Um Clustering na Recuperação de Componentes de Software

Erick Vicente[1], Luis Rivera[2], Manuel Tupia[3]

[1] Universidad Nacional Mayor de San marcos, Facultad de Ingeniería de Sistemas e Informática, Lima1, Lima, Perú
erick.vicente@gmail.com

[2] Universidad Federal Norte Fluminense, Departamento de Informática, Rio de Janeiro, Brasil, Campos dos Goytacazes 28015-620
rivera@uenf.br

[3] Pontifica Universidad Católica del Perú, Facultad de Ciencias e Ingeniería Lima100, Lima 32, Lima, Perú
tupia.mf@pucp.edu.pe

Abstract

Abstrac

Actually, there are Software Systems without documentation, especially when they are legacy systems. Exists many proposed methods to get a structure abstraction from these systems. The methods found in the literature to do this work are based in clustering, because the similar objectives between a software system structure and a clustering process: the software modules must to be high cohesive and low coupling, in similar way a cluster must to have similar objects and different from other clusters. The methods are based mainly in hierarchical clustering. In this work we propose an adaptation of KMeans algorithm inside the GRASP framework known as the GraspKM algorithm, to obtain an abstraction from software systems. This method solves the clustering as a combinatorial optimization problem, and it is efficient optimizing the objective function proposed.
Keywords: Software Clustering, GRASP, Kmeans.

Resumen

Resumen

Na atualidade existe grande quantidade de sistemas de software carentes de documentação, mais ainda quando são sistemas ligados. Na literatura tem se proposto diversos métodos para a obtenção de uma abstração da estrutura desses sistemas. Esses métodos encontram-se baseados principalmente em técnicas de clustering, devido aos objetivos coincidentes do que se quer da estrutura de um sistema e da estrutura dos clusters: os módulos de software devem ser altamente coesivos e com baixo acoplamento, de forma similar um cluster deve conter elementos que sejam similares entre si e que sejam, ao mesmo tempo, diferentes entre clusters. Os métodos encontrados na literatura para o clustering de software encontram-se classificados dentro do clustering hierárquico. Neste trabalho propomos a adaptação do método KMeans no contexto de GRASP, conhecido como GraspKM, para a busca da estrutura de um sistema. Este método trata o clustering como um problema de otimização combinatória e demonstra ser eficiente otimizando a função objetivo proposta.
Palabras clave: Software de agrupamiento, GRASP, Kmédias.

1. Introdução

Em Engenharia de Software, nas linhas de pesquisa de reuso e engenharia reversa, os componentes de software, como: programas, rotinas, procedimentos, etc. devem ser estruturados e reusados por diferentes sistemas. Nessa perspectiva, os mecanismos de agrupamento, tipo clustering, tem sido importante no desenvolvimento de técnicas para o particionamento, recuperação e reestruturação de software [8].

A recuperação é um dos principais problemas que se encontra na Engenharia de Software, o objetivo é obter um modelo do sistema que permita ter um melhor entendimento de como este se encontra estruturado. Quando se trata de sistemas ligados, que foram elaborados sem prévia documentação de design, obter essa abstração de alto nível é realmente importante porque ajudará aos engenheiros de software a ter um melhor entendimento da arquitetura quando se precisem realizar modificações ao sistema. A busca desses grupos se realiza de forma que satisfaça os critérios de coesão e acoplamento. Isto é, os grupos a encontrar devem ter um alto grau de coesão, e baixo acoplamento com respeito a outros grupos.

Estes objetivos concordam com os do clustering, onde se busca obter grupos que sejan os mais similares entre si, e no mesmo tempo diferente dos outros clusters. Witggerts [18] os denomina entidades aos objetos a agrupar e esses podem ser programas, funções ou procedimentos.

Existem duas formas de representação do software para recuperar sua estrutura. A primeira é a través dos grafos, onde os nós representam as entidades, e os vértices às relações que existem entre eles. O objetivo é encontrar grupos contendo nós altamente relacionados (alta coesão) e que tenham menos relações possíveis entre elos (baixo acoplamento), estes grupos serão os módulos ou subsistemas do software em análise. Devido a que este é um problema NP-Difícil, tem sido proposto diversos métodos heurísticos e metaheurísticos para encontrar uma solução eficiente do problema [12, 4].

A segunda forma, abordada neste trabalho, é a través de vetores contendo as características de entidades de software. Essas características podem ser referências a variáveis, chamadas a outros módulos ou tipos de acessos. A representação neste caso é mediante vetores binários, onde 1 indica a presença da característica, e 0 sua ausência. Usando técnicas de clustering se buscam grupos contendo entidades de alta coesão e baixo acoplamento. Na literatura podem-se encontrar diversos métodos de clustering, classificados como jerárquicos, para encontrar soluções ao problema [18, 13, 15, 10]. Esses métodos têm a desvantagem que demandam um alto custo computacional na busca de soluções [7]. Justamente aqui é necessário um método de clustering de otimização baseado em centros. Os centros são aqueles elementos que melhor representam ao cluster, e por isso denominaremos elemento representativo.

Clustering determina grupos de objetos similares entre si e, ao mesmo tempo, diferentes aos objetos de outros grupos, satisfazendo um critério estabelecido. Os grupos com características similares são conhecidos como Clusters. Os objetos são representados por vetores no espaço R^D, e com o uso de uma medida de similaridade, como a distância, são definidos os clusters. Não existe conhecimento prévio no que diz respeito de como se deve definir um cluster; por tanto, o processo de clustering é conhecido como uma classificação não supervisionada.

O clustering tem múltiplas aplicações dentro da Ciência da Computação, como compressão de imagens [16] e voz digitalizada [9]; na recuperação de informação [1]; em mineração de dados [3,5] na segmentação de imagens médicas [14], na classificação de componentes de software [8], e outros.

Neste trabalho propomos um mecanismo de agrupamento de entidades de software, estabelecendo uma quantificação dos atributos dos componentes de forma que possam ser usados por técnicas de clustering baseado em KMeans no entorno da metaheurística GRASP, a qual é uma técnica puramente numérica. Esse mecanismo vai permitir selecionar elementos representativos de cada grupo de um conjunto grande de componentes de software.

O resto do trabalho esta organizado da seguinte forma: na Seção 2 se apresenta Clustering e se analisam os trabalhos relacionados, como a metaheurística GRASP; na Seção 3 se apresenta a adaptação do método GraspKM para o clustering de software; na Seção 4 são mostrados os experimentos computacionais realizados. Finalmente, na Seção 5 se expõem as conclusões e trabalhos futuros.

2. Antecedentes

Nesta seção se apresenta uma revisão dos principais conceitos para a aplicação das técnicas de clustering na Engenharia de Software. Primeiro se define o

problema do clustering e se revisam as medidas de similaridades empregadas para o clustering de software. Depois, são revisados os trabalhos existentes na literatura. Finalmente, se apresenta a metaheurística GRASP.

2.1. Problema de Clustering

Dado um conjunto de n objetos $X = \{x_1, x_2, ..., x_n\}$, $x_i \in R^D$. Seja K um número inteiro positivo, o problema de clustering consiste em encontrar uma partição $P = \{C_1, C_2, ..., C_K\}$ de X, sendo C um cluster definido por objetos similares, satisfazendo uma função objetivo $f : R^D \rightarrow R$ que representa a distância mínima entre os objetos do cluster, e as condições:

$$C_i \cap C_j = \varnothing \text{ para } i \neq j, \text{ y } \bigcup C_i = X.$$

A definição da função de similaridade depende da natureza dos objetos a agrupar.

2.2. Medidas de similaridade

Diversas medidas têm sido propostas para medir a similaridade entre objetos. Wiggerts [18] classifica as medidas de similaridade em quatro tipos: medidas da distância, coeficientes de associação, coeficientes de correlação e medidas probabilísticas. Para o presente trabalho, os dois primeiros tipos de medidas são de interesse. Os coeficientes de associação se usam principalmente para medir a similaridade entre dois vetores binários, obtendo valores no intervalo [0,1] quanto mais próximo a 0 indicaria que os objetos são similares.

Medidas da distância

A noção de similaridade entre duas entidades representadas por dois vetores x e y $\in$ R^D, é caracterizada por uma função distância como:

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

Diz-se que dois objetos $x = (x_1, ..., x_D)$ e $y = (y_1, ..., y_D)$ são similares quando a distância entre os vetores é menor que uma tolerância pequena. Em [2] se descrevem as propriedades que deve ter a distância.
A escolha de uma função distância entre os vetores depende do grau de dificuldade das entidades e sua interpretação. A mais usada é a distância Euclidiana, definida por:

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

A distância Euclidiana é um caso especial da medida de Minkowski quando $p=2$, dada por:

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

Coeficientes de associação

Os coeficientes de associação, ou de comparação, para duas entidades E1 e E2 representadas por vetores binários x e y, respectivamente, estão dadas pelo número de atributos coincidentes de uma entidade em relação a outra. Lung et al. [8] classifica a este tipo de coeficientes como qualitativos, devido a que calcula a similaridade baseada na ausência ou presença de atributos. Según Wiggerts [18] são quatro casos de associação entre as entidades respeito ao número de seus atributos presentes em ambas as entidades (a); presentes em E1 mais não em E2 (b) presentes em E2 mais não em E1 (c); e não presentes em ambos (d). Denota-se por 1 binário como presença de um atributo numa entidade, e por 0 a ausência, é apropriado relacionar a ocorrência desses atributos por uma tabela definida como: por exemplo, sejam duas entidades $E1$ e $E2$, descritas através de dois vetores binários $x = (0,1,0,1,1,1)$ $y = (0,1,1,0,1,0)$, respectivamente. Então, a=2 porque dois atributos estão presentes (na segunda e quinta posição de ambos os vetores, caso 1-1). O valor b=2 porque os atributos quarto e sexto estão em x mas não em y (caso 1-0). Assim, se observam que c=1 (caso 0-1) e d=1 para (caso 0-0).

		E2	
		1	0
E1	1	a	b
	0	c	d

Existem diversos métodos para calcular os coeficientes de associação; eles estão baseados principalmente, na relevância das coincidências entre ambos os vetores e a ponderação que lhe atribuir. Os principais métodos para o cálculo de coeficientes entre dois vetores x e y, usados em [15, 18, 8], são:

- 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**: $S_r(x, y) = 2a/(2a+b+c)$

Observa-se que o coeficiente de Jaccard e Sorensen considera relevante a relação 1-1, mas não a relação 0-0 já que essa indica a ausência de atributos. O coeficiente simples considera relevante tanto a relação 1-1, como a 0-0.

Em [13], se propõe uma medida de similaridade em função de produto e norma de vetores binários x e y, a qual também é usada para calcular a similaridade entre documentos por métodos de recuperação de informação. A medida esta dada pela seguinte expressão:

No mesmo trabalho, se estende esta função de medida a vetores não binários, onde os valores expressam a freqüência de ocorrência de certa característica. Por exemplo, se $x=(x_1,...,x_n)$ e $y=(y_1,...,y_n)$ representam as duas entidades de software (i.e. programas), então os valores x_i e y_i podem expressar a quantidade de vezes que dados tipo T_i são declarados em ditos programas. A função de medida estendida envolve produto interno de vetores,

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

2.3. Clustering de Software

Aqui se faz uma revisão de dois métodos de clustering de software baseados em métodos hierárquicos, e que usam um vetor representativo para a busca dos clusters. Em [15] é apresentado um método para agrupar entidades de software representadas por vetores binários. Uma das características relevantes do método é que a partir dos vetores que compõem um cluster se obtém um vetor representativo. Este vetor é obtido aplicando o operador *OR* aos vetores binários que compõem o cluster. Por exemplo, se os vetores $x_1=(0,1,1)$ e $x_2=(0,0,1)$ compõem o cluster C então, o vetor representativo do cluster deverá ser $x_C=(0,1,1)$ Os clusters vão se construindo a través do método hierárquico Weighted Average Linkage, e a medida de similaridade usada é o coeficiente de associação de Jaccard. O método proposto tem a inconveniência que ao se aplicar o operador *OR* para obter o vetor representativo do cluster, se ignora a quantidade de entidades que apresentam a mesma característica. Por exemplo, suponha os clusters $A=\{(0,1,0),(0,1,0),(1,1,0)\}$ $B=\{(0,0,1),(0,1,1)\}$, então os vetores representativos de A e B são $x_A=(1,1,0)$ e $x_B=(0,1,1)$, respectivamente. Ao calcular o coeficiente de associação de um vetor $x_6=(0,1,0)$ aos clusters A ou B se obteria o mesmo valor, e se poderia atribuir a qualquer deles. Embora se observe que o cluster A tenha mais vetores com valor $x_{i2}=1$, e por tanto o lógico será que se associe ao cluster A.

Em [10] se apresenta uma melhora ao método proposto em [8]. O cálculo do vetor representativo se baseia na freqüência com que se apresentam as características nas entidades que compõem o cluster. Isto é, para o caso anterior: o vetor representativo do cluster A, será $x_A=(1/3, 3/3, 0/3)$ e para o cluster B, será $x_B=(0/2, 1/2, 2/2)$ Isto quer dizer que o elemento representativo de um cluster $C=\{(x_{11},...,x_{1d}),...,(x_{n1},...,x_{nd})\}$ esta dado por

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

O vetor representativo deixa de ser binário e por tanto não se poderia aplicar os coeficientes de associação apresentados anteriormente. O autor estende a medida de similaridade para ter em conta as freqüências das características. À medida da similaridade proposta lhe denomina unbiased Ellenberg, e está dada pela seguinte expressão:

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

onde M_a representa a soma das características que estão presentes em ambas entidades, e b e c representam a quantidade de características que estão presentes numa entidade e não na outra. Com essa medida de similaridade, em [10] se utilizam métodos hierárquicos de clustering melhorando os resultados obtidos em [15].

2.4. Metaheurística GRASP

Um procedimento de busca voraz aleatória e adaptativa (GRASP) é uma metaheurística proposta por Feo e Resende [6] para encontrar soluções aproximadas de problemas de otimização combinatória, mediante um processo iterativo.

Em cada iteração se realizam os processos de construção e busca local. Na construção se gera um conjunto solução s de uma instância E e na busca local se obtém uma possível melhor solução que S; finalmente, se escolhe a melhor solução entre a solução da iteração anterior e a atual. A melhor solução será avaliada por uma função objetivo f. Todo o processo é repetido um número máximo de vezes *(MAX_ITER)*. O Algoritmo 1 mostra o contexto geral da meteheurística GRASP.

Algoritmo 1: Grasp

```
entrada : E, (MAX_ITER), α
1  inicio
2      Inicializar solução S := ϕ : f* = ∞;
3      para i = 1 Até MAX_ITER faça
4          S* := Construcao_Grasp(E, α) ;
5          S* := Busca_Local Grasp( S* ) ;
6          se f(S*) < f* então
7              Atualizar S := S* e f* := f(S*) ;
8          fim
9      fim
10     retorna S
11 fim
```

O fato de que a busca local tenha como entrada a solução obtida na construção proporciona um conhecimento diante dos algoritmos de busca local tradicionais.

Construção GRASP

Nesta fase se adapta um algoritmo guloso que seleciona o melhor elemento de um conjunto de candidatos C para serem incorporados na solução. O critério da seleção gulosa depende do problema, pode ser maximização ou minimização. O construtor de soluções evita a característica determinística dos algoritmos gulosos, utilizando um parâmetro de relaxação $\alpha \in [0,1]$ para formar uma lista restringida de candidatos (Restricted Candidate List - RCL) no entorno do melhor elemento a se selecionar. Uma vez definidos os elementos de RCL, um elemento de RCL será eleito aleatoriamente como uma possível solução do problema. Esta forma estocástica de seleção permite a construção de soluções com tendência a melhores elementos e evita cair em ótimos locais. O parâmetro de relaxação α indica o tamanho do RCL no entorno do melhor candidato. O melhor valor de α para o problema em estudo se obtém a través de múltiplos experimentos computacionais de calibração.

Busca Local GRASP

A busca local se realiza de forma iterativa, explorando na vizinhança de um conjunto de soluções S gerados na fase de construção. O perfomance da operação da busca local dependerá do método elegido. Se N é uma vizinhança de soluções, se diz que $S' \in N(S)$ é um ótimo local se $f(S') < f(S)$. Não existe um esquema de busca local específico a se utilizar, mas é necessário que melhore a solução encontrada na construção.

3. Software Clustering usando GRASP

Na presente seção se apresenta a adaptação do método GraspKM [17] ao clustering de software. Este método se encontra dentro do enfoque da metaheurística GRASP [6] para encontra uma solução viável ao problema do hard clustering; isto é, quando os grupos encontrados são partições do conjunto de objetos em análise. O método GraspKM é uma adaptação do algoritmo KMeans [11] dentro do contexto da metaheurística GRASP.

O algoritmo KMeans constrói os clusters iterativamente, partindo de K possíveis centros selecionados aleatoriamente de um conjunto X de N elementos. Os clusters se definem atribuindo cada elemento de X, de forma que a distância respeito ao possível centro seja mínima respeito a outros centros. Em cada iteração seguinte, se recalcula os centros dos clusters como a media aritmética dos elementos que o compõem; e se as variações dos centros ainda persistem então continua com uma nova iteração. Esse processo é repetido até que os centros não variem.

Algoritmo 2: GraspKM

entrada: X, K, α, MAX_ITER

```
1  início
2      f* := ∞ , C = { };
3      para i = 1 Até MAX_ITER faça
4          C' := InicializacaoKM(X , K) ;
5          C' := ConstrucaoKM(X , K , C' , α ) ;
6          C' := MelhoriaKM(X , K , C' ) ;
7          se f(C') < f* então
8              C := C' ;
9              f* := f(C') ;
10         fim
11     fim
12     retorna C
13 fim
```

O GraspKM, mostrado no Algoritmo 2, inicia de maneira similar ao KMeans. Em fase InicializacaoKM, são selecionados aleatoriamente K objetos como centros e se formam os clusters iniciais atribuindo cada um dos objetos ao cluster cujo centro se encontre mais próximo. Na fase ConstrucaoKM são designados iterativamente cada um dos objetos ao cluster escolhido aleatoriamente de uma RCL. Este processo se repete até que não haja mais designações. Finalmente, se obtém uma melhor solução a través de um algoritmo de busca local na fase de *MelhoriaKM*.

O método GraspKM utilizando como medida de similaridade a distância Euclidiana e mostra ser superior ao algoritmo K-Means e comparável com as outras meta-heurísticas. Embora que este método seja eficiente encontrando soluções para objetos representados por vetores em R^d, este não pode ser aplicado diretamente ao clustering de software. Este deve ser adaptado para cobrir os seguintes pontos:

- As medidas de similaridade no clustering de software não estão baseadas especificamente na distância, senão nos coeficientes de associação devido a que se trabalha com vetores binários.
- Os coeficientes de associação permitem o cálculo de similaridade entre dois vetores e entre grupos de vetores, porem a obtenção de um vetor centro que represente ao grupo não é de maneira natural, mas sim permite a distância Euclidiana.

Nesta perspectiva, deve-se adaptar o método proposto em [17] para agrupar vetores binários que representam às entidades de software. Como descrito na seção anterior, em [10] propõe-se uma medida de similaridade chamada unbiased Ellenberg (2), que considera a freqüência com que se definem as características nos componentes de software. Esta medida obtém um valor entre $[0,1]$, onde um valor de 1, ou próximo, indica a similaridade. Para poder formular a função objetivo respeito à minimização, se requer ter uma medida com tendência a zero quando

os objetos são similares. Por tanto, a medida de similaridade que usamos é:

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

Esta medida permitirá calcular a similaridade entre o elemento representativo do cluster e um vetor binário. No referido trabalho [10], se propõe uso de um elemento representativo para o clustering de software, embora este não seja um método de clustering baseado em centros como o algoritmo KMeans, mas é possível usar este elemento como centro, inclusive estes coincidem com os centros usados no algoritmo KMeans. Por tanto, os elementos representativos que usamos para o algoritmo GraspKM estarão dados pela expressão (1).

Como o que se quer é obter clusters com entidades de software que sejam o mais similares entre si. Então, pode-se definir empiricamente a similaridade de um cluster C como

$$S(C)=\sum_{x\in C} S_m\left(x,\%_{c}^{0}\right)$$

Em base a essa expressão pode-se formular a função objetivo f como:

$$f=\sum_{j=1}^{K} s_m\left(C_j\right)$$

onde K é o número de clusters que se deseja definir e o objetivo do método a desenvolver deve ser minimizar essa função.

Na fase InicializacaoKM, são elegidos aleatoriamente K vetores de X como centros, se conformam os clusters iniciais associando cada um dos elementos de X ao cluster mais próximo. Depois, são recalculados novamente os elementos representativos.

Como os elementos representativos podem ter variado, os objetos devem ser designados novamente aos clusters mais próximos. Este é feito a través do processo iterativo chamado ConstrucaoKM, tal como é mostrado no Algoritmo 3, onde os possíveis clusters que conteria ao objeto x em análise, são agrupados num conjunto RCL cujas medidas de similaridade entre o cluster e o objeto x estão num intervalo definido por $\overline{\beta}$ e $\underline{\beta}$ e regulada linearmente pelo parâmetro de relaxação α Do conjunto RCL será escolhido aleatoriamente um cluster ao qual será retribuído o objeto x, e retirando-o do cluster onde se encontrava previamente. Depois da designação de todos os objetos de X os elementos representativos podem ter variado; por tanto, os centros devem novamente ser recalculados. O processo termina quando os elementos representativos não variam.

Algoritmo 3: ConstruccionKM

```
   entrada: X, K, C, α
1  início
2  repita
3     para cada x∈X tal que x∈C_{j=1,...,K}, fazer
4        β̄ := Max {S_m(x, x̄_l)}_l;
5        S_m(x, x̄_l) ≤ {S_m(x, x̄_j)}_{l=1,...,K};
6        β̲ := Min {S_m(x, x̄_l)}_{l=1,...,K};
7        RCL := C_l;
8        S_m(x, x̄_l) ≤ β̲ + α(β̄ − β̲)}_{l=1,...,K}
9        C_t := Random(RCL);
10       se t ≠ j então
11          C_t := C_t ∪ {x};
12          C_j := C_j − {x};
13       fim
14       Recalcular elementos representativos;
15    fim
16 até  Não se realizam designações;
17 retorna C = {C_j}_{j=1,...,K}
18 fim
```

As soluções obtidas na fase de construção são refinadas na fase denominada melhoria (MelhoriaKM), que é um processo iterativo de duas fases: reagrupação KM (ReagrupacaoKM) e Construção KM (ConstrucaoKM), que se repetem até que a solução não possa ser melhorada, como é mostrada no Algoritmo 4.

Algoritmo 4: MelhoriaKM

```
   entrada: X, K, C, α
1  início
2     repita
3        C := C';
4        C' := ReagrupacaoKM(X, K, C);
5        C' := ConstrucaoKM(X, K, C', α);
6     até f(C') >= f(C);
7     retorna C = {C_j}_{j=1,...,K}
8  fim
```

A idéia da reagrupação é eliminar e gerar novos clusters heuristicamente. Elimina-se e se gera um novo cluster ao mesmo tempo; o cluster a eliminar é aquele que apresenta a menor quantidade de objetos, e se divide aquele cluster que tiver a maior dispersão, isto devido a que em ambos os casos o processo pode ter caído num ótimo local. Se C_l é o cluster com menor número de elementos, então l está dado por:

$$l=ArgMin\left\{\left|C_l\right|\right\}_{j=1,\ldots,K}$$

O indicador ao cluster C_h de maior dispersão é calculado como:

$$h=ArgMin\left\{\frac{S\left(C_j\right)}{\left|C_j\right|}\right\}_{j=1,\ldots,K\ j\neq l}$$

onde, $S(C_j)$ é a similaridade do cluster C_j e é calculada usando a expressão (4). Após da eliminação do cluster e gerado um novo, cada um dos objetos de X são atribuidos aos novos centros. Finalmente, os clusters obtidos são refinados com o processo ConstrucaoKM. O Algoritmo reagrupacaoKM é mostrado a seguir:

Algoritmo 5: ReagrupacaoKM

entrada: X, K, C

1 início

2 $l := ArgMin\{C_j|\}_{j=1,...,K}$

$h := ArgMin\left\{\frac{S(C_j)}{|C_j|}\right\}_{j=1,...,K, j\neq l}$;

4 $\overline{x_r} := Random(C_h)$;

5 Substituir $\overline{x_l}$ por $\overline{x_r}$;

6 **para cada** $x \in X$ **fazer**

7 atribuir x a C_j, onde $j := ArgMin\{d(x, \overline{x_i})\}_{i=1,...,K}$;

8 **fim**

9 Calcular os centros $\{\overline{x_j} := Media(C_j)\}_{j=1,...,K}$;

10 **retorna** $C = \{C_j\}_{j=1,...,K}$

11 **fim**

4. Experimentos Computacionais

Em [3] se apresenta um método para a identificação dos módulos de um sistema baseado em regras de associação. Nesse trabalho se utiliza, para a comprobação do método, um conjunto de dados que consiste em 28 programas escritos em COBOL, definidos como o conjunto $P = \{p_1, p_2, ..., p_{28}\}$ os quais usam 36 arquivos de dados $F = \{f_1, f_2, ..., f_{28}\}$ No referido trabalho é utilizada a metodologia ISA (Identification of Subsystems based on Associations) para identificar os subsistemas baseados em associações. Essa metodologia realiza como primeiro passo, uma seleção dos dados que considera mais relevantes para o processo. O resultado de essa seleção é conhecida como AlphaSet, e consiste em selecionar aqueles programas que usem mais de um valor γ le arquivos, e selecionar os arquivos que usem mais de um valor β de programas. Ambos os parâmetros devem ser inteiros positivos, e para o caso se usam os valores $\gamma = 1$ $\beta = 1$. Após este processo prévio, obtém-se um subconjunto $\overline{P} \subset P$ de 22 programas e um subconjunto $\overline{F} \subset F$ de 24 arquivos de dados. Essas informações são processadas pelo método GraspKM, adaptado ao clustering de software, para identificar os módulos do sistema, agrupando aqueles programas que acedam aos arquivos de dados similares. Na Tabela 1 se mostram os dados a serem usados.

Nro.	Programas ($\bar{P}$)	Arquivo de dados usados ($\bar{F}$)
1	p1	f3, f5
2	p2	f3, f5
3	p5	f3, f5, f26
4	p6	f3, f5, f26
5	p8	f3, f5, f26
6	p9	f3, f5
7	p10	f3, f5
8	p13	f3, f5, f26
9	p14	f3, f5, f26
10	p15	f3, f5, f26
11	p16	f9, f10, f12, f18, f19, f22, f23, f24, f26, f27,
12	p17	f26, f30
13	p18	f9, f10, f12, f17, f18, f19, f22, f23, f24, f25, f26, f27, f29, f32, f33, f34, f35, f36
14	p19	f10, f12, f17, f19, f22, f23, f24, f25, f26, f27,
15	p20	f14, f23, f29, f32
16	p21	f5, f14
17	p23	f3, f5, f23, f26, f27, f28
18	p24	f3, f5, f26
19	p25	f20, f23, f26
20	p26	f3, f23, f26
21	p27	f20, f23, f26
22	p28	f3, f5, f23, f26, f28

Tabela 1. Conjunto de programas a serem agrupados.

Cada programa, dos 22 programas, será representado com um vetor binário de dimensão 24, onde o valor de 1 indicará o uso do arquivo de dados na ordem respectiva.

Para determinar o valor apropriado para o parâmetro α, se realizaram experimentos com um valor de $MAX\_ITER = 1,000$ para distintos valores de α. Os melhores resultados obtidos para a função objetivo, expressão (5), se apresentam na Tabela 2. A tabela mostra que com um valor de $\alpha = 1$ se obtém os melhores resultados, este valor é o mesmo encontrado nas experiências numéricas realizadas em [17]. É necessário ressaltar que com o valor de $\alpha = 1$, o método GraspKM não se comporta como um aleatório puro, devido ás restrições impostas na fase ConstrucaoKM.

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

Tabela 2. Calibrado de α.

Com estes parâmetros, compararemos os resultados obtidos pelo método GraspKM o algoritmo KMeans adaptado também ao clustering de entidades de software. A comparação se realiza em quanto á eficiência para se obter a função objetivo f. Para o algoritmo KMeans se consideram 1,000 execuções e se considera o melhor valor obtido. Os resultados se mostram na Tabela 3, e como pode se ver o método GraspKM encontra melhores valores de f para o conjunto de dados usados.

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

Tabela 3. Valor de *f* obtido por KMeans e GraspKM.

Os clusters encontrados pelo método GraspKM quando K = 3 e K = 4 se ilustram nas Tabelas 4 e 5. Em ambos os casos se apresenta a configuracão dos clusters do melhor resultado obtido para f com os parâmetros de MAXITER = 1000 e $\alpha = 1$.

Clusters	Programas
C1	*p1, p2, p5, p6, p8, p9, p10, p13, p14, p15, p17, p21, p24, p26*
C2	*p16, p18, p19, p20*
C3	*p23, p25, p27, p28*

Tabela 4. Clusters para $K = 3$.

Clusters	Programas
C1	*p1, p2, p5, p6, p8, p9, p10, p13, p14, p15, p21, p24, p26*
C2	*p16, p18, p19, p20*
C3	*p17, p25, p27*
C4	*p23, p28*

Tabela 5. Clusters para K = 4.

Como pode ser apreciado nos resultados, os clusters se encontram definidos por programas que acedam a similares arquivos de dados, o qual dá uma idéia da estrutura do sistema respeito aos dados que manipula, isto, também, ilustrado nas Figuras 1 e 2.

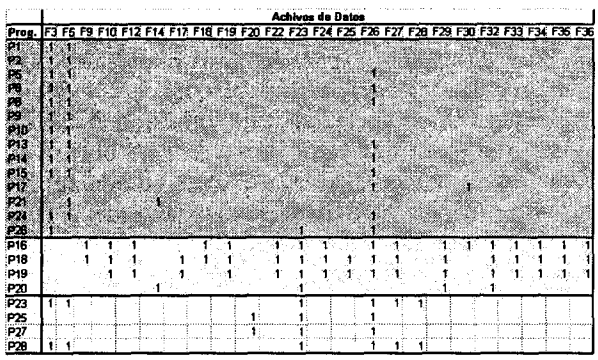

Figura 1. Programas e arquivos de dados para K = 3.

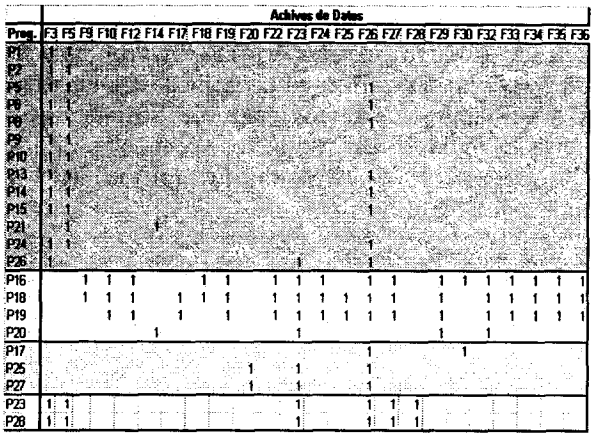

Figura 2. Programas e arquivos de dados para K = 4.

5. Conclusões e trabalhos futuros

No presente trabalho se adapta a metaheurística GRASP para o clustering de software. O método chamado GraspKM, que aborda o problema do clustering como de otimização combinatória e encontra uma solução eficiente baseado nos centros, é adaptado no que respeita ao uso de uma medida de similaridade própria de vetores binários e ao uso de um vetor representativo dos clusters. Nos testes numéricos, o método mostra ser superior que o algoritmo KMeans adaptado para o clustering de software. Este método permite encontrar grupos de entidades de software que apresentem características similares (coesão) e ao mesmo tempo diferentes de outros grupos (baixo acoplamento), otimizando a função objetivo f proposta.

Ao igual que os métodos de clustering revisados na literatura, o valor de K é um parâmetro que deve ser ingressado ao programa. Neste sentido, pode se estender o presente trabalho de maneira que o valor de K possa ser determinado de maneira automática.

O método proposto pode ser estendido para seu uso na área de recuperação de informação, onde se necessita encontrar grupos de documentos similares. Estes documentos geralmente se encontram representados por vetores binários onde 1 indica a presença de certa palavra no documento; os vetores que contem a freqüência de ocorrência de certa palavra no documento. Em ambos os casos, o método proposto pode ser adaptado para seu uso.

Referências bibliográficas

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

[2] Chavez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. Searching in metric

spaces. ACM Compunting Surveys 33, 3 (2001), 273–321.

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

[4] Doval, D., Mancoridis, S., and Mitchell, B. 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] Fayyad, U., Piatetsky-Shapiro, G., and S., P. From data mining to knowledge discovery in databases. American Association for Artificial Inteligence (1996), 37–54.

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

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

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

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

[10] Maqbool, O., and Babri, H. A. The weighted combined algorithm: A linkage algorithm for software clustering. IEEE Computer Society, Proceedings of the Eighth European Conference on Software Maintenance and Reengineering (CSMR'04), 2004, p. 15.

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

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

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

[15] Saeed, M., Maqbool, O., Babri, H., Hassan, S., and Sarwar, S. Software clustering techniques and the use of combined algorithm. vol. 00, IEEE Computer Society, p. 301.

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

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

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