GraspKM en la Recuperación de la Estructura de Software

Erick Vicente
Facultad de Ingeniería, Universidad Ricardo Palma
Lima 33, Perú
evicente@mail.urp.edu.pe

Manuel Tupia
Facultad de Ciencias e Ingeniería, Pontificia Universidad Católica del Perú
Lima 100, Perú
tupia.mf@pucp.edu.pe

Luis Rivera
Universidad Estadual Norte Fluminense, UENF, LCMAT-CCT
Campos dos Goytacazes, Rio de Janeiro, Brasil, 28015-620
rivera@uenf.br

Abstract

En la actualidad existe gran cantidad de sistemas de software carente de documentación, mas aún cuando se tratan de sistemas legados. En la literatura se han propuesto diversos métodos para obtener una abstracción de la estructura de estos sistemas. Estos métodos se encuentran basados principalmente en clustering, debido a los objetivos coincidentes de lo que se quiere de la estructura de un sistema y de la estructura de los clusters: los módulos de software deben ser altamente cohesivos y con bajo acoplamiento, de manera similar un cluster debe contener elementos que sean similares entre sí y que sean a su vez lo mas diferentes entre clusters. Los métodos encontrados en literatura para el clustering de software se encuentran clasificados dentro del clustering jerárquico. En el presente trabajo proponemos la adaptación del método KMeans en el contexto de GRASP, denominado GraspKM, para la búsqueda de la estructura de un sistema. Este método trata el clustering como un problema de optimización combinatoria y demuestra ser eficiente optimizando la función objetivo propuesta.

1. Introducción

En Ingeniería de Software, en las líneas de investigación de reutilización e ingeniería reversa, los componentes de software, como: programas, rutinas, procedimientos, etc. deben ser estructurados y reutilizados por diferentes sistemas. En esa perspectiva, los mecanismos de agrupamiento, tipo clustering, han jugado un papel importante en el desarrollo de técnicas para el particionamiento, la recuperación y reestructuración de software [8].

La recuperación es uno de principales problemas que se presenta en la Ingeniería de Software, el objetivo es obtener un modelo del sistema que permita lograr un mejor entendimiento de como este se encuentra estructurado. La búsqueda de estos grupos se realiza de tal manera que se satisfaga los criterios de cohesión y acoplamiento. Es decir, los grupos a encontrar deben tener un alto grado de cohesión, y bajo acoplamiento con respecto a otros grupos.

Estos objetivos concuerdan con los del clustering, donde se busca obtener grupos que sean lo mas similares entre si, y a la vez diferente de otros clusters. Witggerts [18] les denomina entidades a los objetos a agrupar y estos pueden ser programas, funciones o procedimientos.

Existen dos formas de representación del software para recuperar su estructura. La primera es a través del uso de grafos, donde los nodos representan las entidades, y los vertices las relaciones que existen entre ellos. El objetivo es encontrar grupos que contengan nodos altamente relacionados (alta cohesión) y que tengan menos relaciones posibles entre ellos (bajo acoplamiento), estos grupos serán los módulos o subsistemas del software en análisis. Debido a que este

es un problema NP-Difícil, se han propuesto diversos métodos heurísticos y metaheurísticos para encontrar un solución eficiente del problema [12, 4].

La segunda forma, y aquí es donde encaja el presente trabajo, es a través de vectores que contenga las características que presenta la entidad de software. Estas pueden ser referencias a variables, llamadas a otros módulos o tipos de accesos. La representación en este caso es mediante vectores binarios, en donde 1 indica la presencia de la característica y 0 su ausencia. Mediante técnicas de clustering se buscan encontrar grupos que contengan entidades que sean lo mas similares entre si (alta cohesión) y lo mas diferentes de otros grupos (bajo acoplamiento). Cuando se trata de sistemas legados, que fueron elaborados sin previa documentación de diseño, obtener esta abstracción de alto nivel es realmente importante porque ayudará a los ingenieros de software a tener un mejor entendimiento de la arquitectura del sistema cuando se necesiten realizar modificaciones al sistema. Este problema es también NP-Difícil, y en la literatura podemos encontrar diversos métodos de clustering, clasificados como jerárquicos, para encontrar una solución al problema [18, 13, 15, 10]. Estos métodos tienen la desventaja de que implican un alto costo computacional en la búsqueda de soluciones [7]. Es aquí, donde se hace necesario la propuesta de un método de clustering de optimización basado en centros. Donde, los centros son aquellos elementos que mejor representan al cluster, y por ello también le denominaremos elemento representativo.

El problema del clustering consiste en encontrar grupos de objetos que sean similares entre sí y a la vez diferentes a los objetos pertenecientes a otros grupos, de tal manera que se satisfaga un criterio establecido. Los grupos con características similares son conocidos como *clusters*. Los objetos son representados por vectores en el espacio R^D, y con el uso de una medida de la similaridad, como la distancia, se definen los clusters. No existe conocimiento previo acerca de cómo se debe definir un cluster; por tal motivo, el proceso de clustering es conocido como clasificación no supervisada.

El clustering tiene múltiples aplicaciones dentro de las ciencias de la computación, como compresión de imágenes [16] y voz digitalizadas [9]; en la recuperación de información [1]; en minería de datos [5]; en la segmentación de imágenes médicas [14], en la clasificación de componentes de software [8], y otros.

En este trabajo proponemos un mecanismo de agrupamiento de entidades de software, estableciendo una cuantificación de los atributos de los componentes de forma que puedan ser usado por técnicas de clustering basado en KMeans en el marco de la metaheurística GRASP, técnica puramente numérica. Ese mecanismo va a permitir seleccionar elementos representativos de cada grupo de un conjunto grande de componentes de software.

Resto del trabajo está organizado de la siguiente manera: en la Sección 2 se hace una presentación de Clustering y revisión de los trabajos recientes, así como la presentación de la metaheurística GRASP. En la Sección 3 se presenta la adaptación del método Grasp-KM para el clustering de software. En la Sección 4 se muestran los experimentos computacionales realizados. Finalmente, en la Sección 5 se exponen las conclusiones y trabajos futuros.

2. Antecedentes

En esta sección se hace una revisión de los principales conceptos para la aplicación de las técnicas de clustering en la ingeniería de software. Primero se define el problema del clustering y se revisan las medidas de similaridad empleadas para el clustering de software. Luego, se hará una revisión los trabajos existentes en la literatura. Finalmente, se hace una revisión de la metaheurística GRASP.

2.1. Problema de Clustering

Dado un conjunto de n objetos denotado por $X = \{x_1, x_2, \ldots, x_n\}$, en que $x_i \in R^D$. Sea K un número entero positivo, el problema del clustering consiste en encontrar una partición $P = \{C_1, C_2, \ldots, C_K\}$ de X, siendo C_j un cluster definido por objetos similares, satisfaciendo una función objetivo $f : R^D \rightarrow R$ que representa la distancia mínima entre los objetos del cluster, y las condiciones:

$$C_i \bigcap C_j = \emptyset \text{ para } i \neq j, \quad \text{y} \quad \bigcup C_i = X.$$

La definición de la función de similaridad depende de la naturaleza de los objetos a agrupar.

2.2. Medidas de similaridad

Diversas medidas han sido propuestas para medir la similaridad entre objetos. Wiggerts[18], clasifica las medidas de la similaridad en cuatro tipos: medidas de la distancia, coeficientes de asociación, coeficientes de correlación y medidas probabilísticas. Son destacados los dos primeros tipos de medidas. Los coeficientes de asociación se usan principalmente para medir la similaridad entre dos vectores binarios, obteniendo valores en el intervalo $[0, 1]$; cuanto más cercano a 1, indica que los vectores son similares entre sí. Por otro lado, las medidas de la distancia obtienen como resultado un

valor real positivo entre dos entidades, donde un valor cercano a 0 indicará que los objetos son similares.

Medidas de la distancia

La noción de similaridad entre dos entidades representadas por dos vectores x y $y \in R^D$, es caracterizada por una función distancia como

$$d : (x, y) \rightarrow R.$$

Se dice que dos objetos $x = (x_1, ..., x_D)$ y $y = (y_1, ..., y_D)$ son similares cuando la distancia entre los vectores es menor que una tolerancia pequeña. En [2] se describen las propiedades que debe tener la distancia.

La elección de una función distancia entre los vectores depende del grado de dificultad de las entidades y su interpretación. La más usada es la distancia Euclidiana, definida por:

$$d_2(x, y) = \sqrt{\sum_{h=1}^{D} (x_h - y_h)^2}.$$

La distancia Euclidiana es un caso especial de la medida de Minkowski cuando $p = 2$, dada por

$$d_p(x, y) = \sqrt[p]{\sum_{h=1}^{D} (x_h - y_h)^p}.$$

Coeficientes de asociación

Los coeficientes de asociación, o de comparación, para dos entidades $E1$ y $E2$ representadas por vectores binarios x y y, respectivamente, están dadas por el número de atributos coincidentes de una entidad en relación a otra. Lung et al. [8] clasifica a este tipo de coeficientes como cualitativos, debido a que calculan la similaridad basado en la ausencia o presencia de atributos. Según Wiggerts [18], son cuatro casos de asociación entre las entidades respecto al número de sus atributos: presentes en ambas entidades (a); presentes en $E1$ pero no en $E2$ (b); presentes en $E2$ pero no en $E1$ (c); y no presentes en ambos (d). Si se denota por 1 binario como presencia de un atributo en una entidad, y por 0 la ausencia, es apropiado relacionar la ocurrencia de esos atributos por una tabla definida como:

		$E2$	
		1	0
$E1$	1	a	b
	0	c	d

Por ejemplo, sean dos entidades $E1$ y $E2$, descritas através de dos vectores binarios $x = (0, 1, 0, 1, 1, 1)$ y $y = (0, 1, 1, 0, 1, 0)$, respectivamente. Entonces, $a = 2$ porque los atributos presentes $(1 - 1)$ están en la segunda y quinta posición de ambos vectores. El valor de $b = 2$ porque los atributos cuarto y sexto están en x pero no en y, caso $(1 - 0)$. Así, se observan que $c = 1$, para $(0 - 1)$ y $d = 1$ para $(0 - 0)$.

Existen diversos métodos para calcular los coeficientes de asociación; ellos se basan, principalmente, en la relevancia de las coincidencias entre ambos vectores y la ponderación que le asignan. Los principales métodos para el cálculo de coeficientes entre dos vetores x y y, usados en [15, 18, 8], son:

- Coeficiente de **Jaccard**: $S_j(x, y) = a/(a + b + c)$
- Coeficiente **Simple**: $S_s(x, y) = (a+d)/(a+b+c+d)$
- Coeficiente de **Sorensen**: $S_r(x, y) = 2a/(2a+b+c)$

Se observa que el coeficiente de Jaccard y Sorensen considera relevantes las relación $1 - 1$, pero no las relación $0 - 0$ ya que estas indican la ausencia de atributos. El coeficiente Simple, considera relevantes tanto las relaciones $1 - 1$, como las $0 - 0$.

En [13], se propone una medida de similaridad en función de producto y norma de vectores binarios x y y, la cual también es usada para calcular la similaridad entre documentos por métodos de recuperación de información. La medida esta dada por la siguiente expresión:

$$S_1(x, y) = \frac{x \times y}{\|x\|\|y\|}.$$

En el mismo trabajo, se extiende esta función de medida a vectores no binarios, donde los valores expresan la frecuencia de ocurrencia de cierta característica. Por ejemplo, si $x = (x_1, \ldots, x_n)$ y $y = (y_1, \ldots, y_n)$ representan a dos entidades de software (i.e. programas), entonces los valores x_i y y_i pueden expresar la cantidad de veces que datos tipo T_i son declarados en dichos programas. La función de medida extendida envuelve producto interno de vectores,

$$S_2(x, y) = \frac{x \cdot y}{\|x\|\|y\|}.$$

2.3. Software Clustering

En [15] se presenta un método para agrupar entidades de software representadas por vectores binarios. Una de las características relevantes del método es, que a partir de los vectores que conforman un cluster se obtiene un vector representativo. Este vector se obtiene aplicando el operador OR a los vectores binarios que conforman el cluster. Por ejemplo,

si los vectores $x_1 = (0,1,1)$ y $x_2 = (0,0,1)$ conforman el cluster C entonces, el vector representativo del cluster será $x_C = (0,1,1)$. Los clusters se van construyendo a través del método jerárquico *Weighted Average Linkage*, y la medida de similaridad usada es el coeficiente de asociación Jaccard. El método propuesto tiene el inconveniente de que al aplicar el operador OR para obtener el vector representativo del cluster, se ignora la cantidad de entidades que presentan la misma característica. Por ejemplo, si se tienen los clusters $A = \{(0,1,0),(0,1,0),(1,1,0)\}$ y $B = \{(0,0,1),(0,1,1)\}$, entonces los vectores representativos de A y B son $x_A = (1,1,0)$ y $x_B = (0,1,1)$, respectivamente.

Al calcular el coeficiente de asociación de un vector $x_6 = (0,1,0)$ a los clusters A o B se obtendría el mismo valor, y se podría asignar a cualquiera de ellos. Sin embargo se observa que el cluster A, tiene mas vectores con el valor $x_{i2} = 1$, y por tanto lo lógico sería que se asocie al cluster A.

En [10] se presenta una mejora al método propuesto en [15]. El cálculo del vector representativo se basa en la frecuencia con que se presentan las características en las entidades que componen el cluster. Es decir para el caso anterior: el vector representativo del cluster A, será $x_A = (1/3, 3/3, 0/3)$ y para el cluster B, será $x_B = (0/2, 1/2, 2/2)$. Esto quiere decir que el elemento representativo de un cluster $C = \{(x_{11},\ldots,x_{1d}),\ldots,(x_{n1},\ldots,x_{nd})\}$ estará dado por

$$x_C = \left(\frac{\sum_{i=1}^{n} x_{i1}}{n},\ldots,\frac{\sum_{i=1}^{n} x_{id}}{n}\right). \qquad (1)$$

El vector representativo deja de ser binario y por tanto ya no se puede aplicar los coeficientes de asociación revisados en la punto antrerior. El autor extiende la medida de similaridad denominada *Ellenberg* para tomar en cuenta las frecuencias de las características. A la medida de la similaridad propuesta le denomina *unbiased Ellenberg*, y esta dada por la siguiente expresión:

$$S_e(x,y) = \frac{(0{,}5)M_a}{(0{,}5)M_a + b + c} \qquad (2)$$

Donde, M_a representa la suma de las características que están presentes en ambas entidades, y b y c representan la cantidad de características que están presentes en una entidad y no en la otra. Con esta medida de la similaridad, en [10] se utilizan métodos jerárquicos de clustering mejorando los resultados obtenidos en [15].

2.4. Metaheurística GRASP

Un procedimiento de búsqueda voraz aleatoria y adaptativa (GRASP) es una metaheurística propuesta por Feo y Resende [6] para encontrar soluciones aproximadas de problemas de optimización combinatoria, mediante un proceso iterativo. En cada iteración se realizan los procesos de *construcción* y *búsqueda local*. En la construcción se genera un conjunto solución S de una instancia E de un problema combinatorio, y en la búsqueda local se determina una posible mejor solución a S; finalmente, se elige la mejor solución entre la solución de la iteración anterior y la actual. La mejor solución será evaluada por una función objetivo f. Todo el proceso es repetido un número máximo de veces (MAX_ITER). El Algoritmo 1 muestra el marco general de la meteheurística GRASP.

Algoritmo 1: Grasp

entrada: E, MAX_ITER, α

```
1  inicio
2      Inicializar solución S := ∅ y f* := ∞ ;
3      para i = 1 hasta MAX_ITER hacer
4          S* := Construccion_Grasp(E, α) ;
5          S* := Busqueda_Local_Grasp(S*) ;
6          si f(S*) < f* entonces
7              Actualizar S := S* y f* := f(S*) ;
8          fin
9      fin
10     retornar S
11 fin
```

El hecho de que la búsqueda local toma como entrada la solución obtenida en la construcción proporciona un conocimiento frente a los algoritmos de búsqueda local tradicionales.

Construcción GRASP

En esta fase se adapta un algoritmo goloso que selecciona el mejor elemento de un conjunto de candidatos C ha ser incorporados en la solución. El criterio de selección goloso depende del problema, puede ser de maximización o minimización. El constructor de soluciones evita el determinismo de los algoritmos golosos, utilizando un parámetro de relajación $\alpha \in [0,1]$ para formar una lista restringida de candidatos (*Restricted Candidate List* - RCL) alrededor del mejor elemento a seleccionar. El elemento ha ser incorporado en la solución es elegido aleatoriamente del RCL. Esta forma estocástica de selección, permite construir soluciones con tendencia a los mejores elementos y evita caer en

óptimos locales. El parámetro de relajación α indica la amplitud del RCL alrededor del mejor candidato. El mejor valor de α para el problema en estudio se obtiene a través de múltiples experimentos computacionales de calibración.

Búsqueda Local GRASP

La búsqueda local se realiza de manera iterativa, explorando en la vecindad de un conjunto de solución S, generada en la construcción. El desempeño de la operación de búsqueda local dependerá del método elegido. Si N es una vecindad de soluciones, se dice que $S' \in N(S)$ es un óptimo local si $f(S') < f(S)$. No existe un esquema de búsqueda local específico a utilizarse, solo es necesario que mejore la solución encontrada en la construcción.

Algoritmo 2: GraspKM

entrada: X, K, α, MAX_ITER

```
1  inicio
2      f* := ∞, C := {} ;
3      para i = 1 hasta MAX_ITER hacer
4          C' := InicializacionKM(X, K) ;
5          C' := ConstruccionKM(X, K, C', α) ;
6          C' := MejoriaKM(X, K, C') ;
7          si f(C') < f* entonces
8              C:= C' ;
9              f*:= f(C') ;
10         fin
11     fin
12     retornar C
13 fin
```

3. Software Clustering usando GRASP

En [17] se propone el método GraspKM para encontrar una solución viable al problema del hard clustering, es decir, cuando los grupos encontrados son particiones del conjunto de objetos en análisis. El método GraspKM es una adaptación del algoritmo KMeans [11] dentro del marco de la metaheurística GRASP.

El algoritmo KMeans construye los clusters iterativamente, partiendo de K posibles centros seleccionados aleatoriamente de un conjunto X de N elementos. Los clusters se definen asignando cada elemento de X de forma que la distancia respecto al posible centro sea mínima respecto a otros centros. En cada iteración siguiente, se recalculan los centros de los clusters como la media aritmética de los elementos que lo componen; y si las variaciones de los centros aun persisten, entonces se vuelve a iterar hasta que los centros no varíen.

El GraspKM, mostrado en el Algoritmo 2, inicia de manera similar al KMeans. Se eligen aleatoriamente K objetos como centros y se forman los clusters iniciales asignando cada uno de los objetos al cluster cuyo centro se encuentre más cercano. Luego, se asignan iterativamente cada uno de los objetos a un cluster elegido aleatoriamente de una RCL. Este proceso se repite hasta que no haya mas reasignaciones. Finalmente, se obtiene una mejor solución a través de un algoritmo de búsqueda local.

El método utiliza como medida de similaridad la distancia euclidiana y demuestra ser superior al algoritmo K-Means y comparable con otras metaheurísticas. Aunque este método es eficiente encontrando soluciones para objetos representados por vectores en R^d, este no puede ser aplicado directamente al clustering de software y debe ser adaptado para cubrir los siguientes puntos:

- Las medidas de la similaridad en el clustering de software no están basadas principalmente en la distancia, sino en los coeficientes de asociación debido a que se trabaja con vectores binarios.
- Los coeficientes de asociación permiten el calculo de la similaridad entre dos vectores y entre grupo de vectores, pero la obtención de un vector centro que represente al grupo, no es de manera natural, como si lo permite la distancia euclidiana.

En esta perspectiva, se debe adaptar el método propuesto en [17] para agrupar vectores binarios que representan a las entidades de software. Como se describió en la sección anterior, en [10] se propone una medida de la similaridad llamada *unbiased Ellenberg* (2), que toma en cuenta la frecuencia con que se presentan las características en los componentes de software. Esta medida obtiene un valor entre $[0, 1]$, donde una valor de 1 o cercano indica que son similares. Para poder formular la función objetivo respecto a la minimización, se requiere tener una medida que cuando sea similar se acerque a cero. Por tanto, la medida de la similaridad que usaremos es:

$$S_m(x, y) = 1 - \frac{(0{,}5)M_a}{(0{,}5)M_a + b + c}. \qquad (3)$$

Esta medida permitirá calcular la similaridad entre el elemento representativo del cluster y un vector binario. En el referido trabajo [10], se propone uso de un elemento representativo para el clustering de software, aunque este no es un método de clustering basado en

centros como el algoritmo KMeans, es posible usar este elemento como centro, incluso estos coinciden con los centros usados en el algoritmo KMeans. Por tanto, los elementos representativos que usaremos para el algoritmo GraspKM estarán dados por la expresión (1).

Como lo que se quiere es obtener clusters con entidades de software que sean lo mas parecidas entre si. Entonces podemos definir empíricamente la similaridad de un cluster C como:

$$S(C) = \sum_{x \in C} S_m(x, \bar{x}_c). \quad (4)$$

En base a esta expresión podemos formular la función objetivo f como:

$$f = \sum_{j=1}^{K} S_m(C_j), \quad (5)$$

donde K es el número de clusters que deseamos definir y el objetivo del método a desarrollar debe ser minimizar esta función.

En la fase InicializacionKM, se eligen aleatoriamente K vectores de X como centros, y conforma los clusters iniciales asociando cada uno de los elementos de X al cluster mas cercano. Luego, se recalculan nuevamente los elementos representativos.

Como los elementos representativos han variado, los objetos deben ser asignados nuevamente a los clusters más cercanos. Esto se hace a través del proceso iterativo denominado ConstruccionKM, tal como es mostrado en el Algortimo 3, donde los posibles clusters que contendrían al objeto x en análisis, son agrupados en un conjunto RCL cuyas medidas de similaridad entre el cluster y el objeto x están en un intervalo definido por $\overline{\beta}$ y $\underline{\beta}$ y regulada linealmente por el parámetro de relajación α. Del conjunto RCL será elegido aleatoriamente un cluster al cual será reasignado el objeto x, y retirándolo del cluster donde se encontraba previamente. Después de la reasignación de todos los objetos de X los elementos representativos han variado; por tanto, nuevamente deben ser recalculados. El proceso termina cuando los elementos representativos no varían.

Las soluciones obtenidas en la fase de construcción son refinadas en la fase denominada MejoriaKM, que es un proceso iterativo de dos fases: ReagrupacionKM y ConstruccionKM, que se repiten hasta que la solución no pueda ser mejorada. La idea de la reagrupación es eliminar y generar nuevos cluster heurísticamente. Se elimina y se genera un nuevo clusters a la vez; el cluster a eliminar es aquel que presenta la menor cantidad de objetos, y se divide aquel cluster que tiene la mayor dispersión, esto debido a que en ambos casos el proceso puede haber caído en un óptimo local. Si C_l es el cluster con menor número de elementos, entonces l esta dado por:

$$l = ArgMin\{|C_j|\}_{j=1,...,K}. \quad (6)$$

El calculo del cluster C_h con mayor dispersión esta determinado por:

$$h = ArgMax\left\{\frac{Sim(C_j)}{|C_j|}\right\}_{j=1,...,K, j \neq l}. \quad (7)$$

Donde, $Sim(C_j)$ es la similaridad del cluster C_j y es calculada usando la expresión (4). Luego de que es eliminado el cluster y generado uno nuevo, cada uno de los objetos de X son asignados a los nuevos centros. Finalmente, los clusters obtenidos son refinados con el proceso ConstruccionKM.

Algoritmo 3: ConstruccionKM

entrada: X, K, C, α

```
inicio
    repetir
        para cada  x ∈ X  tal que x ∈ C_{j=1,...,K}
        hacer
            β̄ := Max{S_m(x, x̄_l) :
            S_m(x, x̄_l) ≤ S_m(x, x̄_j)}_{l=1,...,K} ;
            β̲ := Min{S_m(x, x̄_l)}_{l=1,...,K} ;
            RCL := {C_t :
            S_m(x, x̄_t) ≤ β̲ + α(β̄ − β̲)}_{t=1,...,K} ;
            C_t := Random(RCL) ;
            si t ≠ j entonces
                C_t := C_t ∪ {x} ;
                C_j := C_j − {x} ;
            fin
            Recalcular elementos representativos;
        fin
    hasta  No se realicen reasignaciones ;
    retornar C = {C_j}_{j=1,...,K}
fin
```

4. Experimentos Computacionales

En [3] se presenta un método para la identificación de módulos de un sistema basado en reglas de asociación. En este trabajo se utiliza, para la comprobación del método, un conjunto de datos que consiste en 28 programas escritos en COBOL, definidos como el conjunto $P = \{p_1, p_2, ..., p_{28}\}$, los cuales usan 36 archivos de datos $F = \{f_1, f_2, ..., f_{36}\}$. En el referido trabajo se utiliza la metodología ISA (*Identification of Subsystems based on Associations*) para identificar los

subsistemas basados en asociaciones. Esta metodología realiza como primer paso, una selección de los datos que considera mas relevantes para el proceso. El resultado de esta selección se le conoce como *AlphaSet*, y consiste en seleccionar aquellos programas que usen más de un valor γ de archivos, y seleccionar los archivos que usen más de un valor β de programas. Ambos parámetros deben ser enteros positivos y para el caso se usan los valores: $\gamma = 1$ y $\beta = 1$. Luego de este proceso previo, se obtiene un subconjunto $\tilde{P} \subset P$ de 22 programas y un subconjunto $\tilde{F} \subset F$ de 24 archivos de datos. Esas informaciones son procesadas por el método GraspKM, adaptado al clustering de software, para identificar los módulos del sistema, agrupando aquellos programas que acceden a archivos de datos similares. En el Cuadro (1) se muestran los datos a usar.

Nro.	Programas ($\tilde{P}$)	Archivo de datos usados ($\tilde{F}$)
1	p_1	f_3, f_5
2	p_2	f_3, f_5
3	p_5	f_3, f_5, f_{26}
4	p_6	f_3, f_5, f_{26}
5	p_8	f_3, f_5, f_{26}
6	p_9	f_3, f_5
7	p_{10}	f_3, f_5
8	p_{13}	f_3, f_5, f_{26}
9	p_{14}	f_3, f_5, f_{26}
10	p_{15}	f_3, f_5, f_{26}
11	p_{16}	$f_9, f_{10}, f_{12}, f_{18}, f_{19}, f_{22}, f_{23}, f_{24}, f_{26}, f_{27}, f_{29}, f_{30}, f_{32}, f_{33}, f_{34}, f_{35}, f_{36}$
12	p_{17}	f_{26}, f_{30}
13	p_{18}	$f_9, f_{10}, f_{12}, f_{17}, f_{18}, f_{19}, f_{22}, f_{23}, f_{24}, f_{25}, f_{26}, f_{27}, f_{29}, f_{32}, f_{33}, f_{34}, f_{35}, f_{36}$
14	p_{19}	$f_{10}, f_{12}, f_{17}, f_{19}, f_{22}, f_{23}, f_{24}, f_{25}, f_{26}, f_{27}, f_{29}, f_{32}, f_{33}, f_{34}, f_{35}, f_{36}$
15	p_{20}	$f_{14}, f_{23}, f_{29}, f_{32}$
16	p_{21}	f_5, f_{14}
17	p_{23}	$f_3, f_5, f_{23}, f_{26}, f_{27}, f_{28}$
18	p_{24}	f_3, f_5, f_{26}
19	p_{25}	f_{20}, f_{23}, f_{26}
20	p_{26}	f_3, f_{23}, f_{26}
21	p_{27}	f_{20}, f_{23}, f_{26}
22	p_{28}	$f_3, f_5, f_{23}, f_{26}, f_{28}$

Cuadro 1. Conjunto de programas a agrupar.

Cada uno de los 22 programas será representado con un vector binario de dimensión 24, donde el valor de 1 indicará el uso del archivo de datos en el orden respectivo.

Para determinar el valor apropiado para el parámetro α, se realizaron experimentos con un valor de $MAX\_ITER = 1,000$ para distintos valor de α. Los mejores resultados obtenidos para la función objetivo, expresión (5), se presentan en el Cuadro (2). El cuadro muestra que con un valor de $\alpha = 1$ se obtienen los mejores resultados, este valor es el mismo encontrado en las experiencias numéricas realizados en [17]. Es necesario resaltar que para con el valor de $\alpha = 1$, el método GraspKM no se comporta como un aleatorio puro, debido a las restricciones impuestas en la fase ConstruccionKM.

α	0	0.25	0.50	0.75	1.00
K = 3	10.237	10.237	10.237	10.237	7.028
K = 4	5.684	5.684	5.684	5.684	5.449

Cuadro 2. Calibración de parámetro α.

Con estos parámetros, compararemos los resultados obtenidos por el método GraspKM y el algoritmo KMeans adaptado también al clustering de entidades de software. La comparación se realiza en cuanto a la eficiencia para obtener la función objetivo f. Para el algoritmo KMeans se consideran $1,000$ ejecuciones y se considera el mejor valor obtenido. Los resultados se muestran en el siguiente Cuadro (3) y como se puede apreciar, el método GraspKM encuentra mejores valores de f para el conjunto de datos usado.

	KMeans	GraspKM
K = 3	10.237	7.028
K = 4	7.571	5.449

Cuadro 3. Valor de f obtenido por KMeans y GraspKM.

Los clusters encontrados por el método GraspKM cuando $K = 3$ y $K = 4$ se muestran en las cuadros 4 y 5. En ambos casos se muestra la configuración de clusters del mejor resultado obtenido para f con los parámetros de $MAX\_ITER = 1000$ y $\alpha = 1$.

Clusters	Programas
C_1	$p_1, p_2, p_5, p_6, p_8, p_9, p_{10}, p_{13}, p_{14}, p_{15}, p_{17}, p_{21}, p_{24}, p_{26}$
C_2	$p_{16}, p_{18}, p_{19}, p_{20}$
C_3	$p_{23}, p_{25}, p_{27}, p_{28}$

Cuadro 4. Clusters para $K = 3$.

Clusters	Programas
C_1	$p_1, p_2, p_5, p_6, p_8, p_9, p_{10}, p_{13}, p_{14}, p_{15}, p_{21}, p_{24}, p_{26}$
C_2	$p_{16}, p_{18}, p_{19}, p_{20}$
C_3	p_{17}, p_{25}, p_{27}
C_4	p_{23}, p_{28}

Cuadro 5. Clusters para $K = 4$.

Como se puede apreciar en los resultados, los clusters se encuentran definidos por programas que acceden a similares archivos de datos, lo cual nos da una idea de la estructura del sistema respecto a los datos que maneja.

5. Conclusiones y trabajos futuros

En el presente trabajo se adapta la metaheurística GRASP para el clustering de software. El método de-

nominado GraspKM , que aborda el problema del clustering como de optimización combinatoria y encuentra una solución eficiente basado en los centros, es adaptado en lo que respecta al uso una medida de la similaridad propia de vectores binarios y al uso de un vector representativo de los clusters. En las pruebas numéricas, el método demuestra ser superior que el algoritmo KMeans adaptado para el clustering de software. Esta método permite encontrar grupos de entidades de software que presenten características similares (cohesión) y a la vez diferentes de otros grupos (bajo acoplamiento), optimizando la función objetivo f propuesta.

Al igual que los métodos de clustering revisados en la literatura, el valor de K es un parámetro que debe ser ingresado. En este sentido, se puede extender el presente trabajo de manera que el valor de K puede ser determinado de manera automática.

El método propuesto puede ser extendido para su uso en el área de recuperación de información, donde se necesita encontrar grupos de documentos similares. Estos documentos generalmente se encuentran representados por vectores binarios donde 1 indica la presencia de cierta palabra en el documento; o vectores que contienen la frecuencia de ocurrencia de cierta palabra en el documento. En ambos casos, el método propuesto puede ser adaptado para su uso.

Referencias

[1] S. Bathia and J. Deogun. Conceptual clustering in information retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 28(3):427–436, 1998.

[2] E. Chavez, G.Ñavarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Compunting Surveys*, 33(3):273–321, 2001.

[3] C. M. de Oca and D. L. Carver. Identification of data cohesive subsystems using data mining techniques. In *ICSM '98: Proceedings of the International Conference on Software Maintenance*, page 16, Washington, DC, USA, 1998. IEEE Computer Society.

[4] D. Doval, S. Mancoridis, and B. Mitchell. Automatic clustering of software systems using a genetic algorithm. In *IEEE Proceedings of the 1999 International Conference on Software Tools and Engineering Practice (STEP'99)*, Pittsburgh, PA, August 1999.

[5] U. Fayyad, G. Piatetsky-Shapiro, and P. S. From data mining to knowledge discovery in databases. *American Association for Artificial Inteligence*, pages 37–54, 1996.

[6] T. Feo and M. Resende. Greedy randomized adaptative search procedure. *Journal of Global Optimization*, 6:109–133, 1995.

[7] A. Jain, Murty, and M. F. P. Data clustering: a review. *ACM Computer Surveys*, 31(3):264–323, 1999.

[8] C.-H. Lung, M. Zaman, and A.Ñandi. Applications of clustering techniques to software partitioning, recovery and restructuring. *J. Syst. Softw.*, 73(2):227–244, 2004.

[9] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Pattern Recognition*, 73:1551–1558, 1985.

[10] O. Maqbool and H. A. Babri. The weighted combined algorithm: A linkage algorithm for software clustering. volume 00, page 15, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[11] J. McQueen. Some methods for classification and analysis of multivariate observations. *In Preccedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[12] B. Mitchell and S. Mancoridis. Using heuristic search techniques to extract design abstractions from source code. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'03)*, Chicago, Illinois, July 2002.

[13] S. Patel, W. Chu, and R. Baxter. A measure for composite module cohesion. In *ICSE '92: Proceedings of the 14th international conference on Software engineering*, pages 38–48, New York, NY, USA, 1992. ACM Press.

[14] D. Pham and J. Prince. An adaptive fuzzy c-means algorithm for image segmentation in the presence of intensity inhomogeneities. *Pattern Recognition Letters*, 20(1):57–68, 1999.

[15] M. Saeed, O. Maqbool, H. Babri, S. Hassan, and S. Sarwar. Software clustering techniques and the use of combined algorithm. volume 00, page 301, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[16] P. Scheunders. A genetic lloyd-max image quantization algorithm. *Pattern Recognition Letters*, 17(5):547–556, 1996.

[17] E. Vicente, L. Rivera, and D. Mauricio. Grasp en la resolución del problema del clustering. In *CLEI 2005: XXXII Conferencia Latinoamericana de Informática*, Santiago de Cali, Colombia, 2005. CLEI.

[18] T. A. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *WCRE '97: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE '97)*, page 33, Washington, DC, USA, 1997. IEEE Computer Society.