una leve mejoría o

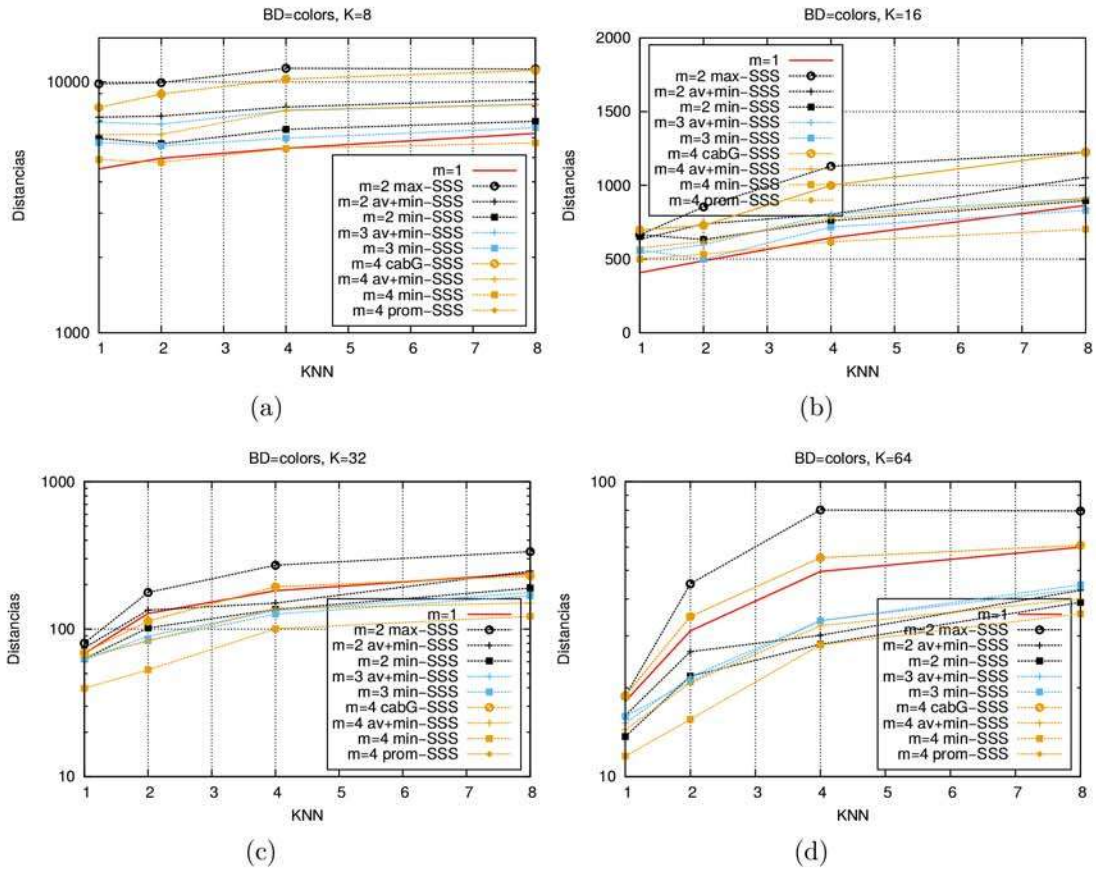


Figura 4.6: Base de Datos *Colors* con KNN={1,2,4,8}

igualan el rendimiento de las permutaciones originales (m=1).

m	critDist	selectP		m	critDist	selectP
2	cabG	SSS		3	min	SSS
2	max	SSS		3	prom	SSS
2	min	rand		3	av+min	SSS
2	min	SSS		4	cabG	SSS
2	av+min	rand		4	mas	SSS
2	av+min	SSS		4	min	rand
2	prom	SSS		4	min	SSS
3	cabG	SSS		4	av+min	rand
3	max	SSS		4	av+min	SSS
3	min	rand		4	prom	SSS

Tabla 4.5: Se muestran los combinaciones con mejor rendimiento para la base de datos *NASA*

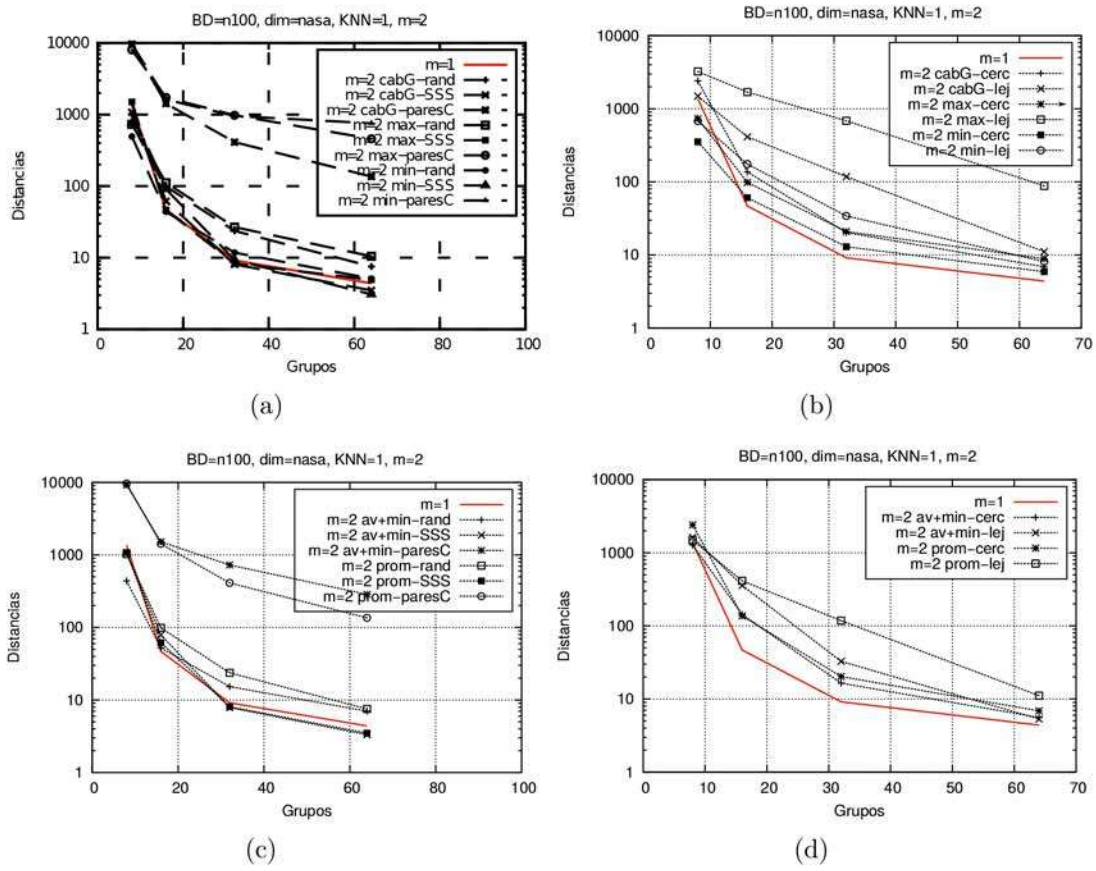


Figura 4.7: Base de datos *NASA* con m =2, k ={8,16,32,64}

Variado K

Discusión

En las gráficas se puede observar que el desempeño de las combinaciones de criterios es mucho mas uniforme debido a la naturaleza de la base de datos. Aunque es de resaltarse que nuevamente el criterio propuesto en este trabajo es mejor que la técnica original, es decir, *av+min* en conjunto con *SSS*; otro criterio con buen desempeño es *prom* y *SSS*. Véase la primera gráfica de las figuras 4.7 y 4.8.

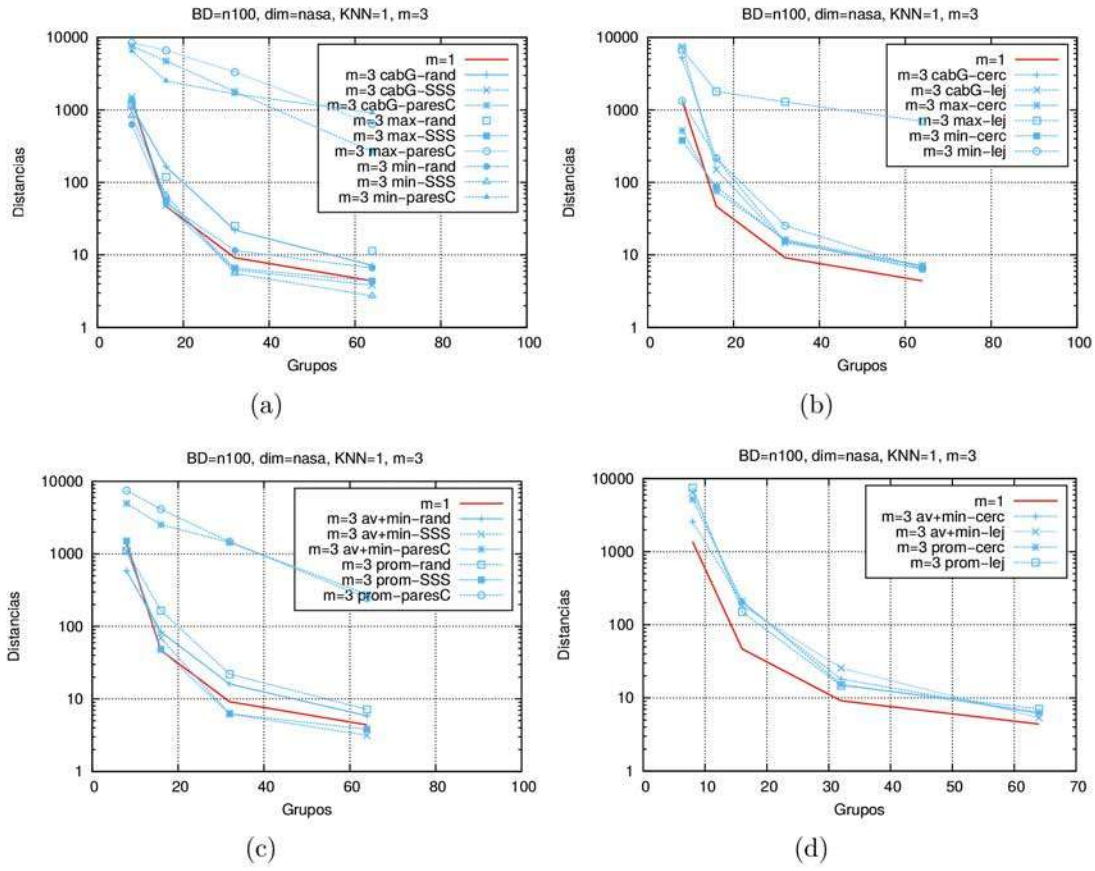


Figura 4.8: Base de datos *NASA* con m =3, k ={8,16,32,64}

4.2.3. Base de Datos de *English*

Variando m

En las Figuras 4.10 y 4.11, se agrupan las gráficas correspondientes a la base de datos *English*, variando m =2,3. Se prueban los diferentes tipos de criterios de distancia y selección de permutantes.

Para esta base de datos y debido al mal desempeño que presentaron los criterios de *lejanos* y *cercanos* se decidió no incluirlos en lo sucesivo.

En la Figura 4.10 se muestran los distintos tipos de combinaciones entre todos los criterios exceptuando los mencionados anteriormente. En el cuadro 4.6 se muestran las combinaciones que tienen una leve mejoría o igualan el rendimiento de las permutaciones originales.

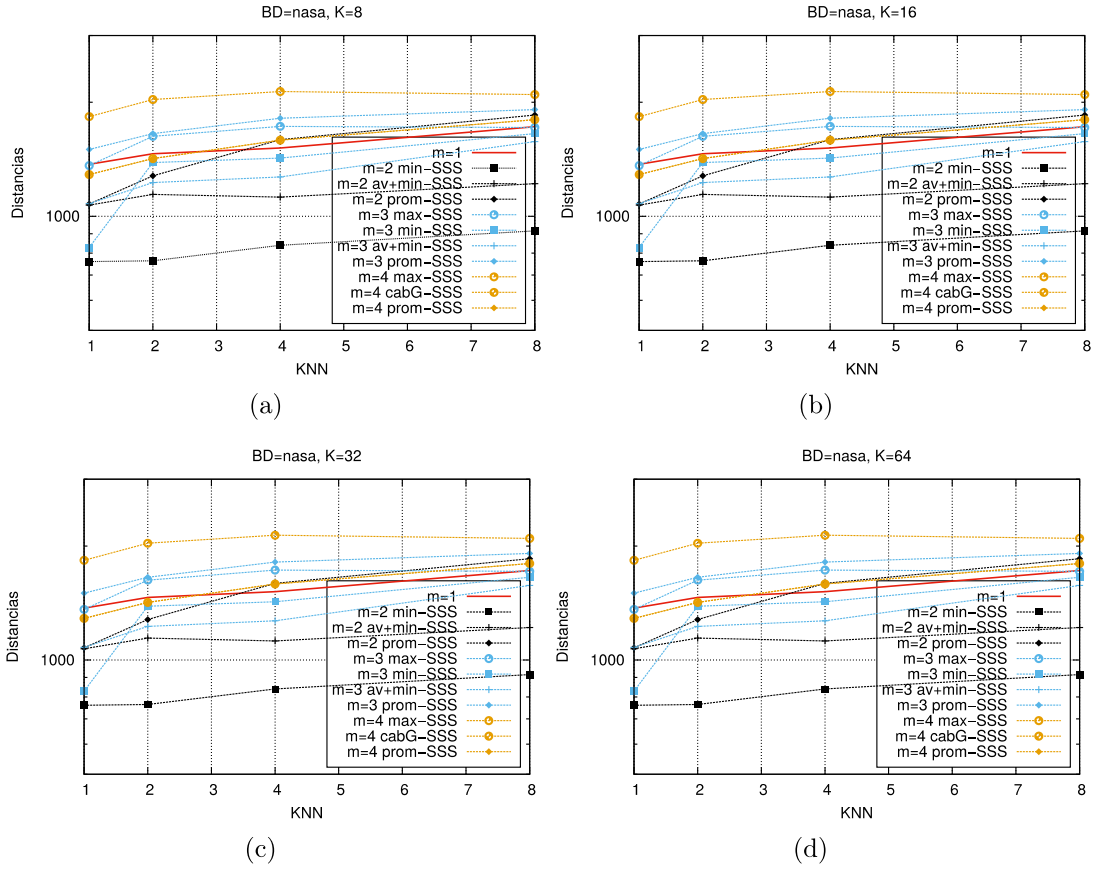


Figura 4.9: Base de Datos *NASA* con KNN={1,2,4,8}

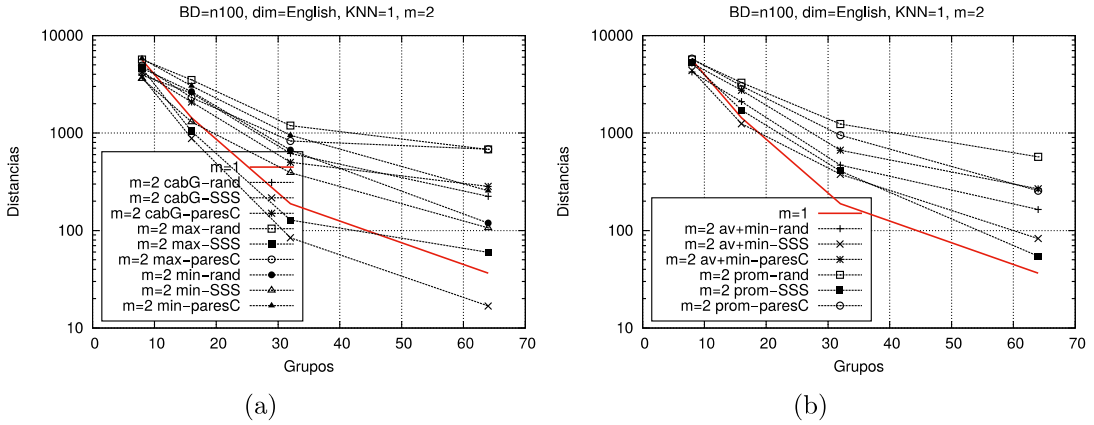
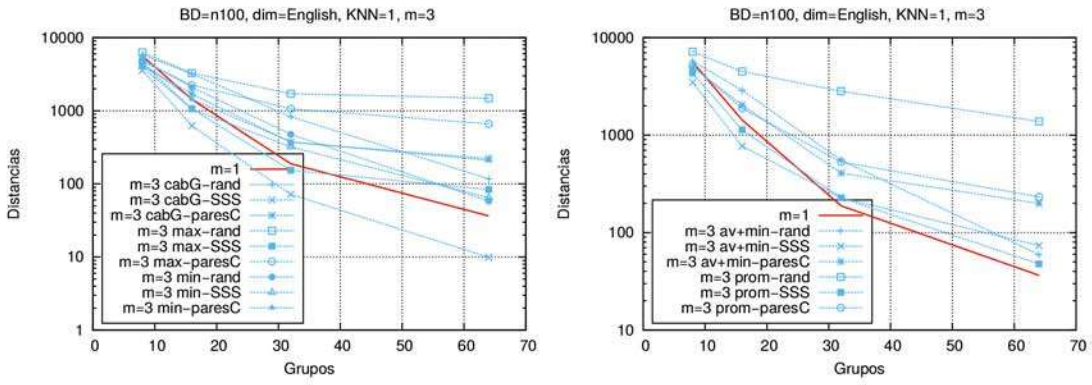


Figura 4.10: Base de datos *English* con m =2, k ={8,16,32,64}

Variando K

Discusión

Para esta base de datos el criterio de los permutantes originales tiene un buen desempeño, sin embargo, la combinación que mejora considerablemente el rendimiento



(a) (b)

Figura 4.11: Base de datos *English* con m =3, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	cabG	SSS		3	av+min	SSS
2	max	SSS		3	prom	SSS
2	av+min	SSS		4	cabG	SSS
2	prom	SSS		4	max	SSS
3	cabG	SSS		4	av+min	SSS
3	max	SSS		4	prom	SSS

Tabla 4.6: Se muestran los combinaciones con mejor rendimiento para la base de datos *English*.

es *cabecera de clase* y *SSS*. Después vienen los criterios propuestos en este trabajo *av+min* y *SSS*, así como *prom* y *SSS*.

El resto de las combinaciones no son competitivos para esta base de datos.

4.2.4. Base de Datos de *Spanish*

Variando m

En las Figuras 4.13 y 4.14 se agrupan las gráficas correspondientes a la base de datos *Spanish*, variando m =2,3. Los criterios probados fueron todos exceptuando los *lejanos* y *cercanos* debido a su mal desempeño para esta base de datos.

En la tabla 4.7 se muestran los mejores 2 criterios por gráfica. Por ejemplo, para la primer gráfica (esquina superior izquierda) los mejores criterios son ...

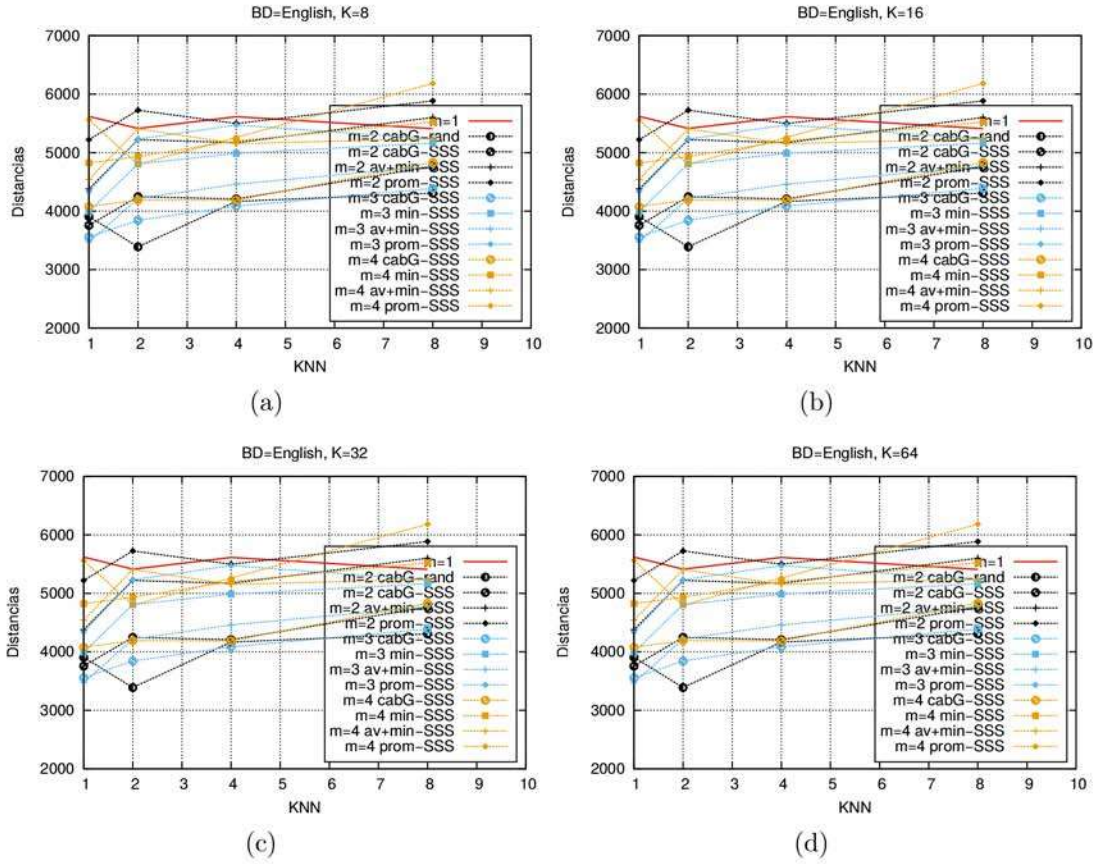


Figura 4.12: Base de Datos *English* con KNN={1,2,4,8}

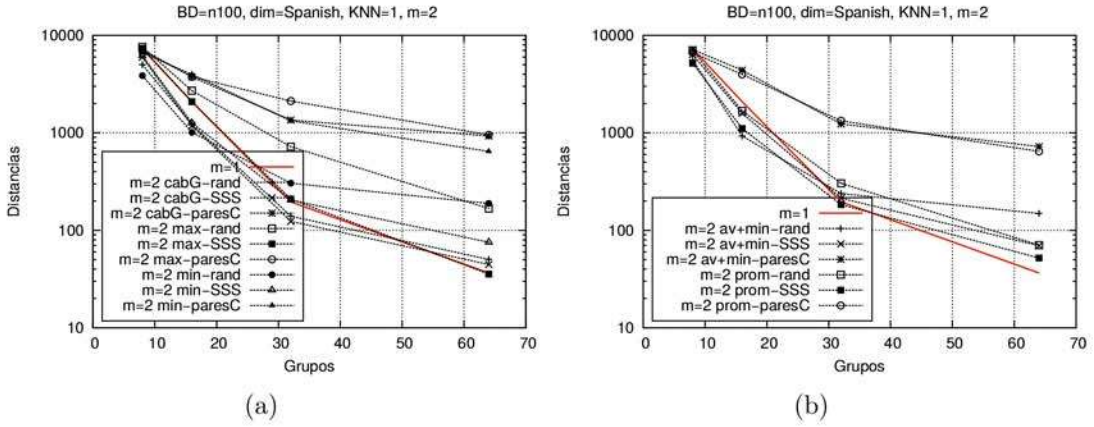
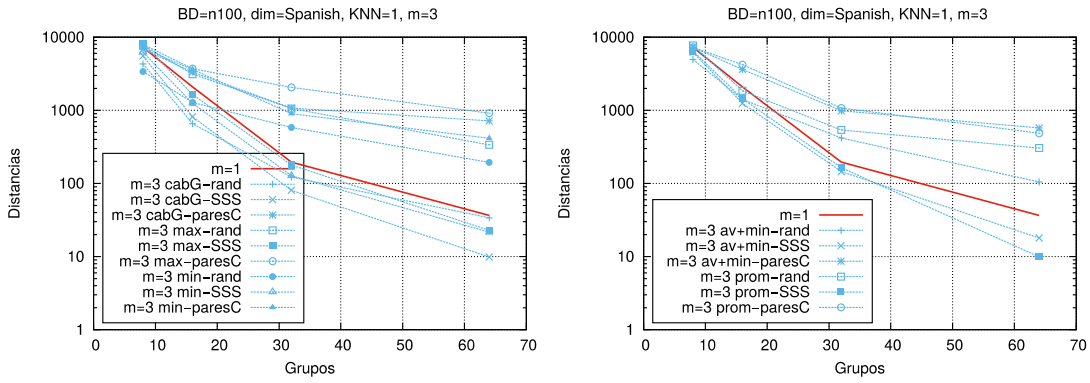


Figura 4.13: Base de datos *Spanish* con m =2, k ={8,16,32,64}

Variando K

En la Figura 4.15 se agrupan las gráficas con un mejor rendimiento de la BD de *Spanish* con cada uno de los distintos números de permutaciones por clase, ahora se



(a) (b)

Figura 4.14: Base de datos *Spanish* con m =3, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	cabG	rand		3	min	SSS
2	cabG	SSS		3	av+min	SSS
2	max	SSS		3	prom	SSS
2	min	SSS		4	cabG	rand
2	av+min	rand		4	cabG	SSS
2	av+min	SSS		4	max	SSS
2	prom	SSS		4	min	SSS
3	cabG	rand		4	av+min	SSS
3	cabG	SSS		4	prom	SSS
3	max	SSS				

Tabla 4.7: Se muestran los combinaciones con mejor rendimiento para la base de datos *Spanish*

tiene que al variar el número de vecinos KNN=1,2,4,8.

Discusión

Para esta última base de datos funciona muy bien el criterio de *CabG* con $m = 2$. Esto sin embargo hace pensar que los clases de permutantes no son necesariamente buenos para todas las bases de datos. Esta en particular parece trabajar mucho mejor con los permutantes originales PeBI.

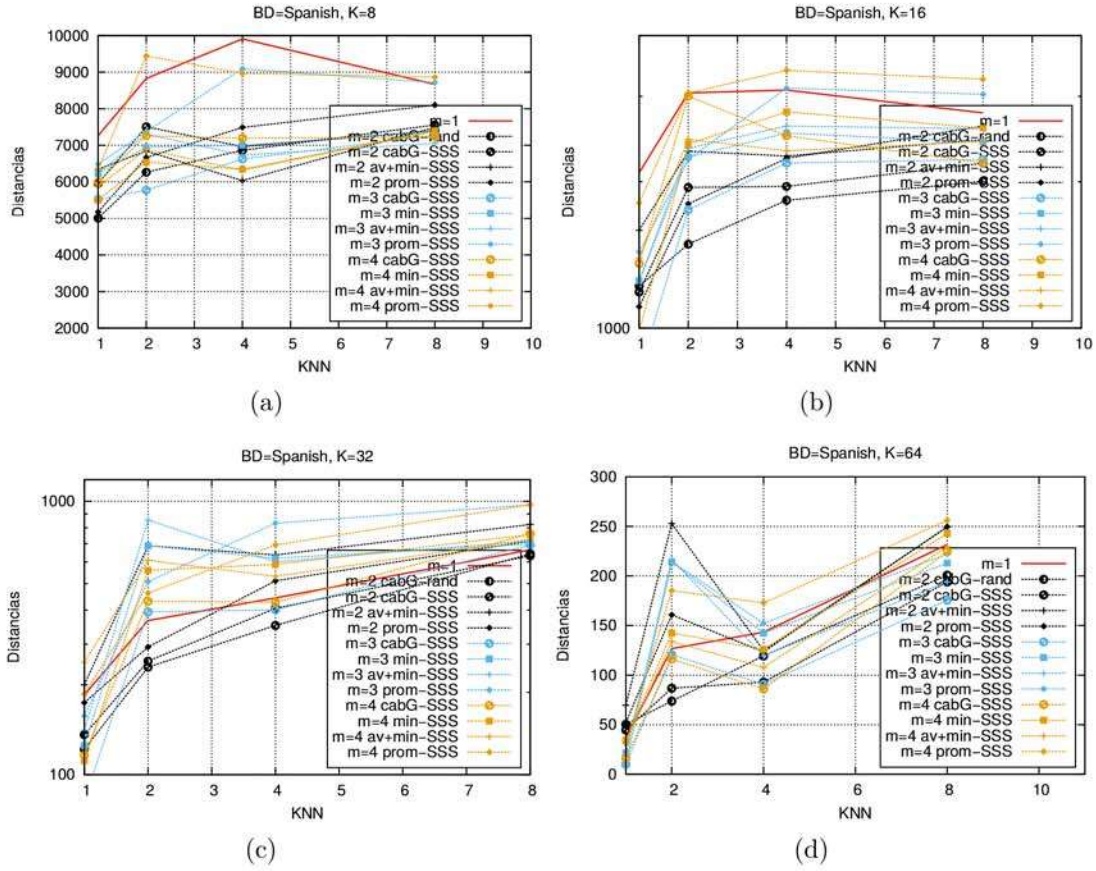


Figura 4.15: Base de Datos *Spanish* con KNN={1,2,4,8}

Capítulo 5

Conclusiones

Una búsqueda por similitud es usada en bases de datos multimedia, en la cual es necesario encontrar aquellos elementos más parecidos a una consulta dentro de la base de datos. Una opción es modelarlo como un espacio métrico, donde necesitamos una función de distancia, que nos permita conocer entre dos elementos qué tan similares son (esta función está definida por expertos del tema a clasificar). El inconveniente de esta técnica es lo costosa que puede resultar la función de distancia por lo que hace necesario encontrar distintas formas de ahorrar cálculos para agilizar el tiempo de búsqueda.

El algoritmo con mejor desempeño es el basado en permutaciones, sin embargo, este tipo de índice no se ha mejorado utilizando menos espacio. Todos los intentos de comprimir el índice provocan pérdida de precisión en la hora de la consulta.

En este trabajo se presenta expuesto en [11] (Efficient Group of Permutants for Similary Searching). En dicho trabajo se propone usar un clase de permutantes en lugar de un solo elemento por permutante. Es decir, se propone usar una permutación de clases.

La propuesta de esta tesis se centró en emplear un nuevo criterio de distancia para la formación de los clases. La técnica propuesta fue:

- *av+min* es D_{av+min} que consiste en la suma del promedio de distancias de todos los objetos del clase D_{prom} más la distancia mínima de todos los objetos del clase

D_{min}.

En general, av+m+ín tienen un desempeño variante dependiendo del tipo de Base de datos. Para el criterio de selección de permutantes con un mejor desempeño SSSlejanos como sugerencia. Cabe resaltar que, con esta técnica de clases es posible usar permutaciones mas cortas que las necesarias en la técnica original.

Apéndice A

Gráficas complementarias

En este capítulo se mostrarán las gráficas obtenidas de manera mas detallada.

A.1. Bases de datos sintéticas

Para la realización de las pruebas realizadas en la BDS se conformaron por 3 dimensiones diferentes, 4 conformaciones distintas de clases, cada uno de los cuales podía tener 3 cantidades diferentes de elementos, comparados contra la técnica de permutantes original,en combinación de los 5 tipos de selección de permutantes, con 5 distancias diferentes. Como primera parte de los experimentos se pondrán a prueba los distintos tipos de criterios contra la técnica original cuando KNN=1, obteniendo una lista de los tendrían un mejor comportamiento. Como segunda parte de los experimentos de la lista obtenida anteriormente se seleccionaran los que obtengan un mejor comportamiento, estos tipos de criterios seleccionados se pondrán a prueba contra la técnica original, pero variando KNN=$\{1,2,4,8\}$ Los criterios para las pruebas realizadas para las BDS se muestras en la tabla A.1:

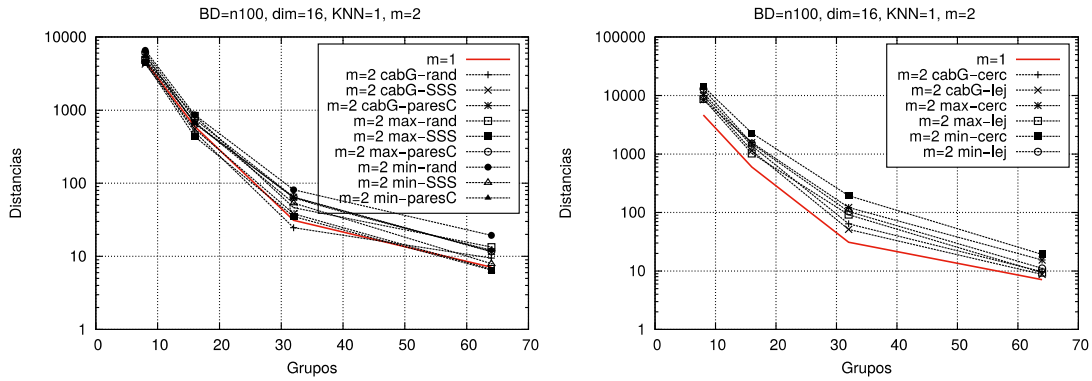
A.1.1. Dimensiones 16 variando m

En las siguientes Figuras A.1, A.2, A.3 puede observase el comportamiento para cada una de las combinaciones obtenidas de los distintos criterios. Teniendo que las

dim	numG	m	selectP	critDist
16	8	2	rand	prom
32	16	3	cerc	max
64	32	4	lej	min
–	64	–	SSS	cabG
–	–	–	paresC	av+min

Tabla A.1: Tipos de pruebas para las BDS.

mejores combinaciones que igualan o tienen un comportamiento mejor que la técnica original para la dimensión 16 se encuentran en la siguiente tabla A.2. De esta tabla obtenemos los siguientes parámetros que son los que tienen mejor rendimiento: con m=2 *cabG-rand*, *max-SSS*, *prom-rand*, con m=3 *cabG-rand*, *cabG-SSS*, *prom-rand*, *prom-SSS*, con m=4 *cabG-SSS*, *prom-SSS*, *prom-rand*, *av+min-SSS*.



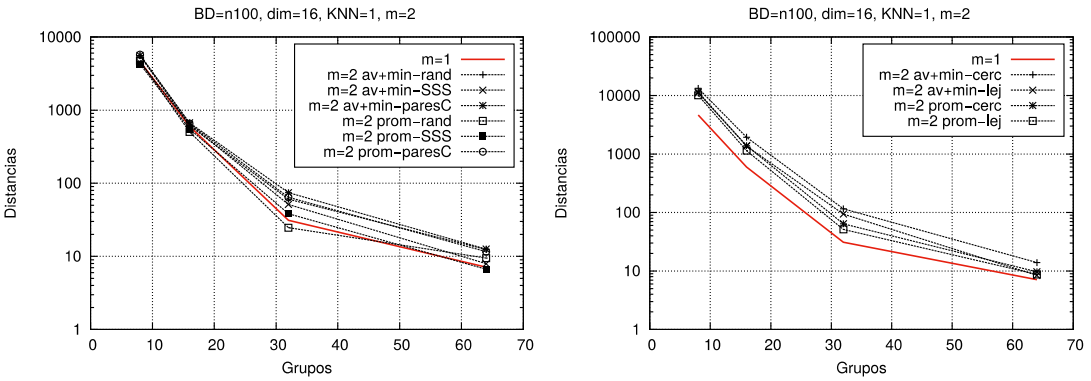


Figura A.1: Dimensión 16 con m =2, k ={8,16,32,64}

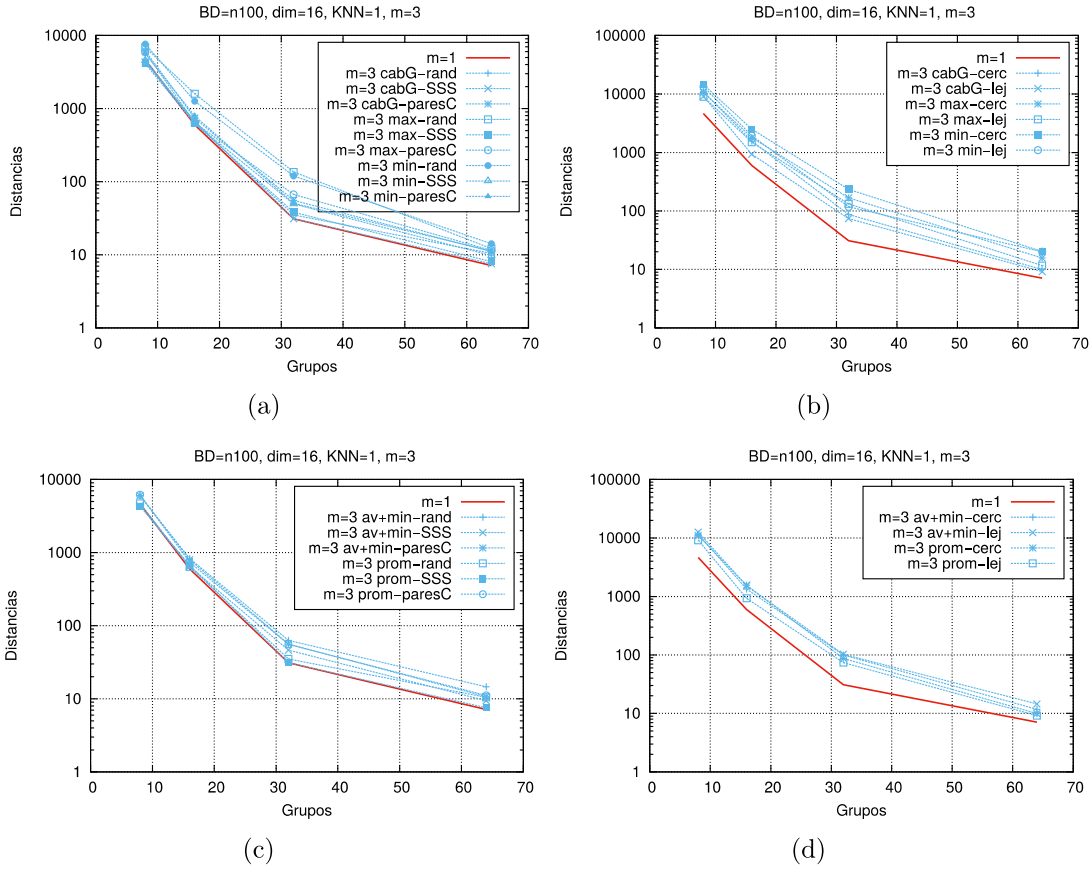


Figura A.2: Dimensión 16 con m =3, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	cabG	rand		3	prom	rand
2	cabG	SSS		3	prom	SSS
2	max	SSS		4	cabG	rand
2	prom	rand		4	cabG	SSS
2	prom	SSS		4	max	SSS
3	cabG	rand		4	prom	rand
3	cabG	SSS		4	prom	SSS
3	max	SSS		4	av+min	SSS

Tabla A.2: Se muestran los combinaciones con mejor rendimiento para la dimensión 16

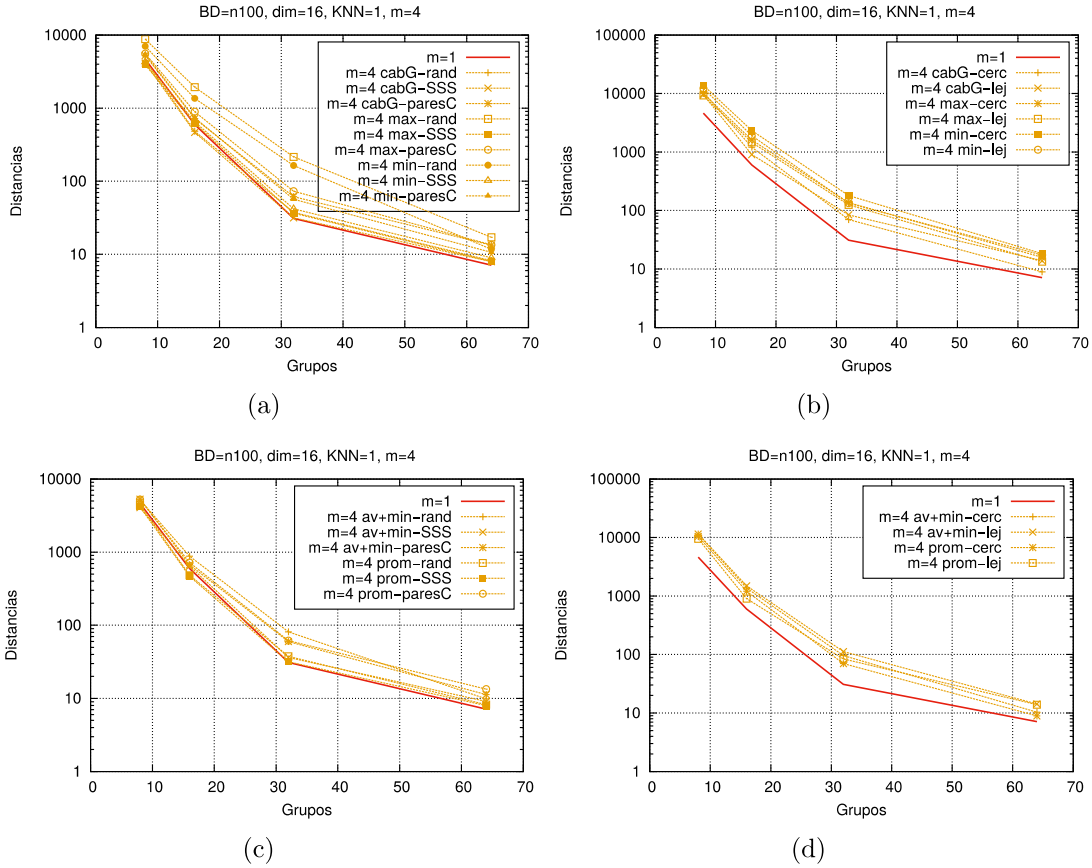


Figura A.3: Dimensión 16 con m =4, k ={8,16,32,64}

A.1.2. Dimensiones 32 variando m

En las siguientes Figuras A.4, A.5, A.6 se muestran las gráficas correspondientes a la dimensión 32, con las diferentes combinaciones de criterios. Obteniendo la siguiente tabla A.3 que muestran los parámetros con un desempeño mejor o similar al de la técnica original. De esta tabla obtenemos los siguientes parámetros que son los que tienen mejor rendimiento: con m=2 *min-paresC*, *av+min-paresC*, *prom-paresC*, con m=3 *cabG-paresC*, *prom-SSS*, *prom-paresC*, con m=4 *cabG-paresC*, *av+min-paresC*, *prom-paresC*.

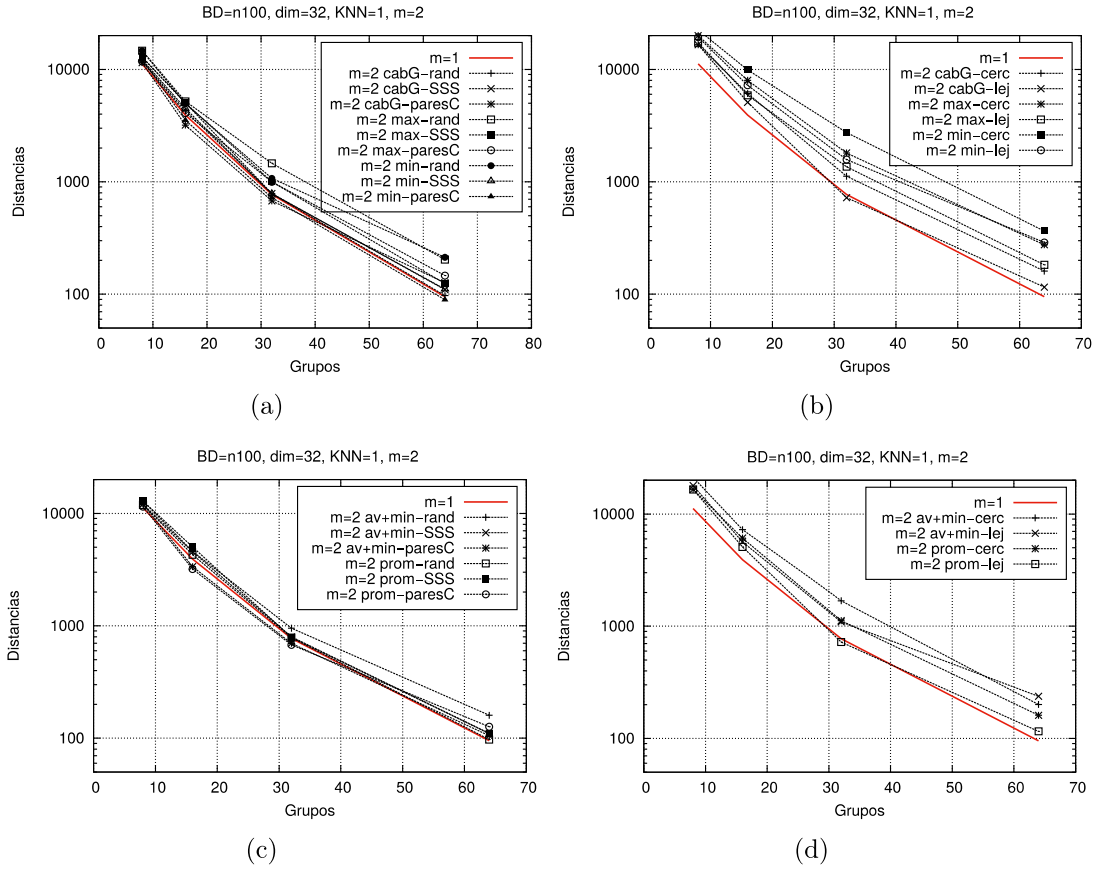


Figura A.4: Dimensión 32 con m =2, k ={8,16,32,64}

A.1.3. Dimensiones 64 variando m

En las siguientes Figuras A.7, A.8, A.9 se agrupan las gráficas correspondientes a la dimensión 64, con los diferentes tipos de criterios, obteniendo nuevamente la

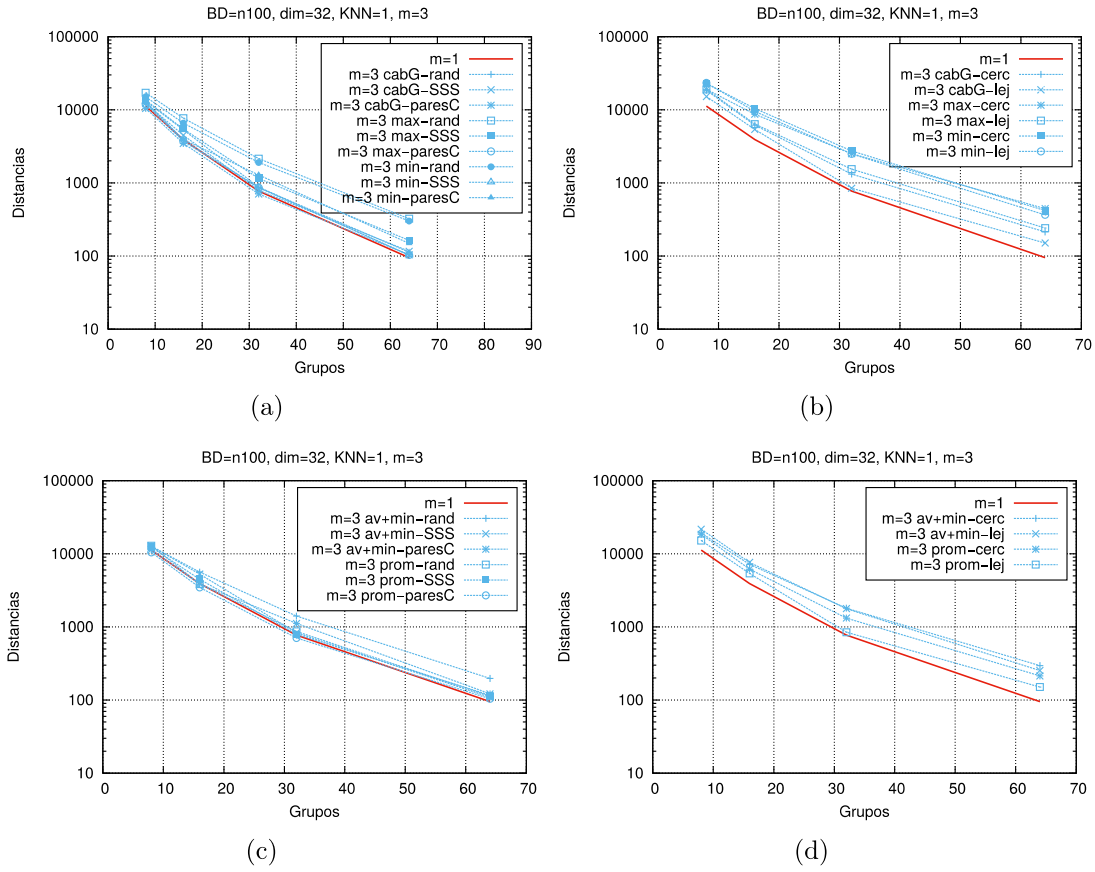


Figura A.5: Dimensión 32 con m =3, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	cabG	rand		3	cabG	paresC
2	cabG	SSS		3	cabG	lej
2	cabG	parsC		3	min	SSS
2	cabG	lej		3	max	paresC
2	min	SSS		3	av+min	SSS
2	min	paresC		3	prom	rand
2	av+min	SSS		3	prom	SSS
2	av+min	paresC		3	prom	paresC
2	prom	rand		3	prom	lej
2	prom	SSS		4	cabG	SSS
2	prom	paresC		4	cabG	paresC
2	prom	lej		4	min	paresC
3	cabG	rand		4	av+min	paresC
3	cabG	SSS		4	prom	paresC

Tabla A.3: Se muestran los combinaciones con mejor rendimiento para la dimensión 32

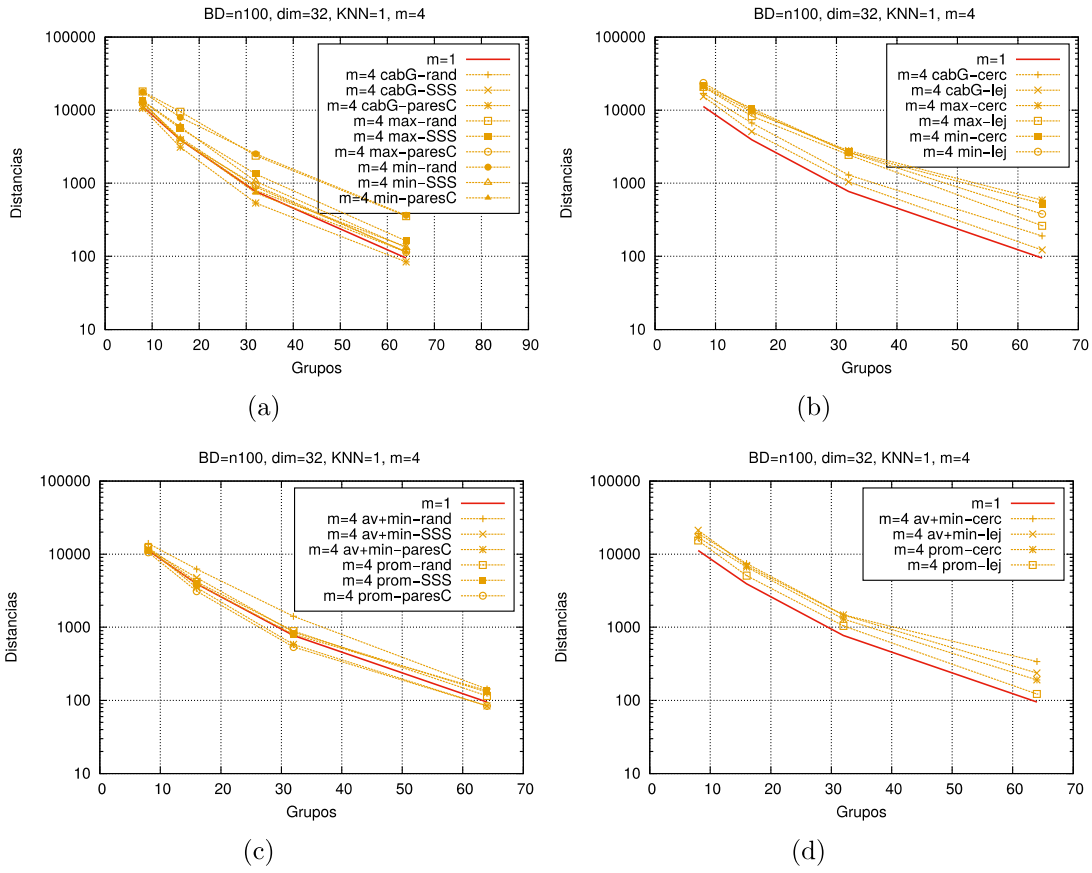


Figura A.6: Dimensión 32 con m =4, k ={8,16,32,64}

tabla A.4 que muestra los parámetros con un desempeño mejor o similar al de la técnica original. De aquí obtenemos los parámetros con mejor desempeño, que son: con m={2,3,4} *cabG-paresC*, *av+min-paresC*, *prom-paresC*.

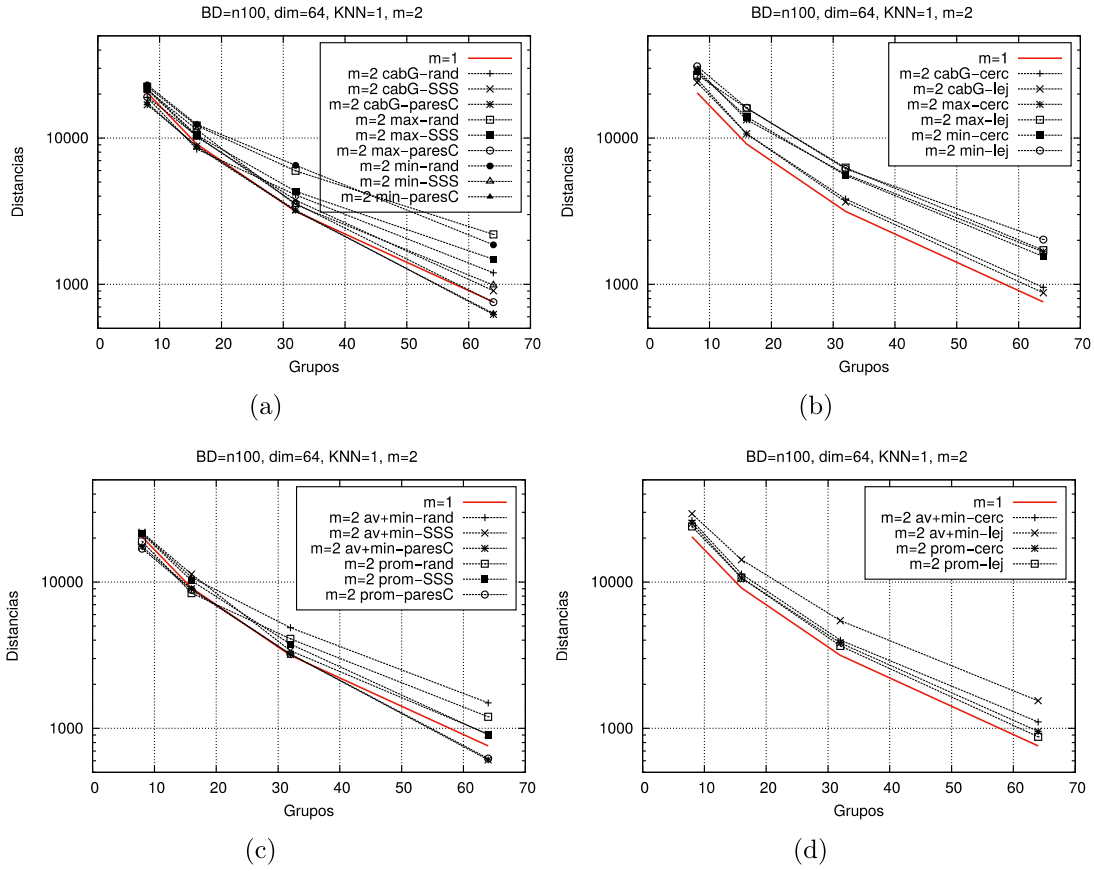


Figura A.7: Dimensión 64 con m =2, k ={8,16,32,64}

A.1.4. Dimensiones 16 variando KNN

En la siguiente Figura A.10 se agrupan las gráficas con un mejor rendimiento de la dimensión 16. En esta Figura A.10 puede observarse que con k =8, los distintos parámetros tienen un comportamiento igual que al de la técnica original, los únicos parámetros con un comportamiento ligeramente peor son: con m =3 *cabG-rand prom-rand* y con m =4 *cabG-rand*. En k =16 el parámetro con m =4 *av+min-SSS* es el parámetro con un comportamiento igual que el de la técnica original. En k =32 los parámetros no muestran mejoría contra la técnica original. En k =64 el parámetros

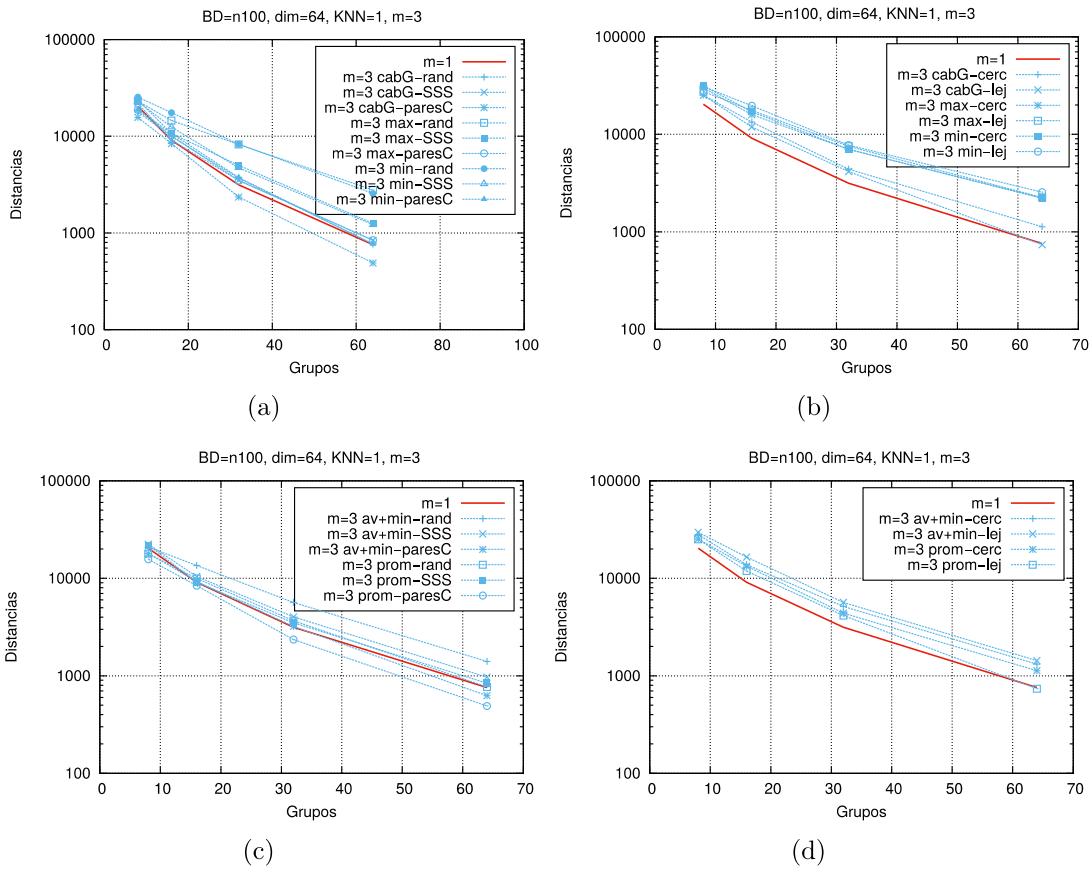


Figura A.8: Dimensión 64 con m =3, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	cabG	paresC		3	max	parsC
2	min	paresC		3	prom	paresC
2	min	SSS		3	prom	rand
2	max	paresC		3	prom	SSS
2	av+min	paresC		3	av+min	paresC
2	prom	paresC		4	cabG	paresC
3	cabG	paresC		4	prom	paresC
3	cabG	SSS		4	av+min	paresC
3	min	paresC				

Tabla A.4: Se muestran los combinaciones con mejor rendimiento para la dimensión 64

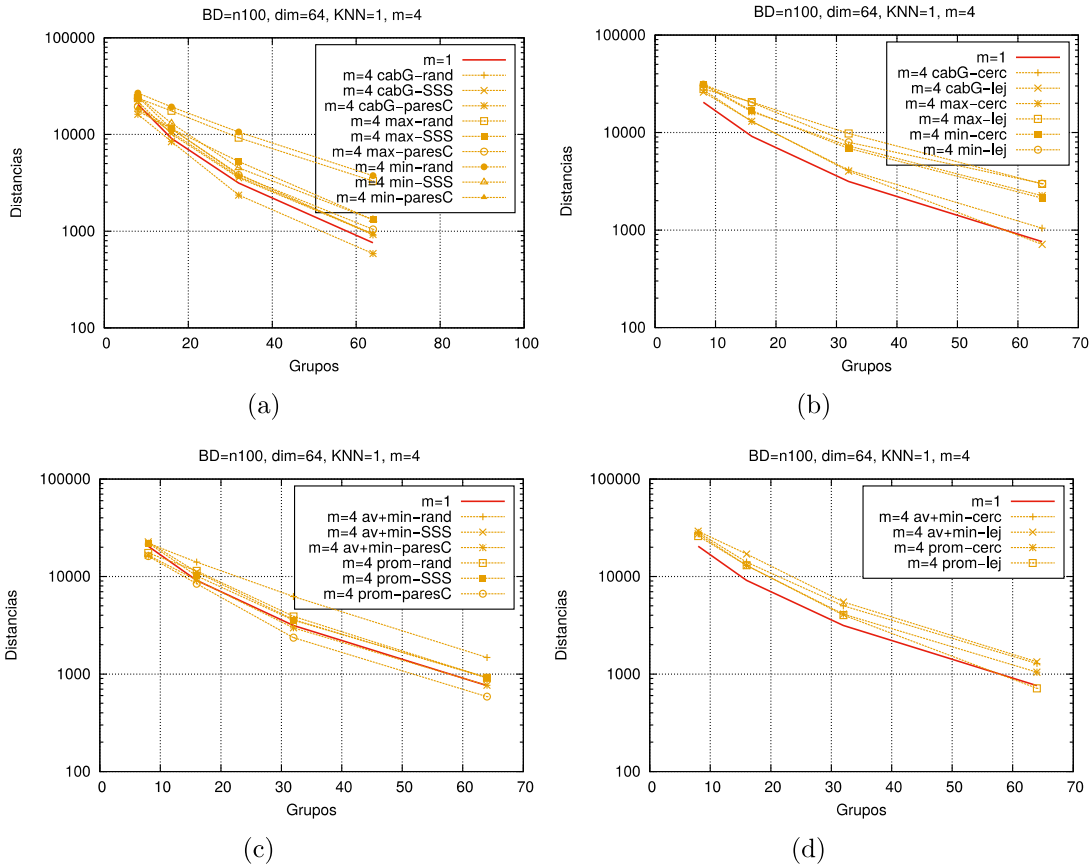


Figura A.9: Dimensión 64 con m =4, k ={8,16,32,64}

con un comportamiento igual al de la técnica original es con m =2 *max-SSS*.

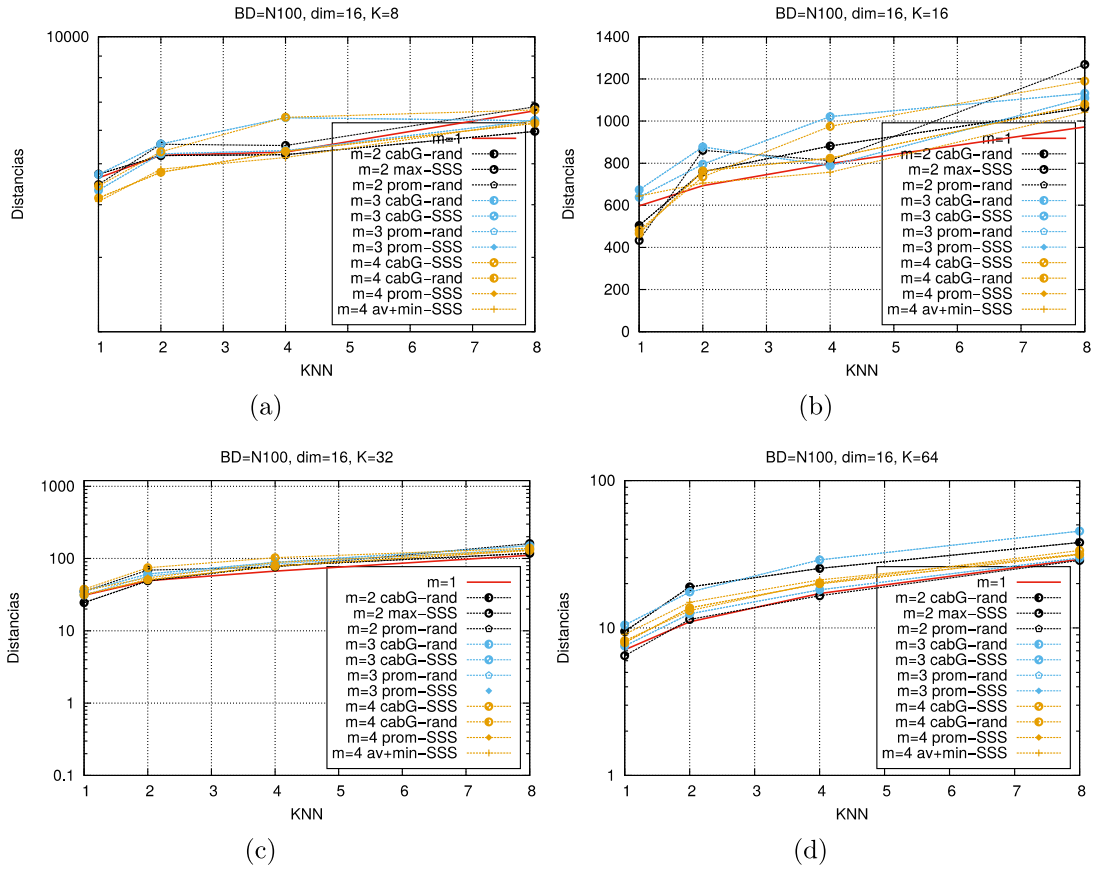


Figura A.10: Dimensión 16 con KNN={1,2,4,8}

A.1.5. Dimensiones 32 variando KNN

En la siguiente Figura A.11 se agrupan las gráficas con un mejor rendimiento de la dimensión 32. En k =8 tenemos que los parametros con un rendimiento ligeramente mejor que la técnica original son: con m =2 *min-paresC*, con m = 3 *cabG-paresC*, *prom-paresC* y con m =4 *cabG-paresC*, *av+min-paresC*, *prom-paresC*. En k =16 los parámetros con con un comportamiento igual que la tecnica origial son: m =2 *av+min-SSS*, *prom-SSS*, con m =3 *prom-SSS*, con un comportamiento ligeramente mejor son: con m =2 *min-parsC*, con m =3 *cabG-parsC*, *prom-paresC*, con m =4 *cabG-parsC*, *av+min-parsC*, *prom-paresC*. En k =32 los parámetros con con un comportamiento igual que la tecnica origial son: con m =2 *min-paresC*, *av+min-*

SSS, *prom-SSS*, con $m = 3$ *prom-SSS*, con un comportamiento ligeramente mejor son: m =3 *cabG-parsC*, *prom-paresC*, con m =4 *cabG-parsC*, *av+min-parsC*, *prom-paresC*.

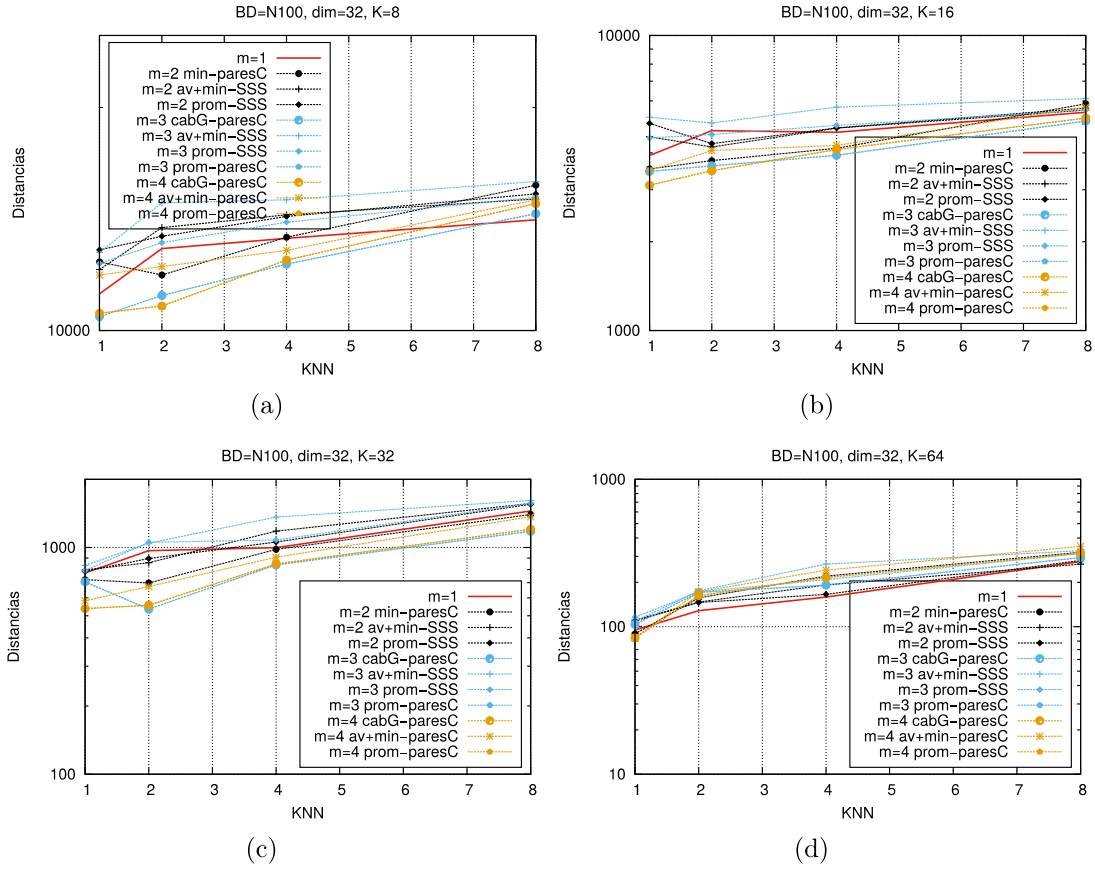


Figura A.11: Dimensión 32 con KNN={1,2,4,8}

A.1.6. Dimensiones 64 variando KNN

En la siguiente Figura A.12 se agrupan las gráficas con un mejor rendimiento de la dimensión 64. En k =8 todos los parámetros utilizados muestran una ligera mejoría contra la técnica original. En k =16 los parámetros muestran una mejoría contra la técnica original con excepción de: con m =2 *av+min-paresC* y con m =3 *av+min-paresC*. En k =32 todos los parámetros muestran una mejoría contra la técnica original. En k =64 los parámetros con un comportamiento igual que la técnica original son: con m =2 *av+min-paresC*, con m =3 *cabG-paresC*, *prom-paresC*, con m =4 *av+min-paresC*, con un comportamiento mejor que la técnica original son: con

m =2 *prom-paresC*, con m =4 *cabG-paresC*, *prom-paresC*.

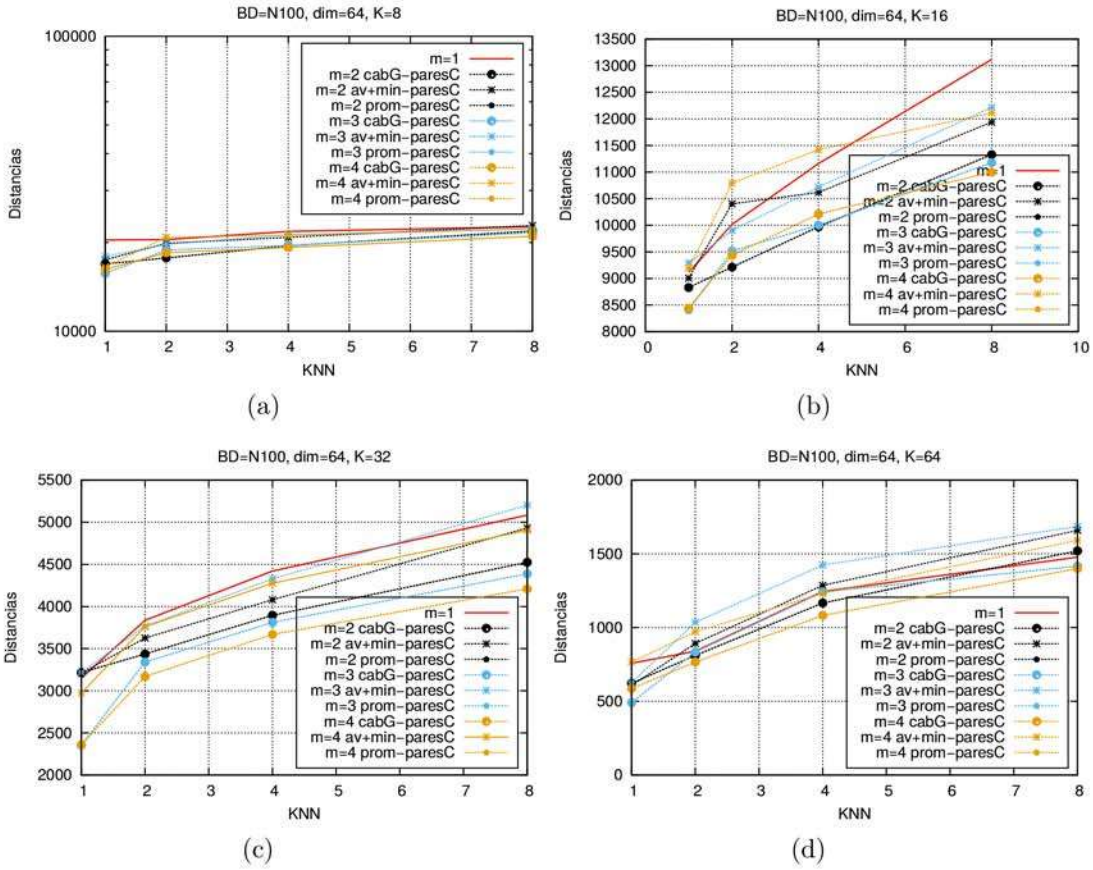


Figura A.12: Dimensión 64 con KNN={1,2,4,8}

A.2. Base de Datos Reales

Para el caso de las base de datos reales se trabajó en 4 base de datos distintas, 2 fueron de diccionarios y 2 fueron de imágenes. Para el caso de las imágenes se trabajó con 4 conformaciones distintas de clases y con 3 cantidades diferentes de elementos por clase, comparados contra la técnica de permutantes original, en combinación 5 tipos de selección de permutantes contra 5 distancias diferentes. Para el caso de los diccionarios se trabajó con 4 conformaciones distintas de clases y con 3 cantidades diferentes de elementos por clase, nuevamente son comparados contra la técnica de permutantes original, en combinación 3 tipos de selección de permutantes contra 5 distancias diferentes. En las siguientes tablas A.5 y A.6 se muestra los criterios

utilizados en cada prueba de las BSR, teniendo los experimentos para las imágenes y los diccionarios respectivamente.

BD	numG	m	selectP	critDist
colors	8	2	rand	prom
nasa	16	3	cerc	max
–	32	4	lej	min
–	64	–	SSS	cabG
–	–	–	paresC	av+min

Tabla A.5: Tipos de pruebas para la BDR para *Colors* y *NASA*

BD	numG	m	selectP	critDist
English	8	2	rand	prom
Spanish	16	3	SSS	max
–	32	4	paresC	min
–	64	–	–	cabG
–	–	–	–	av+min

Tabla A.6: Tipos de pruebas para la BDR para *English* y *Spanish*

A.2.1. Base de Datos de *Colors* y *NASA* variando m

En las siguientes Figuras A.13, A.14, A.15 y A.16, A.17, A.18 se agrupan las gráficas correspondientes a la base de datos de *Colors* y *NASA* respectivamente. De estas gráficas que contienen los distintos parámetros, se puede obtener 2 tablas A.7 y A.8 que muestra las criterios con mejor desempeños de las BDR *Colors* y *NASA* respectivamente. De la tabla A.7 de *Colors* los mejores parámetros obtenidos son : m=2 *max-SS*, *min-SSS*, *av+min-SSS*, con m=3 *min-SSS*, *av+min.SS*,con m=4 *cabG-SSS*, *min-SSS*, *av+min-SSS*, *prom-SSS*.

Y de la tabla A.8 de *NASA* tenemos que los mejores parámetros son: con m=2 *min-SSS*, *av+min-SSS*, *prom-SSS*, con m=3 *max-SSS*, *min-SSS*, *av+min-SSS*, *prom-SSS*, con m =4 *cabG-SSS*, *max-SSS*, *prom-SSS*

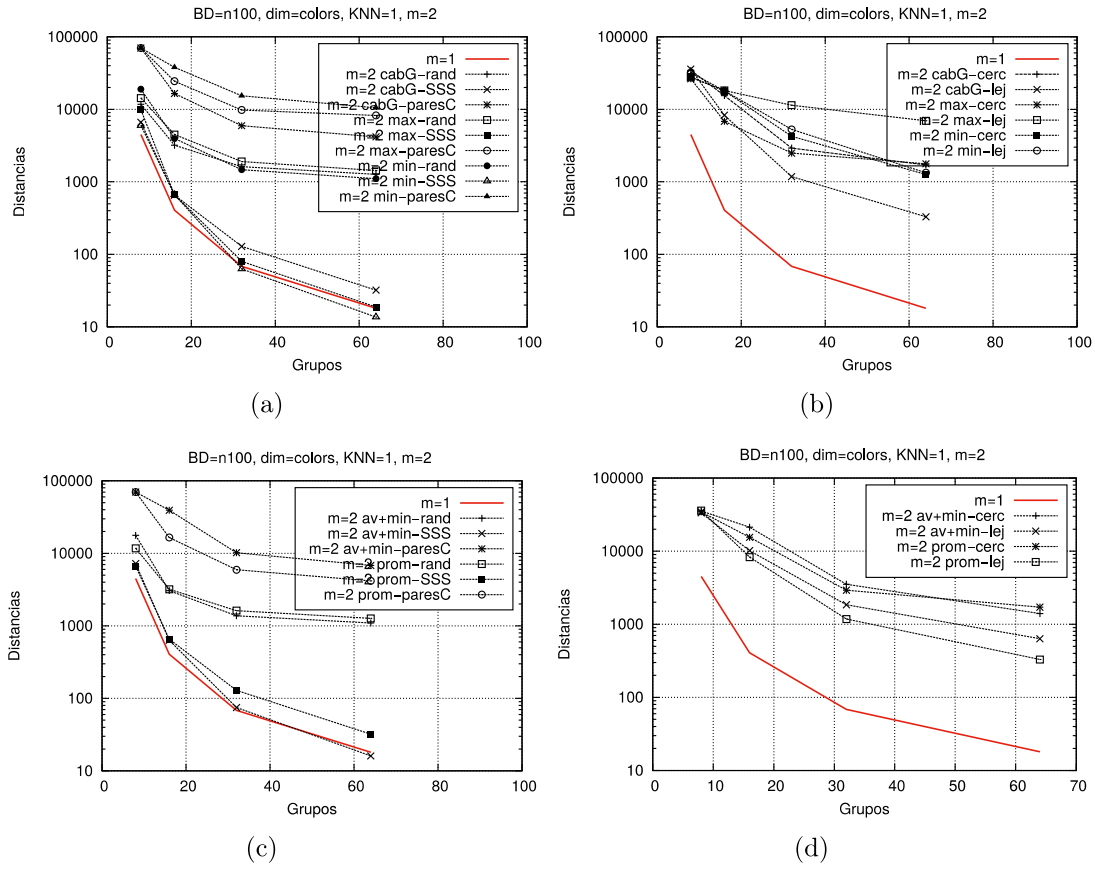


Figura A.13: Base de datos *Colors* con m =2, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	dist	SSS		3	min	SSS
2	max	SSS		3	cabG	SSS
2	dist	rand		3	cabG	rand
2	min	SSS		3	prom	SSS
2	cabG	SSS		3	prom	lej
2	cabG	rand		3	av+min	SSS
2	prom	SSS		3	av+min	rand
2	prom	lej		4	dist	SSS
2	av+min	SSS		4	max	SSS
2	av+min	rand		4	min	SSS
3	dist	lej		4	cabG	SSS
3	dist	SSS		4	prom	SSS
3	max	SSS		4	av+min	SSS

Tabla A.7: Se muestran los combinaciones con mejor rendimiento para la base de datos *Colors*

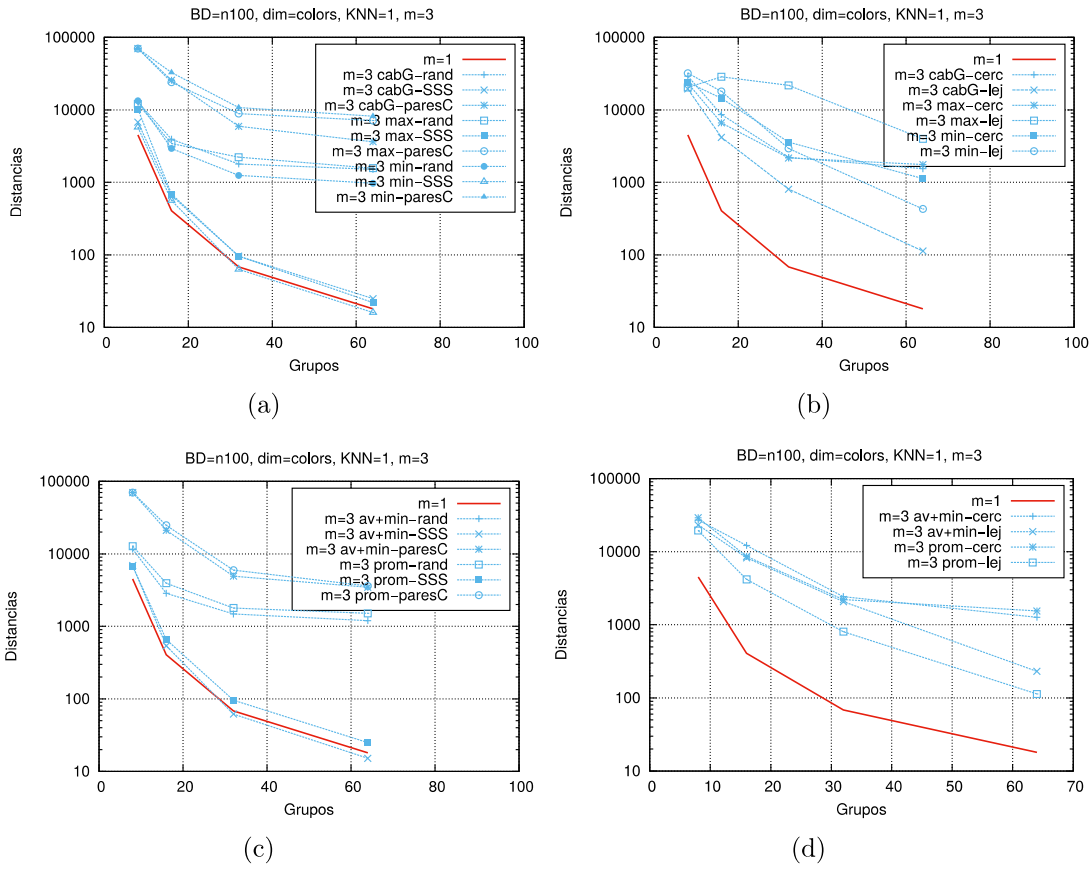


Figura A.14: Base de datos *Colors* con m =3, k ={8,16,32,64}

m	critDist	selectP		m	critDist	selectP
2	dist	SSS		3	cabG	SSS
2	max	SSS		3	prom	SSS
2	min	SSS		3	av+min	SSS
2	cabG	SSS		4	dist	SSS
2	prom	SSS		4	max	SSS
2	av+min	SSS		4	min	SSS
3	dist	SSS		4	cabG	SSS
3	max	SSS		4	prom	SSS
3	min	SSS		4	av+min	SSS

Tabla A.8: Se muestran los combinaciones con mejor rendimiento para la base de datos *NASA*

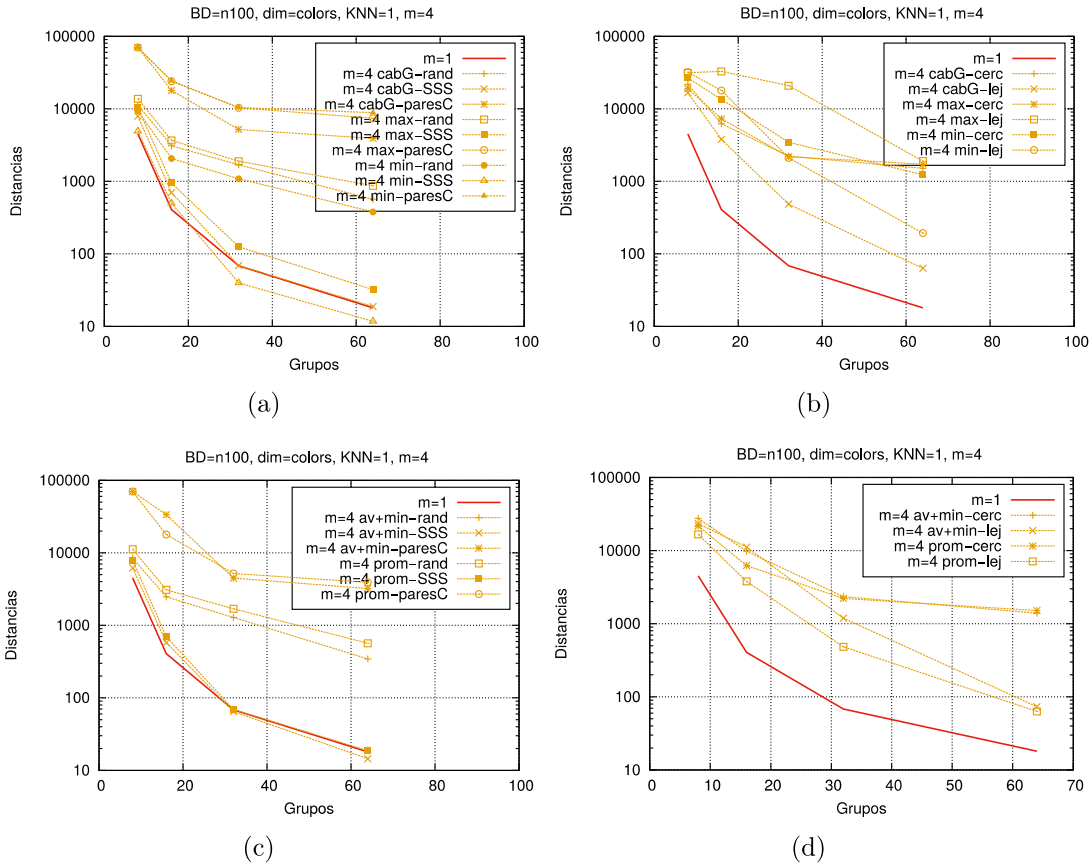


Figura A.15: Base de datos *Colors* con m =4, k ={8,16,32,64}

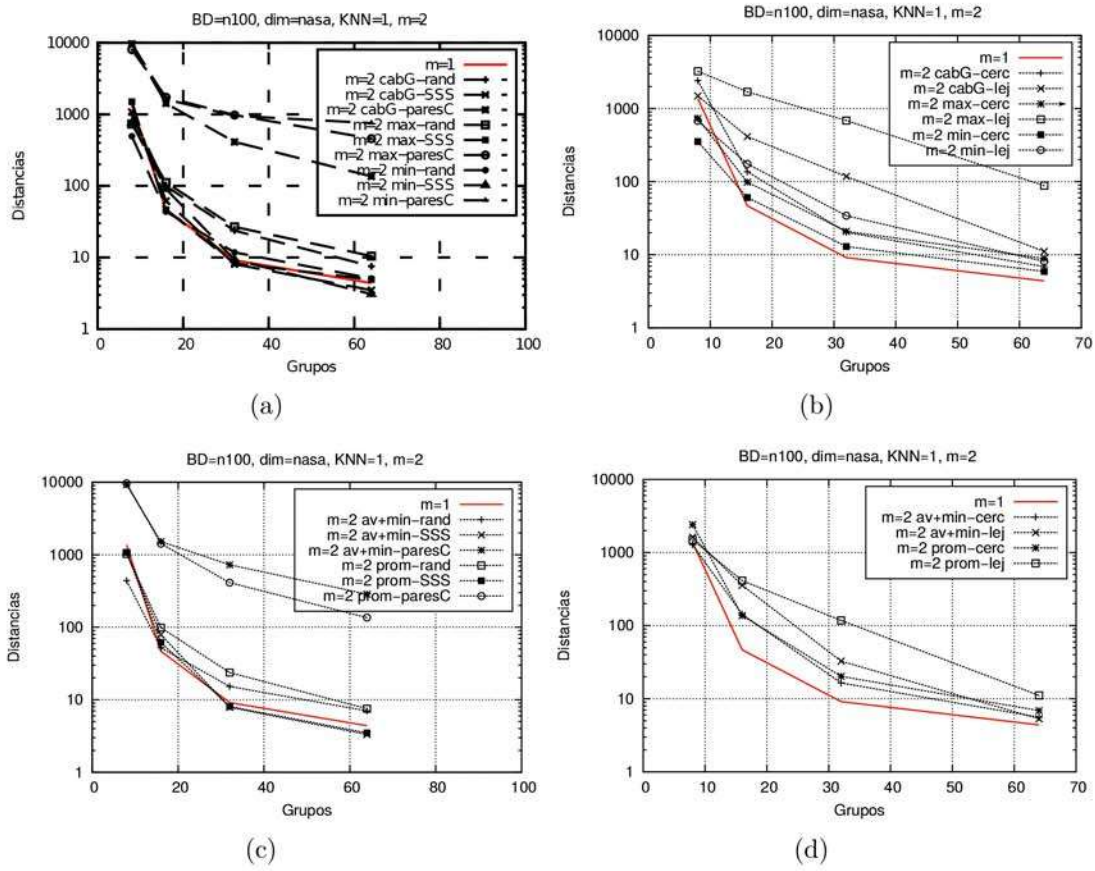


Figura A.16: Base de datos *NASA* con m =2, k ={8,16,32,64}

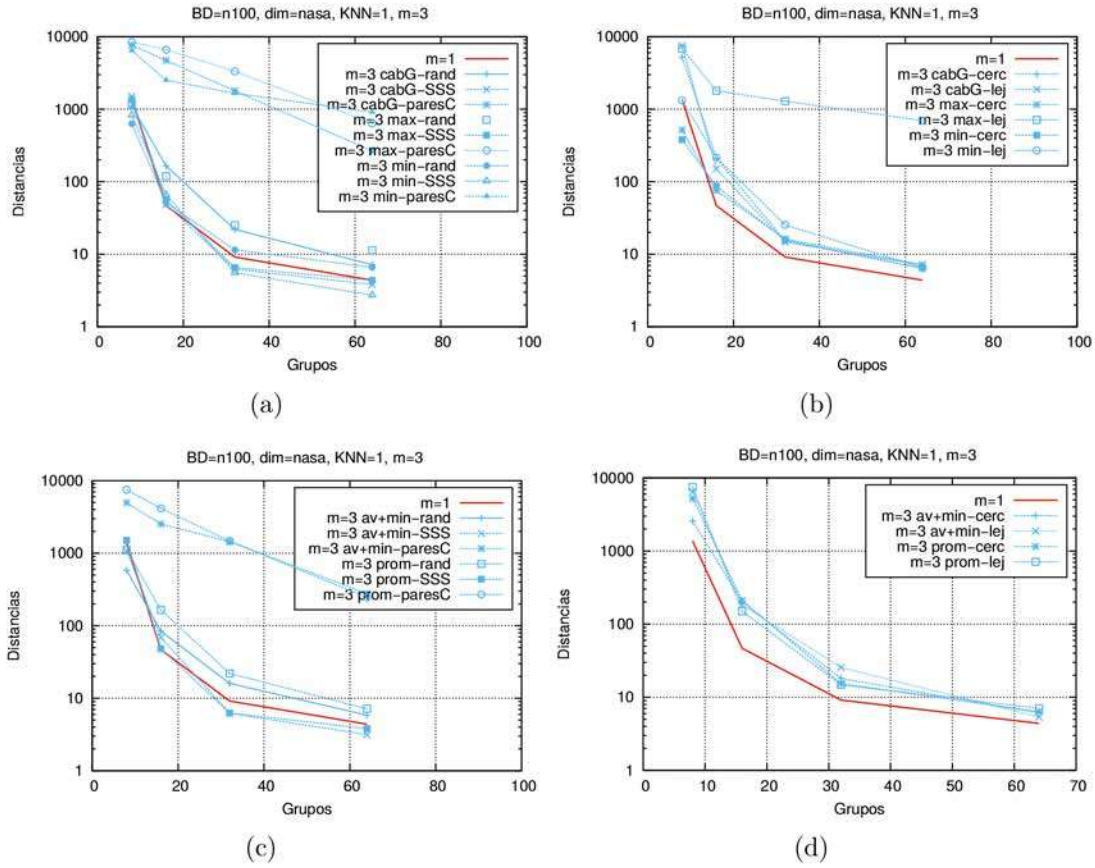


Figura A.17: Base de datos *NASA* con m =3, k ={8,16,32,64}

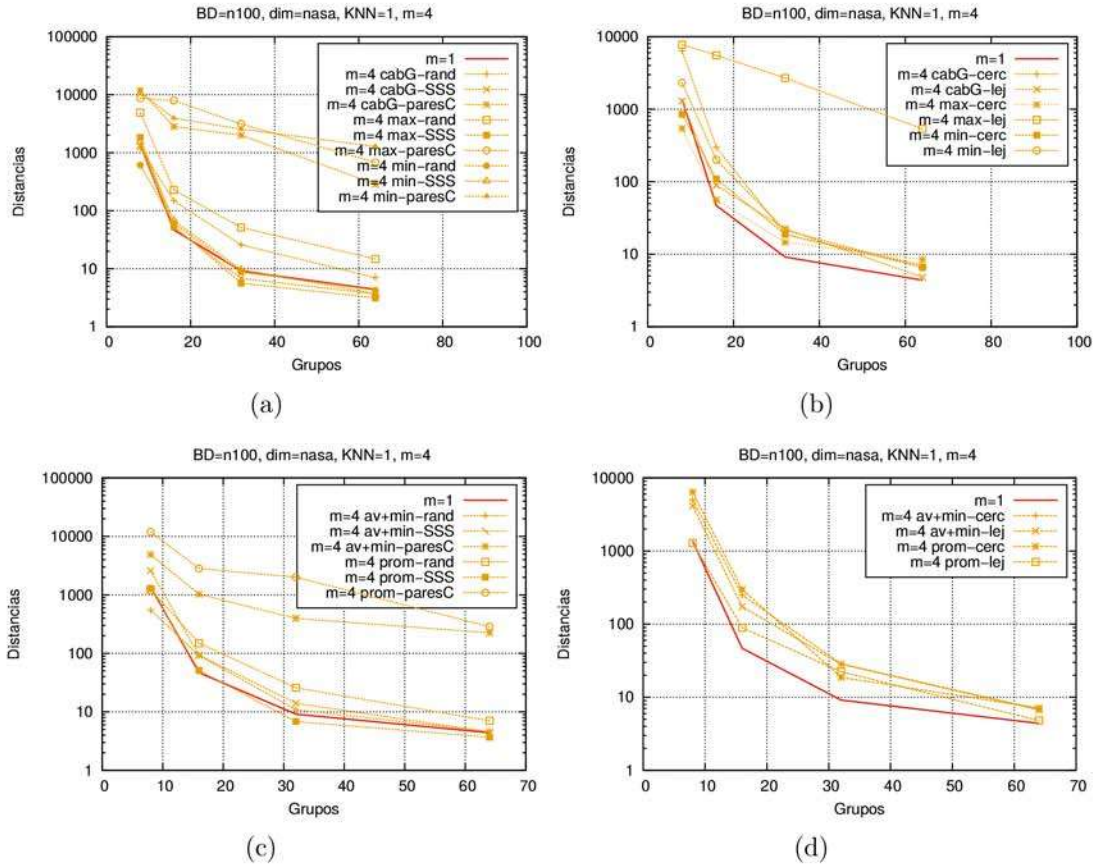
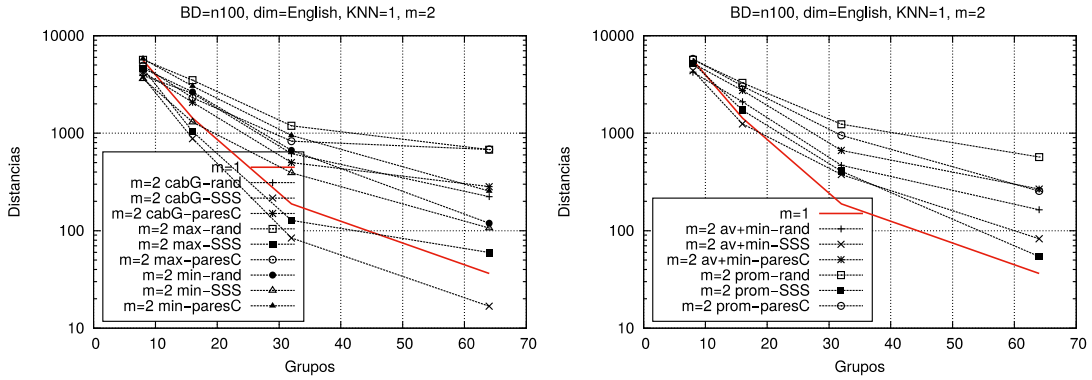


Figura A.18: Base de datos *NASA* con m =4, k ={8,16,32,64}

A.2.2. Base de Datos de *English* y *Spanish* variando m

En las siguientes Figuras A.19, A.20, A.21 yA.22, A.22, A.22 se agrupan las gráficas correspondientes de los distintos parámetros de las base de datos *English* y *Spanish* respectivamente. De estas gráficas podemos obtener las siguientes 2 tablas, A.9 y A.10 de *English* y *Spanish* respectivamente, que contienen los parametros con mejor comportamiento. Para la tabla A.9 de *English* podemos obtener que los mejores parámetros son:con m =2 *cabG-SSS*, *max-SSS*, *av+min-SSS*, con m =3 *cabG-SSS*, *max-SSS*, *av+min-SSS*, *prom-SSS*, con m =4, *cabG-SSS*, *max-SSS*, *prom-SSS*, *av+min-rand*.

Para la tabla A.10 de *Spanish* podemos obtener que los mejores parámetros son: con m =2 *cabG-SSS*, *av+min-SSS*, *prom-SSS*, con m =3 *cabG-SSS*, *min-SSS*, *av+min-SSS*, *prom-SSS*, con m=4 *cabG-SSS*, *min-SSS*, *av+min-SSS*, *prom-SSS*.

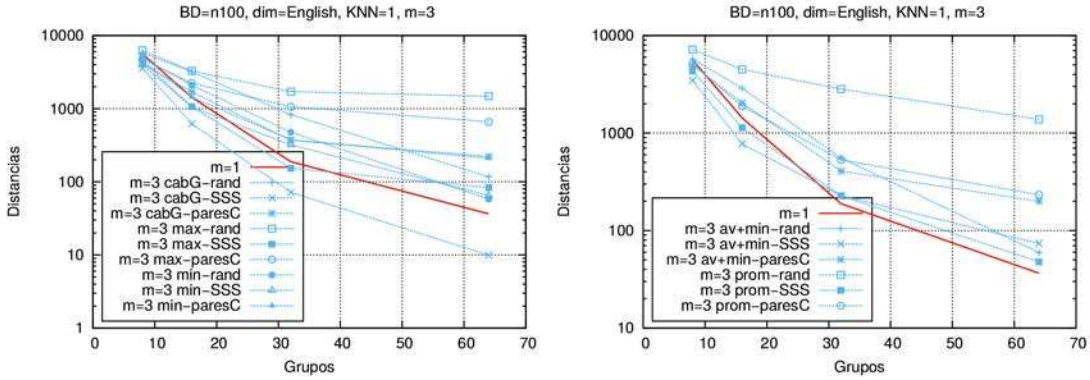


(a) (b)

Figura A.19: Base de datos *English* con m =2, k ={8,16,32,64}

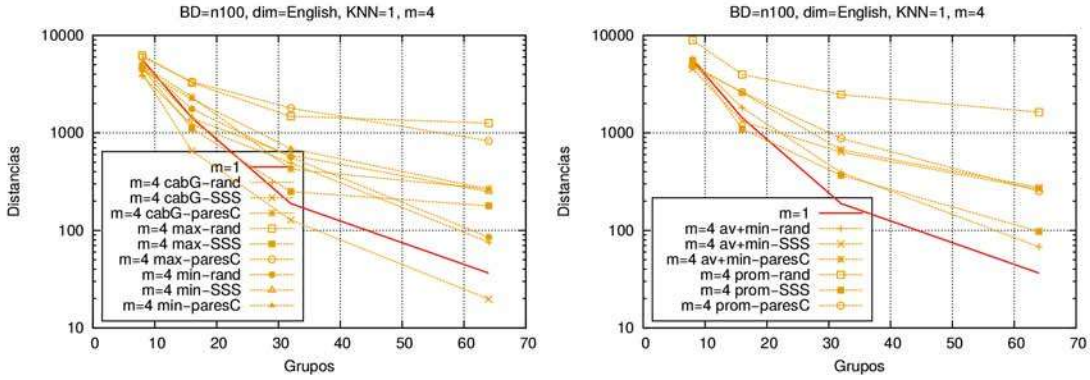
A.2.3. Base de Datos *Colors* y *NASA* variando KNN

En la siguiente Figura A.25 se agrupan las gráficas con un mejor rendimiento de la BD de *Colors*. En k ={8,16} el parámetro que muestra un comportamiento igual que el de la técnica original es con m =4 *min-SSS*. En k =32 los parámetros que muestran un comportamiento igual que al de la técnica original son: m =2 *av+min-SSS*, m =4 *av+min-SSS*, *prom-SSS*, los parámetros que muestran una mejoría contra la técnica



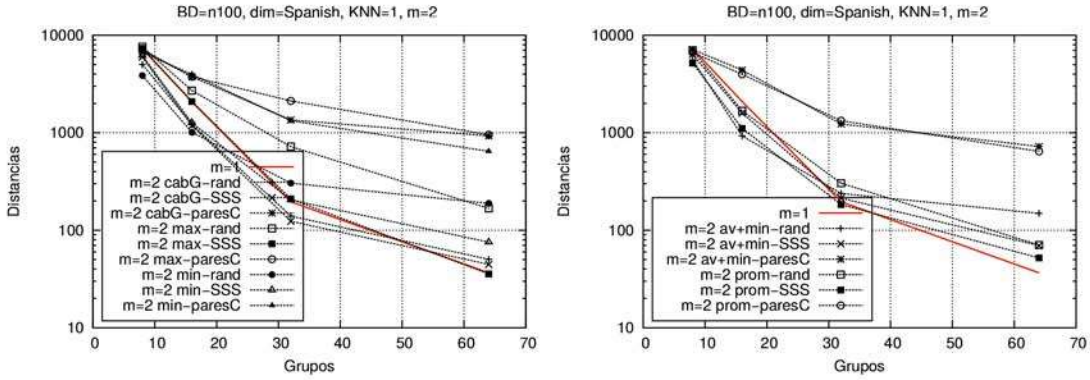
(a) (b)

Figura A.20: Base de datos *English* con m =3, k ={8,16,32,64}



(a) (b)

Figura A.21: Base de datos *English* con m =4, k ={8,16,32,64}

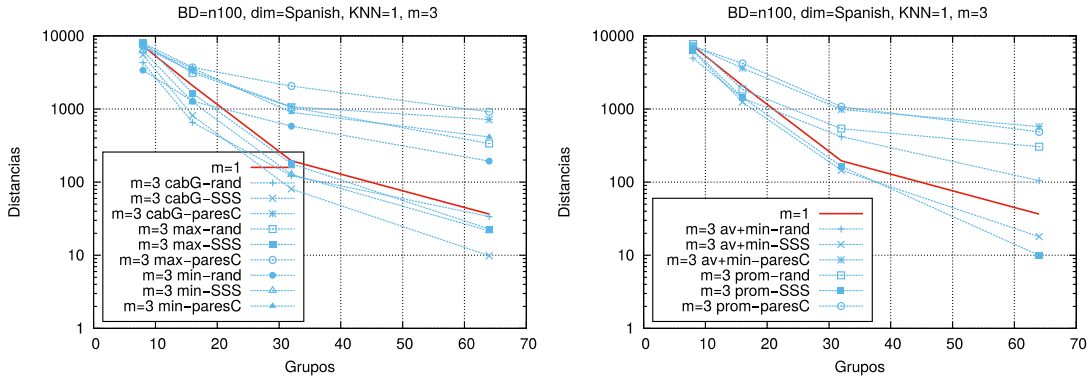


(a) (b)

Figura A.22: Base de datos *Spanish* con m =2, k ={8,16,32,64}

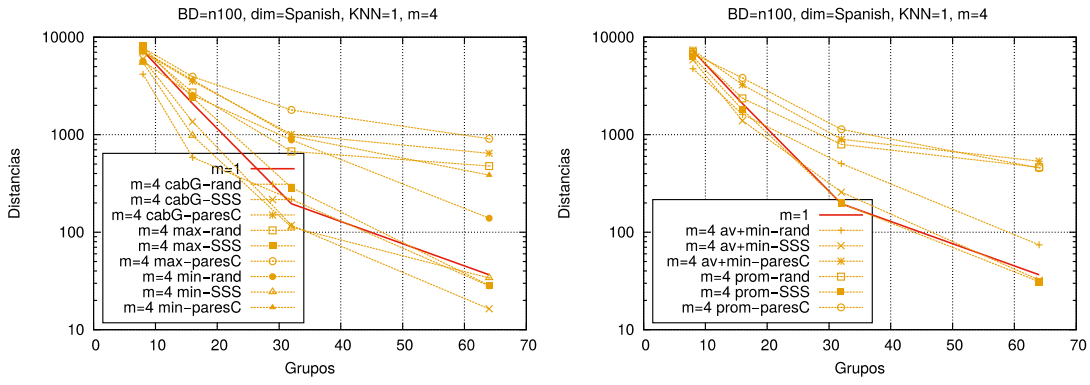
m	critDist	selectP		m	critDist	selectP
2	dist	SSS		3	cabG	SSS
2	dist	paresC		3	prom	SSS
2	max	SSS		3	prom	paresC
2	min	SSS		3	av+min	SSS
2	cabG	SSS		4	dist	SSS
2	cabG	paresC		4	dist	rand
2	prom	SSS		4	max	SSS
2	av+min	SSS		4	min	SSS
2	av+min	rand		4	min	paresC
3	dist	SSS		4	cabG	SSS
3	dist	paresC		4	prom	SSS
3	max	SSS		4	prom	paresC
3	min	SSS		4	av+min	SSS
3	min	paresC				

Tabla A.9: Se muestran los combinaciones con mejor rendimiento para la base de datos *English*



(a) (b)

Figura A.23: Base de datos *Spanish* con m =3, k ={8,16,32,64}



(a) (b)

Figura A.24: Base de datos *Spanish* con m =4, $k = \{8, 16, 32, 64\}$

original son: m =2 *min-SSS*, m =3 *av+min-SSS*, *min-SSS*, m =4 *av+min-SSS*, *min-SSS*. En k =64 la mayoría de los parámetros muestran una mejoria contra la técnica original, con excepción de: m =2 *max-SSS* y m =4 *prom-SSS*.

En la siguiente Figura A.26 se agrupan las gráficas con un mejor rendimiento de la BD de *NASA*. En k =8 los parámetros que muestran un comportamiento igual que el de la técnica original son: con m =2 *prom-SSS*, m =3 *max-SSS*, *min-SSS*, los parámetros que muestran un comportamiento mejor que el de la técnica original son: m =2 *min-SSS*, *av+min-SSS*, m =3 *av+min-SSS*. En k =16 los parámetros que muestran un comportamiento mejor que el de la tecnica original son: m =2 *min-SSS*, *av+min-SSS*, m =3 *min-SSS*, *av+min-SSS*. En k =32 los parámetros que muestran un comportamiento mejor que el de la tecnica original son: m =2 *min-SSS*, *av+min-SSS*, m =3 *min-SSS*, *av+min-SSS*. En k =64 los parámetros que muestran un comportamiento mejor que el de la tecnica original son: m =2 *min-SSS*, *av+min-SSS*, m =3 *min-SSS*, *av+min-SSS*.

A.2.4. Base de Datos *English* y *Spanish* variando KNN

En la siguiente Figura A.27 se agrupan las gráficas con un mejor rendimiento de la BD de *English*. En k={8,16, 32, 64} se muestra que los parámetros con una mejoría a comparación de la técnica original son: con m =2 *cabG-rand*, con m =3 *cabG-SSS*, *av+min-SSS*, con m =4 *cabG-SSS*.

m	critDist	selectP		m	critDist	selectP
2	dist	SSS		3	cabG	SSS
2	dist	rand		3	cabG	rand
2	min	SSS		3	prom	SSS
2	min	rand		3	av+min	rand
2	cabG	SSS		4	dist	SSS
2	cabG	rand		4	dist	rand
2	prom	SSS		4	min	SSS
2	av+min	rand		4	cabG	SSS
3	dist	SSS		4	cabG	rand
3	dist	rand		4	prom	SSS
3	max	SSS		4	av+min	SSS
3	min	SSS		4	av+min	rand
3	min	rand				

Tabla A.10: Se muestran los combinaciones con mejor rendimiento para la base de datos *Spanish*

En la siguiente Figura A.28 se agrupan las gráficas con un mejor rendimiento de la BD de *Spanish*. En $k = \{8, 16\}$ se tiene que los parámetros tuvieron una mejoría a comparación de la técnica. original. En $k = 32$ se tiene que los parámetros tuvieron una mejoría a comparación de la técnica son: con m =2 *cabG-rand*, *cabG-SSS*, con m =2 *cabG-SSS*.

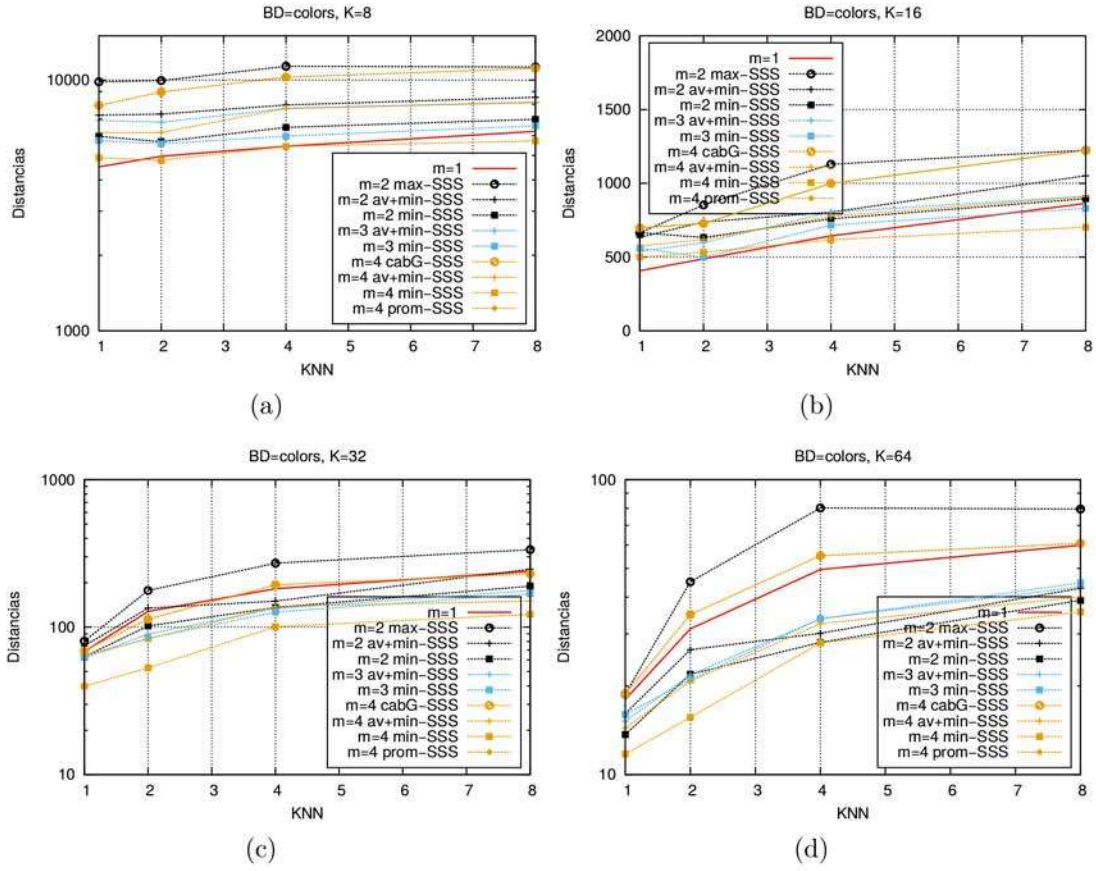


Figura A.25: Base de Datos *Colors* con KNN={1,2,4,8}

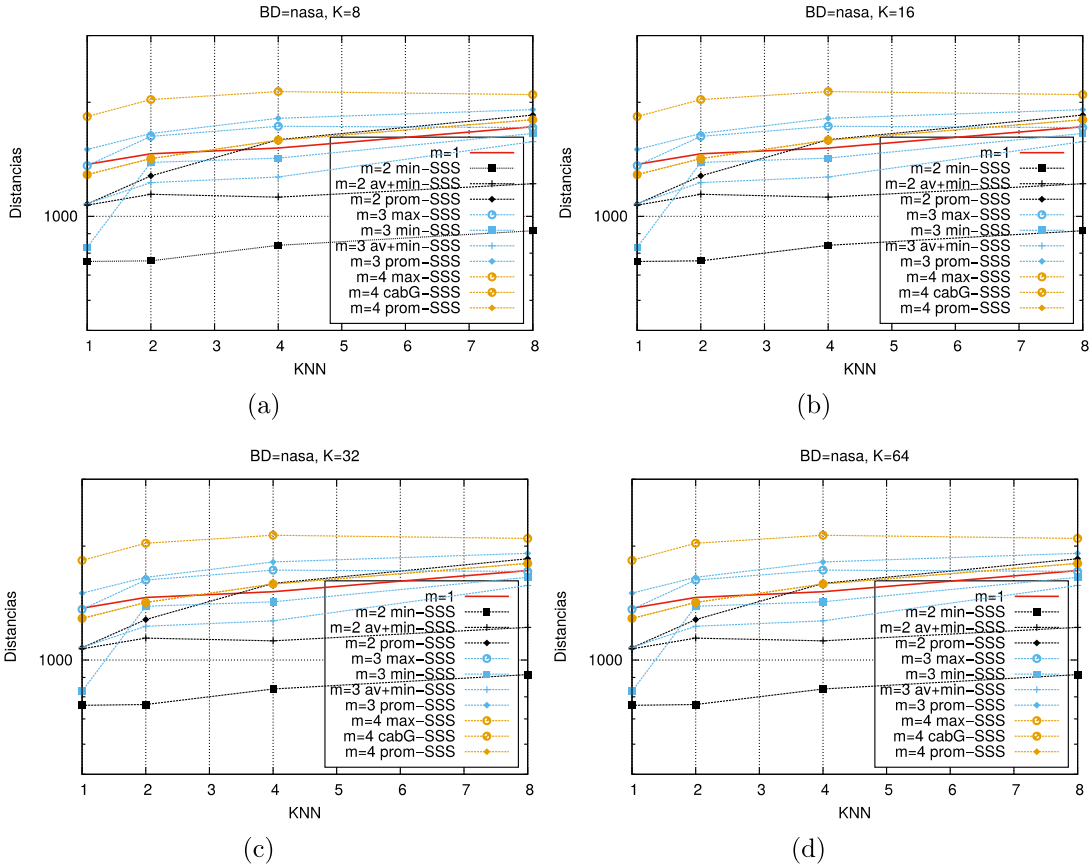


Figura A.26: Base de Datos *NASA* con KNN={1,2,4,8}

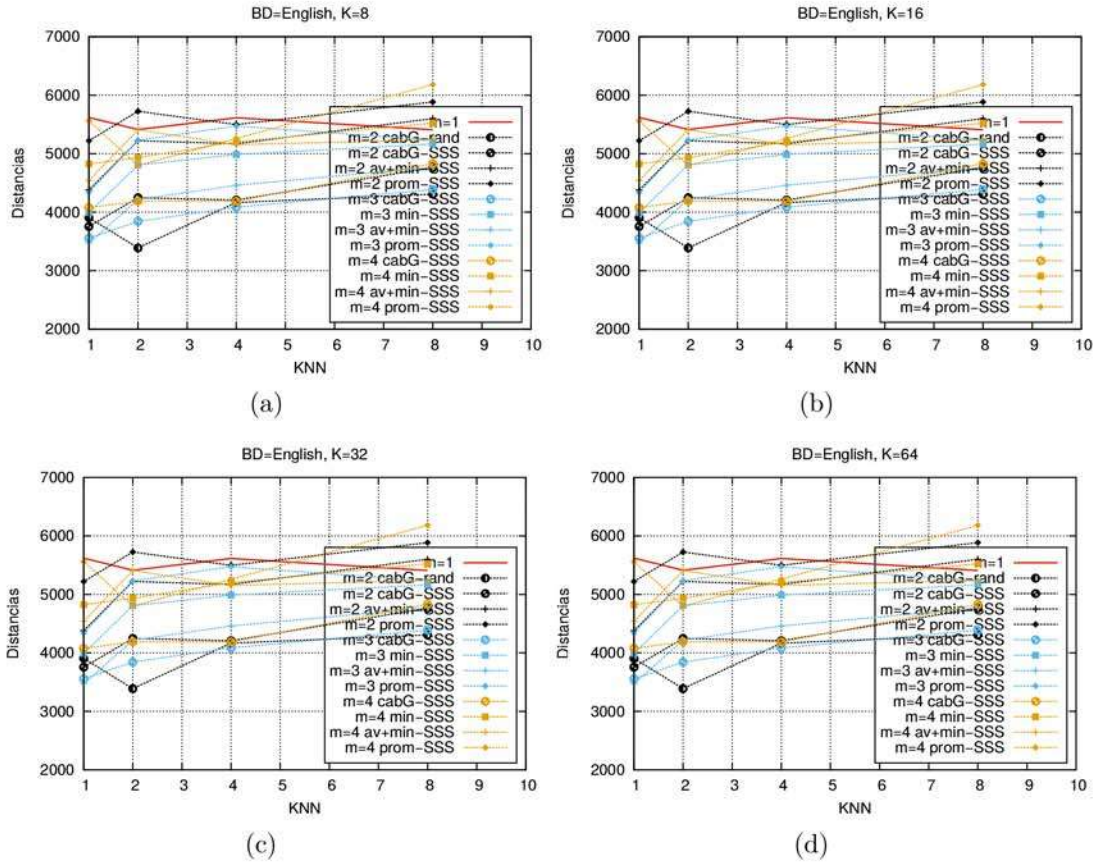


Figura A.27: Base de Datos *English* con KNN={1,2,4,8}

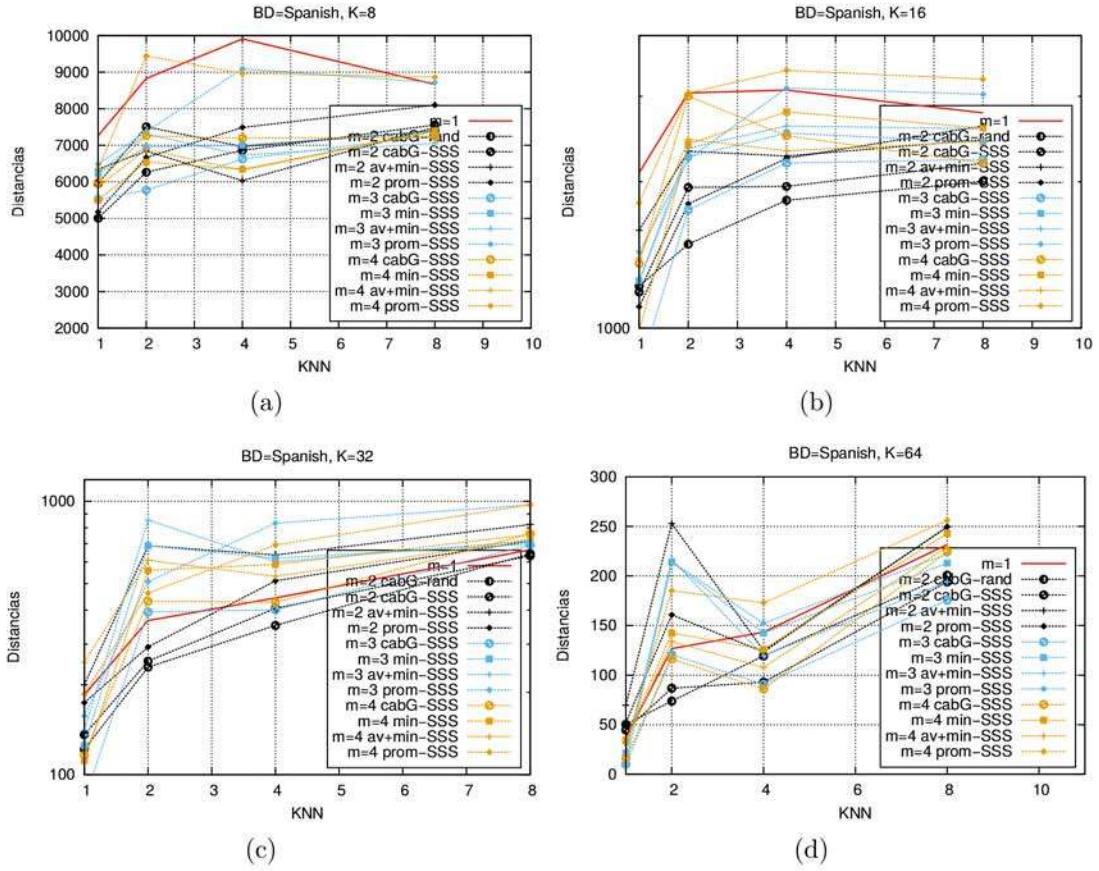


Figura A.28: Base de Datos *Spanish* con KNN={1,2,4,8}

Bibliografía

[1] J. Bentley. «Multidimensional binary search trees in database applications». En: *IEEE Transactions on Software Engineering* 5.4 (1979), págs. 333-340.

[2] J. Bentley. «Multidimensional binary search trees used for associative searching». En: *Communications of the ACM* 18.9 (1975), págs. 509-517.

[3] S. Berchtold, D. Keim y H. Kriegel. «The X-tree: an index Structure for High-Dimensional Data». En: *Proceedings 22nd Conference on Very Large Databases (VLDB'96)*. 1996, págs. 28-39.

[4] S. Berchtold y col. «On Optimizing Nearest Neighbor Queries in High-Dimensional Data Spaces». En: *8th International Conference Database Theory (ICDT)*. Vol. 1973. Springer, 2001, págs. 435-449.

[5] C. Böhm, S. Berchtold y D. A. Keim. «Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases». En: *ACM Computing Surveys* 33.3 (2001), págs. 322-373.

[6] Nieves R. Brisaboa, Antonio Farina y Oscar Pedreira. «Spatial Selection of Sparse Pivots for Similarity Search in Metric Spaces». En: *JCS* 7.1 (2007), págs. 8-13.

[7] E. Chavez y col. «Proximity searching in Metric Spaces». En: *ACM Computing Surveys* 33(3) (2001), págs. 273-321.

[8] E. Chavez y col. *Searching in Metric Spaces*. Inf. téc. TR/DCC-99-3. `ftp://-ftp.dcc.uchile.cl/pub/users/gnavarro/survmetric.ps.gz`. Dept. of Computer Science, Univ. of Chile, 1999.

[9] Edgar Chavez, Karina Figueroa y Gonzalo Navarro. «Proximity Searching in High Dimensional Spaces with a Proximity Preserving Order». En: *MICAI 2005: Advances in Artificial Intelligence*. Vol. 3789. Lecture Notes in Computer Science. 2005, págs. 405-414.

[10] K. Figueroa y R. Paredes. «An Effective Permutant Selection Heuristic for Proximity Searching in Metric Spaces». En: *Proc. 6th Mexican Conf. on Pattern Recognition (MCPR'14)*. LNCS 8495. Springer, 2014, págs. 102-111.

[11] K. Figueroa, R. Paredes y R. Rangel. «Efficient Group of Permutants for Proximity Searching». En: *Proc. 3rd Mexican Conference on Pattern Recognition (MCPR 2011)*. LNCS 6718. Springer, 2011, págs. 42-49.

[12] A. Guttman. «R-trees: a dynamic index structure for spatial searching». En: *Proceedings ACM SIGMOD International Conference on Management of Data*. 1984, págs. 47-57.

[13] G. Hjaltason y H. Samet. «Ranking in Spatial Databases». En: *Fourth International Symposium on Large Spatial Databases*. 1995, págs. 83-95.