Submetido para TEMA

Um Algoritmo de Construção e Busca Local para o Problema de Clusterização de Bases de Dados [1]

S. S. R. F. SOARES, L. SATORU OCHI, L. M. A. DRUMMOND[2], Instituto de Computação, Universidade Federal Fluminense, 24210-240, Niterói, RJ, Brasil.

Resumo. O Problema de Clusterização de uma base de dados, embora já tenha sido bastante explorado por pesquisadores de áreas como matemática, estatística e computação, traz na maioria dos trabalhos apresentados, uma abordagem do caso em que o número de clusters é previamente fixado pelo usuário como um parâmetro de entrada. Entretanto, em muitas aplicações práticas o número de clusters é uma variável que deve ser determinada pelo algoritmo. Esta generalização é denotada por Problema de Clusterização Automática (PCA). Neste trabalho, apresentamos um algoritmo de construção e busca local através de sobreposição e inversão de janelas para o PCA e demonstramos sua eficiência comparado-o com um Algoritmo Genético que até então apresentava os melhores resultados para este tipo de problema.

1. Introdução

Clusterização é uma técnica de agrupar dados (objetos) de uma base de dados de acordo com alguma medida de similaridade ou de dissimilaridade. Os métodos de clusterização podem ser classificados como hierárquicos ou não-hierárquicos, também conhecidos como particionais. Os métodos não-hierárquicos dividem o conjunto de objetos de uma base de dados em vários subconjuntos disjuntos e procuram iterativamente o melhor particionamento até atingir uma condição de parada. Os métodos hierárquicos são aqueles que constroem a clusterização através de uma árvore de clusters (dendograma). Os métodos hierárquicos são divididos ainda em aglomerativos, onde cada cluster é iniciado com um objeto da entrada e dois ou mais clusters de um nível anterior são recursivamente agrupados para formarem um novo cluster no próximo nível, e métodos de particionamento, que iniciam com um único cluster formado por todos os objetos da base de dados, e que a cada nível, cada cluster é dividido em dois ou mais clusters conforme a similaridade.

Dentre os algoritmos não-hierárquicos, o algoritmo K-means é um dos mais conhecidos e representa cada cluster pelo seu centróide, tendo como função objetivo a minimização da soma das distâncias entre cada ponto e o centróide do cluster ao qual pertence. Inicialmente, são escolhidos K objetos para representar o centróide

[1]Parcialmente financiado com recursos do CNPq.

[2]ssoares@ic.uff.br, satoru@dcc.ic.uff.br, lucia@dcc.ic.uff.br

de cada um dos K clusters, e após a inserção de cada um dos demais objetos em um cluster, o centróide é recalculado. Um ponto fraco do algoritmo K-means é a dependência da escolha inicial dos centróides dos clusters, o que pode levá-lo à soluções muito pobres. Além disso, por depender da prefixação do número de clusters, não é aplicável nos casos em que o número de cluster deve ser encontrado pelo algoritmo. Uma abordagem híbrida apresentada em [1] ameniza o problema das paradas prematuras do K-means em ótimos locais ainda distantes de um ótimo global aplicando conceitos de Algoritmos Genéticos - (AG's) [6], onde cada indivíduo da população é uma solução encontrada através do K-means. As abordagens mais comuns apresentam uma medida de similaridade baseada na distância entre objetos, o que resulta em clusters com forma convexa. Entretanto, algumas outras abordagens apresentam soluções com clusters de formato não-convexo, como em [5].

Muitas abordagens para o problema de clusterização utilizam metaheurísticas para encontrar uma boa solução. Em [2] foi utilizado um Algoritmo Evolutivo, técnica baseada na evolução de sistemas naturais através de seleção e combinação de características de soluções, cujo principal representante são os AG's. Uma abordagem híbrida de *Greedy Randomized Adaptive Search Procedure* - GRASP [8], metaheurística que combina aspectos de algoritmos gulosos, adaptativos e randômicos, com Busca Tabu [3] foi usado em [7] para um problema de *timetabling*, que é uma extensão do problema de clusterização.

A necessidade de encontrar automaticamente o número de clusters de um conjunto e ao mesmo tempo encontrar a clusterização ideal, torna o problema ainda mais complexo. Em [4], é apresentado um algoritmo que, através de conceitos ligados à Lógica Fuzzy com probabilidade de pertinência de um objeto a um cluster, resolve o problema de clusterização com ruídos. Em [9], é apresentado o CLUSTERING, um AG para clusterização automática que faz uso de uma heurística baseada em busca binária para o cálculo da clusterização mais adequada através de várias chamadas do próprio AG com variações de um peso que, na função de aptidão, prioriza uma clusterização com muitos ou poucos clusters.

Neste trabalho, apresentamos um algoritmo de construção e busca local através de sobreposição e inversão de janelas para o Problema de Clusterização Automática que faz uso da mesma função de aptidão do CLUSTERING. Nós mostramos a eficiência das técnicas de construção e busca local adotadas quando comparado aos resultados obtidos pelo AG proposto em [9], que até então obtinha os melhores resultados para problemas de CA.

O restante deste trabalho está dividido da seguinte forma: na seção 2 apresentamos a formalização do problema de clusterização e a estratégia de construção. A seção 3 mostra o nosso algoritmo de costrução e busca local. A seção 4 mostra a heurística usada para a obtenção da melhor clusterização dada em [9]. A eficiência do nosso algoritmo é mostrada na seção 5, onde comparamos a nossa abordagem com a apresentada em [9], e finalmente na seção 6, são apresentadas as conclusões.

2. O Problema de Clusterização

Clusterização em um espaço euclidiano n-dinensional R^n é o processo de particionar um conjunto de N pontos em k grupos (clusters) baseado em alguma medida de similaridade ou dissimilaridade. Em alguns casos, o valor k é previamente conhecido e em outros este valor deve ser encontrado. Neste trabalho abordamos o caso onde o número de clusters k é uma variável a ser definida pelo algoritmo de clusterização, conhecido por Clusterização Automática (CA). Assim, seja S o conjunto de N pontos $\{x_1, x_2, ..., x_N\}$ e $C_1, C_2, ..., C_k$ os k clusters. Então, devemos ter:

$$C_i \neq \emptyset \qquad i = 1, \cdots, k$$
$$C_i \cap C_j = \emptyset \quad i, j = 1, \cdots, k \text{ e } i \neq j \qquad \bigcup_{i=1}^{k} C_i = S$$

A fase de construção proposta reduz o tamanho da base de dados da entrada através de uma pré-clusterização baseada no critério do vizinho mais próximo. Nosso propósito é abreviar o tempo de processamento da etapa de busca local através da construção de uma solução inicial otimizada, onde informações como densidade do cluster sejam consideradas na ocasião de sua geração. Basicamente, a redução do tamanho da entrada é realizada agrupando-se em um mesmo cluster os pontos considerados adjacentes segundo uma medida de vizinhança parametrizada. Isto gera um conjunto de clusters iniciais Y, cuja cardinalidade é bem inferior à cardinalidade do conjunto de entrada S ($|Y| \ll |S|$). Assim, considere cada ponto x_i como um objeto caracterizado por p atributos. Desta forma, cada ponto do conjunto S é uma tupla $x_i = (a_{i1}, a_{i2}, \ldots, a_{ip})$. Para reduzir a cardinalidade de S, devemos encontrar, para cada ponto x_i, a distância euclidiana entre x_i e seu vizinho mais próximo, dada por $d_{vp}(x_i) = \min \|x_j - x_i\| \ \forall x_j \in S$ e $j \neq i$, onde $\|x_j - x_i\| = \left(\sum_{q=1}^{p}(a_{jq} - a_{iq})^2\right)^{1/2}$. Em seguida, calculamos a média destas distâncias, dada por $d_m = \frac{1}{n}\sum_{i=1}^{n} d_{vp}(x_i)$. Para parametrizarmos a visão de vizinhança do algoritmo, fazemos $\delta = u \ldots d_m$, onde u é um parâmetro de entrada. Agora, tomamos os N pontos de S como vértices de um grafo e calculamos sua matriz de adjacência dada pela equação 2.1, onde $1 \leq j \leq i \leq N$, determinando os componentes conexas deste grafo.

$$\mathbf{A_{(i,j)}} = \begin{cases} 1 & \text{se } \|x_i - x_j\| \leq \delta \\ 0 & \text{caso contrário} \end{cases} \tag{2.1}$$

Considere os componentes conexas encontrados $G_1, G_2, \ldots, G_m$ como sendo clusters iniciais e V_i, para $i = 1, \ldots, m$, o centróide do cluster G_i. Pode-se facilmente verificar que o parâmetro u da função δ funciona como um redutor de cardinalidade de S, pois se tomarmos u com valores muito pequenos, o valor de m (número de componentes) tende a N e se tomarmos u com valores muito grandes, m tende a 1.

Até aqui, mostramos como se processa a pré-clusterização. Para a clusterização ser concluída, considere um vetor binário $r = (r_1, r_2, \ldots, r_m)$, onde cada r_i pode assumir valor 0 ou 1, representando uma solução semente. Se o i-ésimo elemento de r contiver o valor 1, a componente conexo G_i fará parte da solução como cluster pai (cluster raiz). Caso contrário, G_i será anexado a um dos clusters pais que constam

na solução inicial. Como exemplo, da semente $r = (01010001)$ de uma entrada que gerou 8 componentes conexas, podemos dizer que a solução terá três clusters, que iniciarão a partir de G_2, G_4, G_8 (clusters pais) e que G_1, G_3, G_5, G_6, G_7 são os clusters filhos que, no processo de clusterização, serão anexados a um dos três clusters pais para formar com este um novo cluster, como veremos a seguir.

Na geração da semente, procuramos priorizar os componentes conexas mais densos. Para isso, a probabilidade de ocorrência do valor 1 em cada elemento r_i de r é proporcional à cardinalidade do i-ésima componente conexo. Este método funciona como o método de seleção denominado *método da roleta* usado em Algorimos Genéticos [2], onde a seleção do indivíduo para cruzamento é tão provável quanto melhor for a sua qualidade. A possibilidade de tendermos a um ótimo local com esta estratégia torna-se irrelevante, uma vez que a fase de busca local do nosso algoritmo consegue cobrir o espaço de busca eficientemente.

Após a geração da semente, faça $Y = \{Y_1, Y_2, \cdots, Y_t\}$ o conjunto formado pelos componentes conexas $G_{i'}s$ associados às posições com 1 na semente. Os clusters iniciais $C_{i'}s$ são os próprios $Y_{i'}s$ e os centros iniciais $H_{i'}s$ dos clusters da solução são inicialmente os centros $V_{i'}s$ destes clusters iniciais, onde $|C_i| = |Y_i|$ para $i = 1, 2, \ldots, t$ e $|Y_i|$ denota o número de pontos pertencentes a Y_i. No processo de clusterização, os componentes $G_{i'}s$ em $\{G_1, G_2, \ldots, G_m\} - Y$ são analisados um de cada vez da seguinte forma: $G_i \subset C_j$ se $\|V_i - H_j| \leq |V_i - H_q|$ para $1 \leq q \leq t$ e $q \neq j$. Quando algum G_i é associado a um cluster C_j, a cardinalidade e o centróide do cluster C_j são recalculados pelas Eqs. (2.2) e (2.3).

$$H_j^{'} = \frac{H_k * |C_j| + V_i * |G_i|}{|C_j| + |G_i|} \tag{2.2}$$

$$|C_j^{'}| = |C_j| + |G_i| \tag{2.3}$$

A idéia de usarmos probabilidade na construção da semente se justifica no fato de que clusters com maior cardinalidade são mais representativos do que clusters menos densos, e embora isto não se verifique para algumas instâncias, a fase de busca local do nosso algoritmo avalia as regiões do espaço de busca que porventura tenham sido excluídas durante a fase de construção.

3. A Busca Local RandomSearchDual

Propomos neste trabalho um algoritmo de busca local randomizada baseado em sobreposição e inversão de janelas denominado RandomSearchDual. Para melhor entender o algoritmo proposto, vamos definir o que chamamos de janela. Seja r um vetor binário de tamanho $|r|$. Uma janela $s_{p,q}$ de r, com $p, q \in [1, |r|]$ e $p < q$ é a sub-cadeia de r com tamanho $(q - p) + 1$ cujos elementos são os mesmos elementos da cadeia r com índices dentro do intervalo $[p, q]$. Assim, considere uma semente r gerada no algoritmo de construção, onde $|r| = m$ com m igual ao número de componentes conexas encontrados na pré-clusterização. Nós propomos uma estratégia que consiste na aplicação de três procedimentos de busca local aleatória, onde a

alternância de um para outro se dá sobre duas condições opostas: quando a melhor solução gerada até o momento deixa de ser atualizada após um número x de iterações sem ganho na função de custo ou quando um número y de iterações em que houve ganho na qualidade da solução seja obtido.

O primeiro procedimento de busca, chamado *RandomSearch*, é baseado na sobreposição de janelas, onde dois limitantes p e q, gerados randomicamente, determinam a *sub-string* $s_{p,q}$ a ser usada na sobreposição. A partir de um ponto t escolhido aleatoriamente, todos os $(q - p) + 1$ elementos da semente r são substituídos pelos elementos da janela $s_{p,q}$, gerando uma nova semente r'. Caso r' apresente melhor qualidade que r, r' passa a ser a solução atual. Há dois critérios de parada da busca: quando um número atualizações de r previamente fixado seja atingido (Nreloaded); ou quando o número de iterações sem atualização de r atingir um teto estabelecido (NmaxNoReloaded). Ou seja, o procedimento pode ser interropido mesmo que esteja obtendo sucesso na atualização da solução.

O uso desta estrutura de busca isoladamente, embora possa levar à boas soluções, para aquelas instâncias onde o número de componentes conexas geradas na préclusterização é muito elevado em comparação com o número ideal de clusters, a mesma demonstrou-se muito lenta. Para suprir esta deficiência, foi proposta uma outra estrutura de busca local chamada *RandomDual*, que é baseada na inversão de janelas, e é inicializada quando o primeiro critério de parada é aceito, ou seja, quando o número de atualizações da solução através da busca *RandomSearch* for obtido. Este segundo método de busca consiste em tomar uma janela da semente atual e inverter todos os elementos desta janela, fazendo com que a posição que contém 0 passe a ter 1 e vice-versa. Assim como no primeiro procedimento, a nova semente é avaliada e substituirá a atual caso a sua qualidade seja superior. Os critérios de parada são os mesmos da estrutura anterior, proporcionando a alternância das duas buscas.

```
1.  Procedimento RandomSearchDual(r, N_reloaded, NmaxNoReloaded)
2.      repita
3.          best ← (Random_Dual(r));
4.          se (N_sucesso = N_reloaded) entao
5.              best ← (Dual(r));
6.              best ← (Random_Search(r));
7.              best ← (Dual(r));
8.              N_sucesso ← (N_sucesso + 1);
9.          senão
10.                 N_insucesso ← (N_sucesso + 1);
11.     até N_insucesso = N_reloaded;
12. fim Algoritmo.
```

Figura 1: Procedimento de busca local *RandomSearchDual* ()

Foi observado que nos casos em que a fase de construção não gera uma boa solução inicial, a convergência usando apenas os dois métodos já citados é lenta e depende da inversão de uma janela igual à própria *string*, o que somente é possível pelo segundo método com $p = 1$ e $q = m$. Como forma de resolver esta limitação, uma terceira estrutura de busca, chamada *Dual*, foi proposta. A mesma consiste em aplicar o operador binário *not* a todos os elementos da semente. Tal operação é chamada cada vez que um dos dois métodos anteriores é acionado, isto se a qualidade de r for negativa. Desta forma, abrevia-se a convergência para os casos onde a fase de construção não é eficiente.

Existem dois critérios de parada para o algoritmo **RandomSearchDual**, formado pelos três procedimentos de busca, o primeiro é o número máximo de iterações e o segundo é o número máximo de insucessos dos três métodos juntos. O algoritmo *RandomSearchDual* é apresentado na Figura 1.

Para verificar o efeito dos procedimentos de busca *Dual*, *RandomSearch* e *RandomDual* sobre uma semente de entrada r, veja o exemplo a seguir, onde r é a solução no momento da chamada de cada um dos três procedimentos. Para $r =$ [0001011110], $m = 10$, suponha $p = 2$, $q = 5$ e $t = 6$ obtidos randomicamente dentro do intervalo $[1, 10]$. Desta forma, a janela $s_{2,5}$ é [0010] e a semente r' resultante do procedimento *Dual* (r) é [**1110100001**]. A solução r' resultante de *RandomSearch* (r) é [00010**0010**0], ao passo que a solução resultante do proceimento *RandomDual* (r) é [0**1101**11110]. Os elementos em negrito indicam aqueles segmentos de r que sofreram alteração durante a aplicação do procedimento. A semente é tratada como uma lista circular, permitindo que, mesmo no caso de $(q - p) + t > m$, onde t é o ponto escolhido dentro do intervalo $[1, m]$, o número de elementos que podem sofrer alterações pelo procedimento *RandomSearch* é sempre igual ao tamanho da janela formada por p e q. Isto permite que a semente sofra mudanças simultâneas nas duas extremidades. Vê-se no exemplo anterior que o número de 1's da semente pode permanecer inalterado, aumentar ou diminuir. Isto amplia a exploração do espaço de busca e facilita a obtenção do número adequado de clusters, já que o número de 1's na semente equivale ao número k de clusters gerados na clusterização.

4. A Heurística de Busca da Melhor Clusterização

Nesta seção, apresentamos a heurística usada em [9] aplicada ao nosso algoritmo de construção e busca local *RandomSearchDual*. Antes, porém, vamos mostrar como se dá a avaliação da qualidade da solução, representada por uma semente r através da sua função de aptidão $f(r)$. No cálculo de f, duas equações são definidas. Uma avalia a distância entre os centróides dos clusters filhos pertencentes a um mesmo cluster raiz: D_{intra}; e a outra avalia a menor distância entre os centróides de todos os clusters raiz: D_{inter}, conforme as Eqs. 4.1 e 4.2. A Eq. 4.1 dá uma medida de coesão do cluster. Já a Eq. 4.2 dá a medida de dispersão dos clusters.

$$D_{intra}(C_i) = \sum_{G_k \subset C_i} |V_k - H_i| * |G_k| \tag{4.1}$$

$$D_{inter}(C_i) = \sum_{G_k \subset C_i} (\min_{j \neq i} |V_k - H_j|) * |G_k| \tag{4.2}$$

Agora, podemos definir a função de aptidão de uma *string* r como sendo:

$$f(r) = \sum_{i=1}^{k} D_{inter}(C_i) * w - D_{intra}(C_i)$$

onde o peso w é um parâmetro de entrada e k é o número de clusters gerados, sendo que durante a execução do algoritmo, o valor de $f(r)$ deve ser maximizado.

No cálculo da função de aptidão de uma semente, as equações 4.1 e 4.2 são relacionas de maneira que o uso de um peso w, passado como parâmetro, possibilite priorizar D_{intra} ou D_{inter} e assim, o número de clusters fica condicionado ao peso w, já que seu valor pode levar a uma solução com muitos clusters compactos (w com valor pequeno), ou uma solução com poucos clusters com uma uma configuração mais dispersa (w com valor grande).

De posse do valor de $f(r)$, devemos agora determinar a melhor clusterização segundo o valor do peso w. Para isso, utilizaremos uma heurística que consiste em um algoritmo de busca binária proposto em [9]. Antes de definirmos a heurística, consideremos como resultado da clusterização os k clusters $\{C_1, C_2, \ldots, C_k\}$. As equações 4.3 e 4.4 são definidas sobre a clusterização gerada, como segue:

$$D_a(w) = \min_{1 \leq i \leq j \leq k} |H_i - H_j| \tag{4.3}$$

$$D_b(w) = \max_{1 \leq i \leq k} \max_{G_r \subset C_i} \frac{\sum_{x_j \in C_i} \frac{|x_j - H_i|}{|C_i|}}{\sum_{x_j \in G_r} \frac{|x_j - V_r|}{|G_r|}} \tag{4.4}$$

onde w é o valor do parâmetro w usado na função de aptidão.

A equação 4.3 calcula a menor distância entre os centros dos clusters da solução, e a equação 4.4 calcula a maior razão entre o raio médio de um cluster raiz e o raio médio dos seus clusters filhos. Assim, D_a revela a proximidade dos k clusters encontrados e D_b a coesão dos mesmos.

Na heurística para encontrar a melhor clusterização (**GetClustering**), o algoritmo *RandomSearchDual* é usado para valores de w dentro do intervalo $[w_1, w_2]$ através de uma busca binária cujo critério de parada é que a diferença entre w's do intervalo seja menor que um limiar λ ($(w_2 - w_1) < \lambda$). Desta forma, chama-se o *RandomSearchDual* usando como valor do parâmetro w os extremos do intervalo (w_1 e w_2) e o ponto médio $w_m = (\frac{w_1 + w_2}{2})$. De acordo com os valores calculados pelas Eqs. 4.3 e 4.4, w_m passará a ser o limite inferior ou superior do intervalo. Ao final do procedimento de busca binária, teremos como resultado, a clusterização obtida usando o melhor extremo do intervalo segundo o esquema 2 descrito a seguir.

```
1. Procedimento GetClustering ()
2.     enquanto (w_L − w_S) > λ faça
3.         Executa_Busca_RandomSearchDual(w_S, w_m, w_L);
4.         encontre o sub-intervalo [w_a, w_b] onde D_a(w_b)/D_a(w_a) seja máximo;
5.         faça w_L = w_b; w' = w_L;
6.         enquanto (w_L − w_S) > λ faça
7.             Executa_Busca_RandomSearchDual(w_S, w_m, w_L);
8             encontre o sub-intervalo [w_a, w_b] onde D_b(w_b)/D_b(w_a) seja máximo;
9.             faça w_L = w_b
10.        w'' = w_L; w''' = w_S;
11.        Se w'' ≥ w' então retorne solução dada por w'
12.        Se w'' < w' então retorne solução dada por w'''
13. fim Algoritmo.
```

Figura 2: Algoritmo do cálculo da melhor clusterização

Testes	AG	RS	ta	tr	na	nr
300.4	20866	**31928**	544	**32**	2	4
350.5	23133	**63733**	1202	**71**	3	5
450.4	218214	**315430**	4155	**69**	3	4
500.3	169389	**266783**	4297	**164**	2	3
600.3	65461	**247370**	3155	**193**	12	3
700.4	87135	**306677**	3188	**135**	6	4
900.5	196683	**325437**	5915	**198**	8	5
1000.6	101497	**501580**	22772	**639**	15	6
1500.6	218809	**440604**	39782	**900**	19	6
2000.11	258780	**977507**	43845	**863**	68	11

Tabela 1: Desempenho médio dos algoritmos Clustering e RandomSearchDual. (AG): valor da solução usando o AG *Clustering*; (RS): valor da solução usando *RandomSearchDual*; (ta): tempo em segundos do AG; (tr): tempo em segundos do *RandomSearchDual*; (na): número de clusters encontrados pelo AG; (nr): número de clusters encontrados pelo *RandomSearchDual*.

5. Resultados Computacionais

De nosso conhecimento, não existe nenhuma biblioteca com instâncias para o Problema de Clusterização com número variável de clusters. Desta forma, para avaliar o algoritmo RandomSearchDual, foi gerado aleatoriamente um conjunto de problemas testes, cujas dimensões da base de dados variam de 300 a 2000 pontos no espaço R^2. Além do algoritmo proposto, implementamos também o AG de [9].

Devido às características das intâncias geradas, onde os clusters estão bem definidos, através de uma análise visual temos condições de saber se a solução ótima foi ou não encontrada, como pode ser observado nos resultados apresentados na Figura 3. Cada um dos 10 testes foi executado 10 vezes pelos dois algoritmos e os resultados médios são ilustrados na Tabela 1. Note que o algoritmo proposto sempre obtém uma melhor solução, como pode ser visto comparando-se as colunas **AG** e **RS**, que contém o valor da solução do AG da literatura e do nosso algoritmo respectivamente, sendo que, quanto maior o valor, melhor a qualidade da solução

encontrada. Na primeira coluna, o número à esquerda do ponto representa o número de objetos da instância e o número à direita representa a quantidade de clusters em uma solução ótima. As colunas *na* e *nr* apresentam respectivamente o número de clusters encontrados pelo AG da literatura e pelo algoritmo proposto. As colunas *ta* e *tr* da Tabela 1 mostram os respectivos tempos de execução (em segundos) do AG e do algoritmo proposto. Note que os tempos de execução do algoritmo proposto sempre foram bem menores que os do AG da literatura.

Em todos os testes realizados, o algoritmo proposto encontrou solução ótima. Devemos esclarecer que o fato de afirmarmos que a solução ótima foi ou não atingida se deve a uma análise visual de cada problema teste, como ilustrado pela Figura 3, onde percebemos que no caso da solução gerada pelo AG [9] (Figura 3 (b))encontramos num mesmo cluster, diferentes tipos de objetos (elementos), enquanto na solução do nosso algoritmo (Figura 3 (a)), cada cluster é formado por um único tipo de objeto (elemento). Além disso, nos testes realizados, todos os parâmetros comuns tiveram seus valores mantidos para os dois algoritmos.

6. Conclusões

Neste trabalho, verificamos que o uso de uma estrutura de busca local eficiente com critérios randômicos para seleção dos itens a serem alterados, amplia consideravelmente a eficácia do processo de busca de um algoritmo. Além disso, mostramos que uma única estrutura de busca não é interessante, tendo em vista o uso de bases de dados de natureza diversa. Assim, à medida que a base de dados cresce, o desempenho do AG apresentado em [9] cai consideravelmente, ao passo que o nosso algoritmo além de continuar apresentando soluções de alta qualidade, mantém o bom desempenho para diferentes classes de bases de dados, embora usando funções de aptidão idênticas e a mesma heurística para o cálculo do peso w.

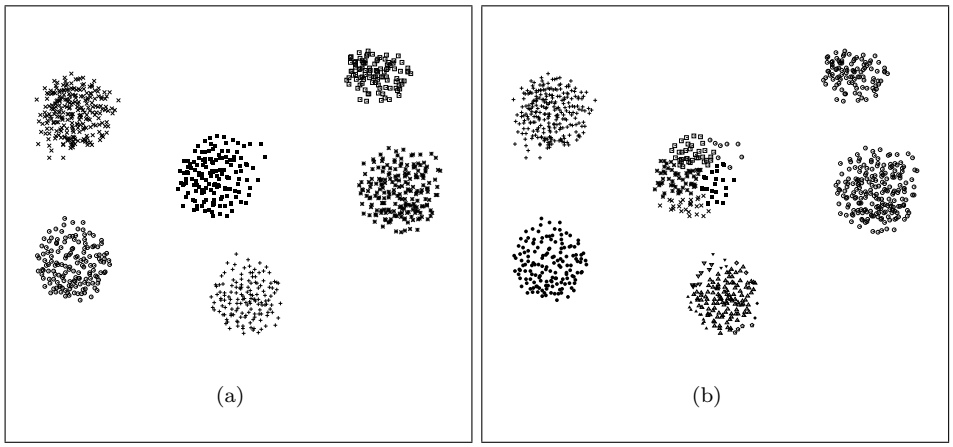

Figura 3: Resultados dos dois algoritmos para a instância 1000.6: (a) Clusterização com RandomSearch (b)Clusterização com CLUSTERING.

Desta forma, podemos dizer que o ganho obtido em termos de qualidade da solução e tempo de processamento deve-se às estratégias usadas na fase de construção, aos algoritmos de busca local e a como estes algoritmos trocam informações sobre a solução atual durante o processamento da clusterização, o que sugere uma eficiência que pode ser aplicada a outros algoritmos de clusterização de outras metaheurísticas que tenham uma representação da solução através de seqüência binária.

Abstract. The Clustering Problem of a database, even so already has been sufficiently explored for researchers of areas as mathematical, statistics and computation, in the majority of the presented works has an approach of the case where the number of clusters is previously fixed by the user as a parameter of entrance of the algorithm. However, in many practical applications the number of clusters is a variable that must be determined by the algorithm. In this work we present a construction and local search algorithm place through overlapping and inversion of windows for this class of the problem (automatic clustering) and we show its efficiency when comparated with a Genetic Algorithm that until then presented the best ones resulted for this type of problem.

Referências

[1] B. Sanghamitra, M. Ujjwal, An evolutionary technique based on K-Means algorithm for optimal clustering, *Information Sciences*,**146** (2002), 221-237.

[2] C. R. Dias, L. S. Ochi, Efficient evolutionary algorithms for the clustering problem in directed graphs, To appear in Proc. of the 2003 Congress on Evolutionary Computation (IEEE-CEC), Canberra - Austrália.

[3] F. Glover, Tabu Search - Part I, *ORSA Journal on Computer*,**1**,1989.

[4] F. Hichem, K. Raghu, A robust algorithm for automatic extraction of an unknown number of clusters from noisy data, *Pattern Recog. Letters*, **17** (1996).

[5] G. Karypis, E. Han e V. Kumar, CHAMELEON: A hierarchical clustering algorithm using dynamic modeling, *COMPUTER*, **32** (1998), 68-75.

[6] J. H. Holland, Adaptation in Nature and Artificial Systems, *University of Michigam Press - MI*, 1975.

[7] L. S. Ochi, M. J. F. Souza, N. Maculan, A GRASP - TABU SEARCH algorithm to solve a School Timetabling Problem, To appear in *Combinatorial Optimization Book Series, Metaheuristics: Computer Decision* - Making, vol. 15 (2003), chapter 31, pp: 659-672, editors: D. Z. Du and P. M. Pardalos (serie editors). KLUWER Academic Publishers.

[8] T. A. Feo, M. G. C. Resende, Greedy Randomized Adaptative Search Procedures, *Journal of Global Optmization*, **6** 1995,109-133.

[9] Y. T. Lin, Y. B. Shiueng, A genetic approach to the automatic clustering problem,*Pattern Recognition*, **34** (2001), 415-424.