10th Brazilian Congress on Computational Intelligence (CBIC'2011), November 8 to 11, 2011, Fortaleza, Ceará Brazil

O PROBLEMA DE CLUSTERIZAÇÃO AUTOMÁTICA: UM NOVO MÉTODO UTILIZANDO O ILS

Marcelo Dib Cruz[1], Luiz Satoru Ochi[2]

[1]Departamento de Matemática – Universidade Federal Rural do Rio de Janeiro (UFRRJ)
BR.465, km7 – 23.890-000 – Seropédica – RJ – Brasil
[2] Instituto de Computação – Universidade Federal Fluminense (UFF)
Caixa Postal 15.064 – 91.501-970 – Niteroi – RJ – Brasil
{dib@ufrrj.br, satoru@ic.uff.br}

Resumo – Clusterização é o processo de alocar os elementos de uma base de dados em um conjunto de clusters, de modo que os elementos similares permaneçam no mesmo cluster e elementos não similares sejam alocados para clusters distintos. Nos algoritmos de clusterização, normalmente é assumido que o número de clusters é um dado de entrada. Contudo, em muitas aplicações, este número ideal de clusters não pode ser determinado ou estimado previamente. Estes problemas são conhecidos como Problemas de Clusterização Automática (PCA). Neste trabalho é apresentado um Algoritmo utilizando a metaheurística ILS para a solução do PCA, incluindo três módulos de busca local e memória adaptativa. Resultados computacionais realizados para um conjunto de instâncias mostram a eficiência e a robustez da heurística proposta.

Palavra Chave. Clusterização Automática, ILS, Memória Adaptativa, Mineração de Dados

Abstract – Clustering is the process by which elements of a database are assigned for clusters of similar elements. In clustering algorithms, it is usually assumed that the number of clusters is known or given. Unfortunately, the optimal number of clusters is unknown for many application of this problem. These problems are known as Automatic Clustering Problems (ACP). In this work we present a ILS Algorithm for ACP including tree local search and adaptive memory programming procedures. Computational results on a set of instances illustrate the effectiveness and the robustness of the proposed heuristic.

Keywords. Automatic Clustering, ILS, Adaptative Memory, Data Mining

1 Introdução

Clusterização é o termo genérico para um processo que une objetos similares de uma base de dados em um mesmo grupo. Cada grupo é denominado um cluster. O número de clusters pode ser conhecido a priori ou não. Quando o número de clusters é conhecido a priori, ele é conhecido como *Problema de K-Clusterização* ou simplesmente *Problema de Clusterização* (PC). Caso contrário, o problema é denominado *Problema de Clusterização Automática* (PCA). Ambos os problemas são classificados como NP-Hard [22], mas o fato do valor de *k* não ser previamente conhecido torna o problema ainda mais complexo, aumentando substancialmente o número de soluções possíveis.

Devido a elevada complexidade do PCA, a maioria dos trabalhos encontrados na literatura utiliza técnicas heurísticas para a sua solução aproximada. Contudo, observamos que a maioria das contribuições para o problema ou apresentam heurísticas simples de construção e/ou busca local ou metaheuristicas em sua forma básica [6] [16] [18]. Neste contexto, acreditamos haver a necessidade de explorar melhor o potencial das metaheurísticas, procurando desenvolver versões mais eficientes de algoritmos heurísticos [2] [14] [15].

O objetivo deste trabalho é propor uma heurística para o PCA, utilizando a metaheurística ILS (Iterated Local Search), denominada *ILS_PCA*. Neste contexto, é efetuado um processamento inicial para tentar reduzir as dimensões dos dados de entrada do problema, através de um algoritmo construtivo. Após esta primeira fase, são utilizadas buscas locais, cuja função é refinar as soluções encontradas pelo algoritmo construtivo e pelas perturbações das soluções. O algoritmo proposto utiliza adicionalmente conceitos de memória adaptativa, na estrutura de um conjunto de soluções denominado CE, cuja função, é armazenar as melhores soluções distintas geradas até o momento pelo ILS_PCA. O algoritmo utiliza uma perturbação que varia o número de clusters, com o objetivo de encontrar a melhor configuração possível para a solução.

Este trabalho está organizado da seguinte forma: Na seção 2, é descrito o PCA acompanhado de uma revisão parcial dos trabalhos da literatura; na seção 3 é descrito o algoritmo proposto; na seção 4 são mostrados os resultados e análise dos testes computacionais realizados e finalmente, na seção 5, são apresentadas as conclusões.

2 O Problema de Clusterizacao Automática

Formalmente podemos definir o problema de clusterização automática da seguinte forma: dado X um conjunto de n objetos $X=\{x_1, x_2, x_3...., x_n\}$, onde cada objeto x_i é uma tupla $(x_{i1},x_{i2},...,x_{ip})$, e cada coordenada x_{ij} está relacionada com um atributo j do objeto i. Cada objeto pode ser considerado um ponto no espaço R^p. Como objetivo, devemos encontrar o conjunto $C=\{C_1,C_2,...,C_k\}$ de clusters, onde k não conhecido previamente, onde a similaridade entre os objetos de um mesmo cluster seja maximizada e a similaridade entre objetos de diferentes clusters seja minimizada, sujeito as seguintes condições:

$$C_i \neq \phi \text{, } para \quad i = 1,..,k \tag{1}$$

$$C_i \cap C_j = \phi \text{, } para\, i, j = 1,..,k\, e\, i \neq j \tag{2}$$

$$\bigcup_{i=1}^{k} C_i = X \tag{3}$$

Outra definição necessária é o conceito de centróide de um cluster C_i. A substituição de um conjunto C_i com t pontos similares por um único ponto $v_m = (v_{m1}, v_{m2},....v_{mp})$ que os represente, pode ser feito considerando-se v_m o centróide de C_i , onde cada coordenada de v_m é dada pela equação:

$$v_{mi} = \frac{1}{t}\sum_{j=1}^{t} x_{mj} \quad , \quad i = 1,..,p \tag{4}$$

2.1 Trabalhos Relacionados

O tema clusterização é antigo e já muito estudado. Os primeiros trabalhos datam da década de 60. Porém, a maioria dos trabalhos é para o PC, onde o número de clusters é definido previamente. Um dos métodos mais conhecidos para o PC é o Método *K-Means* [10], que utiliza o conceito de centróide (equação 4) para representar os clusters.

No entanto, o PCA ainda não é tão explorado como o PC. O X-means [12] adapta o k-means para o PCA. O trabalho de [19] é mais recente e também adapta o *k-Means* para resolver o PCA.

Alguns trabalhos encontrados na literatura utilizam algoritmos evolutivos como em [16] e [18].

Os métodos propostos em [3] [4] consistem em dividir o espaço (base de dados) que contem os objetos em um determinado numero de subespaços ou células. As células contendo um número grande de objetos são candidatos a formarem um cluster. O resultado do método depende do tamanho das células, que normalmente é um parâmetro de entrada.

Um método apresentado por [20] utiliza o conceito de árvore geradora para obter uma árvore inicial e depois fazer cortes, gerando sub-árvores, que formarão os possíveis clusters.

Em [1] [21] são apresentados métodos baseados em funções que definem densidade e conectividade. Eles são baseados na idéia que objetos que formam uma região densa podem ser agrupados previamente num mesmo cluster.

3 O Algoritmo ILS Proposto

O *Algoritmo ILS para o PCA (ILS_PCA)*, aqui proposto, é um método composto por quatro componentes principais: a Fase de Construção, as Buscas Locais, a Perturbação e o Critério de Aceitação.

3.1 A Fase de Construção

A Fase de construção do ILS_PCA é uma etapa inicial que tem por objetivos tentar reduzir a cardinalidade dos dados de entrada do problema, bem como gerar uma solução inicial de boa qualidade. A idéia é substituir grupos de objetos da base de dados cuja similaridade é considerada alta, por um único objeto *(cluster inicial*) que represente o grupo.

Neste trabalho usamos como medida de similaridade a distancia euclidiana entre dois pontos. A redução do tamanho da entrada (pré-processamento) é realizada agrupando-se em um mesmo cluster os pontos pertencentes a uma região densa, como mostra a figura 1. Inicialmente, nas linhas 2, 3 e 4, para cada ponto é definido a menor distância a outro ponto qualquer. Depois, na linha 5, é calculada a média destas distâncias, denominada d_{medio} . Então, cada ponto $x_i \in X$ é considerado o centro de um círculo cujo valor do raio é $r = u * d_{medio}$, onde u é um parâmetro de entrada. Logo após, na linha 8, é calculado o conjunto de pontos contidos no círculo de centro x_i e raio r ($N_i = circulo\ (x_i,r)$). Estes valores são colocados em uma lista T que é ordenada em ordem decrescente. Os elemento de T são considerados os clusters parciais $B = \{B_1,B_2,...B_t\}$. Para que os clusters não possuam elementos em comum, toda vez que um círculo é selecionado, todos os pontos do seu interior não podem mais entrar em outro círculo. Com este procedimento as regiões mais densas são selecionadas.

Após este procedimento inicial, é realizado um refinamento que tem como objetivo central diminuir o número de clusters parciais. Assim, é efetuada uma agregação de clusters parciais de menor cardinalidade ($\leq min$, onde min é um dado de entrada), que estejam bem próximos a algum outro cluster de maior cardinalidade. Isto é realizado verificando se este cluster menor está a uma distancia d_{adj} de outro cluster, onde $d_{adj} = v * d_{medio}$ (onde o valor v é um dado de entrada).

1. ***Procedimento Construtivo (X, u)***
2. ***PARA*** *i = 1* ***até*** *n* ***FAÇA***
3. $d_{\min}(x_i) = \min \| x_i - x_j \|, i \neq j\ , j = 1,...n$
4. ***FIM PARA***

5. $d_{medio} = \frac{1}{n}\sum_{i=1}^{n} d_{\min}(x_i)$
6. $r = u * d_{medio}$
7. **PARA** $i = 1$ **até** n **FAÇA**
8. N_i = circulo (x_i,r)
9. $T = T \bigcup N_i$
10. **FIM PARA**
11. ordenar T em ordem decrescente
12. $i = 1$
13. **ENQUANTO** ($T \neq \emptyset$) **FAÇA**
14. B_i = próximo($N_j \in T$)
15. $T = T - \{N_j\}$
16. $i = i + 1$
17. **FIM ENQUANTO**
18. retornar $B = \{B_1, B_2, ...B_t\}$, os clusters parciais.
19. **FIM construtivo**

Figura 1: Descrição do procedimento construtivo

Os clusters gerados no procedimento construtivo são utilizados nesta fase da seguinte forma, como mostra a figura 2. Para cada cluster parcial *de menor cardinalidade*, verifique qual o cluster que está mais próximo em relação à distância entre os centróides (linhas 4, 5 e 6). Se este cluster possuir cardinalidade maior que *min* pontos e possuir pelo menos um ponto a uma distancia de d_{adj} de qualquer ponto do outro cluster, então ele será incorporado ao outro, como mostram as linhas 7 e 8. Denominamos este procedimento de *Junção de Clusters Iniciais Adjacentes (JCIA)* e tem por objetivo incorporar clusters de menor cardinalidade a clusters de maior cardinalidade. Este procedimento pode reduzir o número de clusters remanescentes da primeira fase. Os clusters gerados após o Procedimento Construtivo e o JCIA são denominados clusters iniciais.

1. **Procedimento** JCIA **(X, u, v, d_{medio}, min)**
2. B = Procedimento construtivo(X,u)
3. $d_{adj} = v * d_{medio}$
4. **PARA** $i = 1$ **até** t **FAÇA**
5. **SE** (cardinalidade (B_i) <= min) **ENTÃO**
6. B_k = menor_distancia_centroide(B_i)
7. **SE** ($\exists x_r \in B_i \; e \; \exists x_s \in B_k \; tal \; que \; || x_r - x_s || < d_{adj}$)
8. **ENTÃO**
9. $B_k = B_k \; U \; B_i$
10. **FIM SE**
11. **FIM SE**
12. **FIM PARA**
13. retornar $B = \{B_1, B_2, ...B_m\}$ os clusters iniciais, onde $m <= t$
14. **FIM JCIA**

Figura 2: Descrição do procedimento JCIA

Considere os clusters iniciais gerados como $B = \{B_1, B_2, ..., B_m\}$ e seja v_i , $i = 1,2,...,m$ o centróide (equação 4) de cada cluster B_i. Para representar uma solução é utilizada uma cadeia binária de m posições. Por exemplo, se $m = 7$, então uma cadeia binária poderia ser $\{0110010\}$. Se o valor correspondente ao B_i *na cadeia binária for igual a 1*, isso significa que o cluster inicial B_i faz parte da solução como cluster pai. Se o valor correspondente ao B_i *na cadeia binária for igual a 0*, B_i é considerado um cluster filho. Os clusters filhos são unidos aos clusters pais utilizando o critério de menor distância entre os centróides. A cada união, o valor do centróide do cluster pai é recalculado. No final, todos os clusters filhos são unidos aos clusters pais para gerar uma solução completa. Portanto, nesta representação, a cada nova solução poderemos ter um número distinto de clusters pais, que não se alteram depois deste processo. Os clusters gerados após esse processo são denominados clusters finais $C = \{C_1, C_2, ..., C_k\}$. Para descrever melhor este processo, seja $B^o = \{B_1^o, B_2^o, ..., B_r^o\}$ um subconjunto de B onde seus elementos tem valor 0 na cadeia binária e $B^1 = \{B_1^1, B_2^1, ..., B_s^1\}$ um subconjunto de B onde seus elementos tem valor 1 na cadeia binária, onde $m = r + s$. Inicialmente cada elemento do conjunto B^1 é candidato único a cada cluster C_i , $i = 1,..,s$. Então, para cada elemento de B^o ($B_j^o \in B^o$, cujo centróide é v_j^o), encontrar um elemento de B^1 ($B_z^1 \in B^1$, cujo centróide é v_z^1), onde a distância entre os seus centróides seja mínima, como mostra a equação abaixo:

$$|| v_z^1 - v_j^o || = mínimo \; || v_i^1 - v_j^o || \; , \; i = 1,....,s \qquad (5)$$

A cada busca, B_z^1 é atualizado incorporando B_j^o e o centróide é recalculado. O conjunto B^o é atualizado. O processo continua até $B^o = \emptyset$.

Para avaliar a qualidade da solução encontrada, é utilizada uma função de avaliação F (equação 10), proposta por Kaufman e Rousseeum [9], conhecida como *Índice Silhueta*. Esta função admite valores dentro do intervalo [-1,1] e não é necessário definir o valor de *k*, pois o valor mais adequado de *k* é atingido ao *maximizar* esta função. Seja x_i um ponto pertencente ao cluster $C_w \in C$, com $|C_w| = M > 1$. A distância entre os objetos x_i e x_j é definido por d_{ij}. A similaridade média de x_i em relação a todos os pontos $x_j \in C_w$ é dada por $a(x_i)$ onde

$$a(x_i) = \frac{1}{M-1}\sum d_{i,j} \quad \forall x_j \neq x_i \quad , x_j \in C_w \cdot \tag{6}$$

Nos casos em que C_w possuir um único elemento, definimos $a(x_i) = 0$. Considere ainda, cada um dos clusters $C_t \in C$ com $t \neq w$ e $|C_t| = T > 1$. A similaridade média do ponto x_i em relação a todos os pontos de C_t é

$$d(x_i, C_t) = \frac{1}{T}\sum d_{i,j} \quad \forall x_j \in C_t \cdot \tag{7}$$

Seja $b(x_i)$ o menor valor dentre todos os $d(x_i,C_t)$. Então, $b(x_i) = Min\ d(x_i, C_t)$, $C_t \neq C_w$, $C_t \in C$ (8)

O valor de Silhueta do ponto $x_i \in X$ é dado por

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))} \tag{9}$$

e a função F , denominada Índice Silhueta, é definida como: $F = \frac{1}{n}\sum_{i=1}^{n} s(x_i)$ (10)

Para gerar uma solução, inicialmente é realizado um procedimento para construir uma lista LRC. Nesta etapa, várias soluções são geradas, e o número de clusters pais das melhores são armazenados na LRC.

```
1. Procedimento GSI (B ; MaxIter)
2.   Seja B= {B1,B2,...,Bm} o conjunto de clusters iniciais
3.   PARA i = 1 até MaxIter FAÇA
4.       PARA k = 2 até m-1 FAÇA
5.           s^0 = Gerar Solução (k , B)
6.           Atualizar LRC(s^0)
7.           SE (s^0 > s*) ENTÃO
8.               s* = s^0
9.           FIM SE
10..     FIM PARA
11.  FIM PARA
12.  Retornar s* e LRC
13. FIM GSI
```

Figura 3: O procedimento *GSI*

O procedimento denominado *Gerar Solução Inicial (GSI)* é mostrado na figura 3. Na linha 1, os clusters iniciais são dados e são armazenados no conjunto B. Depois é feito um processo iterativo, que a cada momento, gera uma solução s^0 com k clusters pais (s^0 = *Gerar Solução (k , B)), k=2,..,m-1*. Cada solução é avaliada e o número de clusters pais das melhores são armazenadas na lista LRC (*Atualizar LRC(s^0)).* Isto está indicado entre as linhas 4 e 6. Depois, nas linhas 7, 8 e 9, a melhor solução s^* até o momento é atualizada. Este processo iterativo é repetido por MaxIter vezes, como indica a linha 3. O retorno deste procedimento é a solução inicial s^* e a lista *LRC*. A lista LRC define o limite inferior (l_i) e o limite superior (l_s) de clusters pais que as soluções do algoritmo podem ter.

O uso de memória Adaptativa em metaheuristicas ainda tem sido pouco explorado, mas se mostrado bastante promissor como visto por Glover [7][15]. O caminho para tornar um algoritmo heurístico auto-adaptativo, pode estar relacionado ao processo de calibração de seus parâmetros e/ou uso de informações relevantes das iterações passadas num processo de busca. A proposta utilizada neste trabalho é a de fazer um bom uso da memória, através da utilização de informações contidas num *pool* das melhores soluções geradas. A memória Adaptativa utiliza um conjunto com as melhores soluções do algoritmo, que são atualizadas ao longo das iterações. Neste trabalho é utilizado um conjunto, denominado *CE* (Conjunto ELITE), que armazena a melhor solução de cada iteração do algoritmo. As soluções armazenadas são diferentes entre si. O CE é utilizado em dois momentos diferentes no algoritmo proposto: no meio do algoritmo, para efetuar a busca local Reconexão por Caminhos, que permite procurar soluções melhores através de boas soluções encontradas pelo algoritmo e

armazenadas no CE; e no final do algoritmo, para efetuar a busca local *Inversão Individual*, que tem como objetivo tentar melhorar as melhores soluções encontradas ao longo das iterações do algoritmo e armazenadas no CE.

3.1 As Buscas Locais

A primeira busca local proposta, denominada *Troca Entre Pares*, é uma busca intensiva que troca o status de dois elementos da solução com valores diferentes. Por exemplo, suponha que a solução corrente é *{10111010}*. Primeiro é trocado o primeiro e o segundo elemento da solução. Então a nova solução é *{01111010}*. Se este novo solução melhorar o valor da função, então ele será aceito e será a nova solução corrente. A próxima troca é feita entre o primeiro e o terceiro elemento da solução corrente. A busca local termina quando todas as trocas entre dois elementos com valores distintos são testadas. O objetivo da busca local *Troca Entre Pares* é investigar possíveis soluções diferentes com o mesmo número de clusters pais.

A segunda busca local utilizada, denominada Reconexão por Caminhos (RC), foi proposta inicialmente por F. Glover para as metaheuristicas *Tabu Search* e *Scatter Search* [7] e tem como objetivo, dado duas soluções extremas de boa qualidade geradas até o momento por uma heurística iterativa, encontrar soluções intermediárias de melhor qualidade.

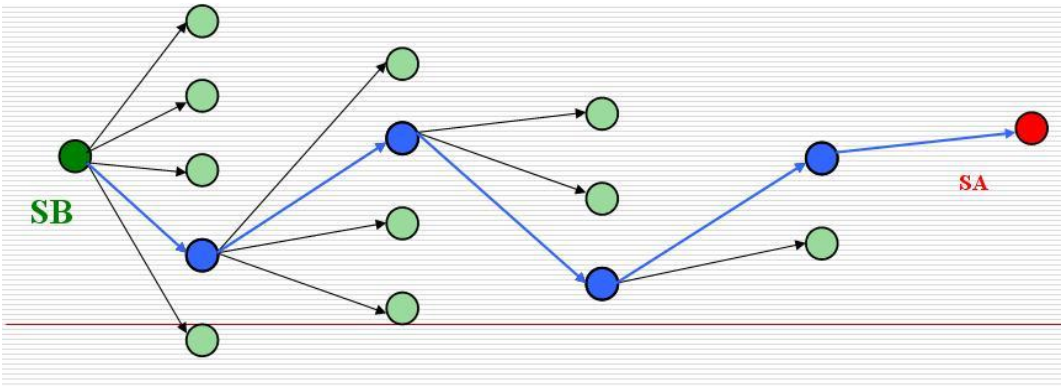

Figura 4: Descrição da RC

Na Reconexão por Caminhos utilizada neste trabalho, o sentido da trajetória adotado é o percurso do caminho partindo da solução de melhor qualidade (S_{melhor}) para a de pior qualidade(S_{pior}). Na RC, a idéia é a cada iteração deste módulo, inserir um elemento da solução alvo (SA) na solução base (SB). Neste contexto, a primeira solução intermediaria é obtida da seguinte forma: pegar o i-ésimo elemento da solução alvo e permutar pelo i-ésimo elemento associado da solução base, se estes forem distintos, repetir este procedimento para todos os elementos que compõe uma solução. A cada permutação gera-se uma nova solução candidata; avalia-se cada candidato e o melhor candidato (a que retorna a melhor solução) será escolhida como a primeira solução intermediária. Num segundo passo, essa solução selecionada passa a ser a nova solução base e o procedimento é repetido para gerar a segunda solução intermediária até que a solução base seja igual a solução alvo. A figura 4 ilustra este procedimento. O que podemos observar na RC, é que a busca é mais intensiva nas escolhas das primeiras soluções intermediárias o que justifica a escolha da busca ser sempre da melhor (S_{melhor}) para a pior solução extrema (S_{pior}). Outra justificativa para o uso da RC é pelo fato de encontrar o número ideal de clusters ser um dos objetivos do problema, e ao analisar duas soluções extremas com números de clusters pais diferentes, teremos possibilidade de obter soluções intermediárias com número de clusters diferentes das existentes nas soluções extremas.

A idéia básica desta terceira busca local proposta, denominada *Inversão Individual*, é tentar melhorar a solução corrente analisando soluções próximas a ela. Para isso, essa busca permuta o valor de cada elemento da solução (1 por 0, ou 0 por 1), um por vez, e calcula o novo valor da função. Porém, o algoritmo só aceita a mudança, se o novo valor da função for maior que o valor anterior. Por exemplo, imagine que a solução corrente é *{0101101}*. Primeiramente, é trocado o primeiro elemento da solução. Então a nova solução é *{1101101}*. Se esta solução tem valor da função maior que o anterior, então ele será a nova solução. E então, é trocado o segundo elemento. A nova solução agora é *{1001101}*. Se este possuir valor da função maior que o anterior, então a mudança é aceita. Caso contrario, a solução é mantida. A busca acaba quando todos os elementos da solução são testados. A busca local *Inversão Individual* se justifica, pois encontrar o número ideal de clusters é um dos objetivos do problema, e a inclusão ou retirada de um cluster pai pode melhorar uma determinada solução.

3.3 A Perturbação

O objetivo da Perturbação é alterar o número de clusters pais da solução corrente. Dado uma solução inicial, esta pode ter o número de clusters pais maior ou menor que o ideal. Portanto, pode ser necessário aumentar ou diminuir este número. A perturbação deve aumentar ou diminuir este valor em *uma unidade*, caso o número de clusters pais esteja bem próximo ao ideal, ou ainda, aumentar ou diminuir em *n unidades*, para procurar soluções afastadas da solução inicial. A Perturbação começa incrementando o número de clusters pais em uma unidade. Se a solução obtida após a Perturbação e a busca local for melhor que a anterior, o processo é repetido, ou seja, o número de clusters pais desta solução é incrementado em uma unidade de novo. Caso a solução obtida não melhore a anterior, então a Perturbação aumenta o número de clusters pais em duas

unidades. O processo continua enquanto a solução possuir o número de clusters pais dentro do limite superior (l_s) definido pela *LRC*. Depois, o número de clusters pais da solução é decrementado até atingir o limite inferior (l_i) da *LRC*. A Perturbação não permite que um determinado número de clusters pais seja utilizado mais de uma vez, para efetuar a busca local *Troca entre Pares*.

3.4 O Critério de Aceitação

O critério de aceitação utilizado é que a solução gerada é aceita como a solução corrente, se esta melhorar a melhor solução encontrada até o momento.

O algoritmo ILS_PCA gera uma solução inicial e aplica a busca local *Troca entre Pares*. Após cada Perturbação, o algoritmo também aplica a busca local *Troca entre Pares*. A cada t iterações, o algoritmo aplica a busca local *Reconexão por Caminhos* entre a solução corrente e a melhor solução do conjunto CE. No final, o algoritmo aplica a busca local *Inversão Individual*, no conjunto *CE* gerado.

4 Resultados Computacionais

Para analisar o desempenho do algoritmo ILS_PCA aqui proposto, testes computacionais foram realizados. O ILS_PCA foi comparado com um dos algoritmos mais recentes da literatura, denominado CLUES, que foi publicado em 2007 pela revista *Computacional Statistics & Data Analysis* [21]. O CLUES foi implementado usando o *software* estatístico R, e o seu código fonte foi disponibilizado e utilizado para executar as instâncias propostas. O código fonte do CLUES foi executado sem qualquer alteração.

O ILS_PCA foi implementado usando compilador C++ no ambiente Linux Ubuntu 7.5. Nos testes de contagem de tempo, foram utilizados computadores, para os dois algoritmos, com processadores Intel Xeon Quad Core onde cada processador tem 3.0 Ghz e com 16G de memória Ram.

Das instâncias mostradas na tabela 1, seis são instâncias conhecidas na literatura como RuspiniDataSet[13], Iris Plants Database [5], MaronnaDataSet[11], 200DATA[5], VowelDataset[8] e Broken Ring[21]. A quantidade de instâncias encontradas para o PCA é pequena e, além disso, possuem um número pequeno de pontos: *75, 150, 200, 200, 500 e 800* respectivamente. Por isso, achamos interessante e importante gerar outras instâncias com dimensões maiores. Neste contexto foram construídas instâncias de *100* a *2000* pontos no R^2 com o número de clusters variando de *2* a *26*. As instâncias foram construídas através de uma ferramenta gráfica denominada *Dots*, desenvolvida por Soares e Ochi [16]. A característica principal desta ferramenta, é possibilitar a geração também de instâncias onde a solução ótima pode ser visualizada, e portanto, onde o número ideal de clusters é pré-determinado mas obviamente não fornecido ao algoritmo. Cada instância possui o número de pontos e o número ideal de clusters (o número ideal de clusters não é repassado aos algoritmos, pois um de seus objetivos é descobrir este valor). O nome da instância mostra informações sobre a instância. Por exemplo, a instância 100p3c possui 100 pontos e 3 clusters. Quando o nome da instância termina em 1, como em 200p2c1, significa que existem muitos pontos entre os clusters e que, em alguns casos, o número ideal de clusters não está bem definido. Todas as instâncias são do espaço R^2, exceto a Iris Plants Database que é do R^4.

Tabela 1: Tabela de Comparação dos algoritmos

		ILS_PCA				CLUES			
NOME	**melhor**	**Média**	**t(s)**	**NC**	**Desvio**	**Média**	**t(s)**	**NC**	**Desvio**
RuspiniDataSet	0,7376	0,7376	0,8	4	**0,00**	0,7376	0,6	4	**0,00**
Íris Plants Database	0,6862	0,6862	3,4	3	**0,00**	0,5626	1,3	3	18,01
MaronnaDataSet	0,5745	0,5745	1,7	4	**0,00**	0,5745	4,2	4	**0,00**
200DATA	0,8231	0,8231	1,9	3	**0,00**	0,4622	2,8	3	43,85
VowelDataSet	0,4483	0,4341	87,3	23	3,17	0,4483	26,8	9	**0,00**
Broken Ring	0,4995	0,4994	17,5	5	**0,00**	0,4995	31,8	5	**0,00**
100p2c1	0,7427	0,7427	9,5	2	**0,00**	0,5213	3,7	5	29,81
100p3c	0,7858	0,7858	0,9	3	**0,00**	0,7858	1,6	3	**0,00**
100p3c1	0,5966	0,5802	9,4	3	2,75	0,5966	1,8	3	**0,00**
100p5c1	0,7034	0,6958	1,1	7	1,08	0,7034	3,1	6	**0,00**
100p7c	0,8338	0,8338	23,2	7	**0,00**	0,8338	3,7	7	**0,00**
100p7c1	0,5511	0,4835	1,1	7	12,27	0,5511	4,1	7	**0,00**
100p10c	0,8336	0,8336	20,3	10	**0,00**	0,8336	2,7	10	**0,00**
200p2c1	0,7642	0,7642	1,8	2	**0,00**	0,5912	3,1	3	22,64
200p3c1	0,6797	0,6797	1,7	3	**0,00**	0,6740	3,9	3	0,84
200p4c	0,7725	0,7725	1,7	4	**0,00**	0,7725	3,6	4	**0,00**
200p4c1	0,7544	0,7449	1,9	4	1,26	0,7544	3,4	4	**0,00**
200p7c1	0,5553	0,5394	1,5	2	2,86	0,5553	8,8	9	**0,00**
200p12c1	0,5693	0,5693	1,9	8	**0,00**	0,5624	8,5	9	1,21
300p2c1	0,7764	0,7764	4,2	2	**0,00**	0,4899	4,7	5	36,90

300p3c	0,7663	0,7663	1,9	3	**0,00**	0,5520	9,2	3	27,97
300p3c1	0,6768	0,6768	2,7	3	**0,00**	0,5924	4,4	4	12,47
300p4c1	0,5924	0,5910	2,6	2	0,24	0,5924	4,6	7	**0,00**
300p6c1	0,6636	0,6636	3,1	8	**0,00**	0,5788	11,9	7	12,78
300p13c1	0,5994	0,5441	2,8	2	9,23	0,5994	10,8	9	**0,00**
400p3c	0,7985	0,7985	5,2	3	**0,00**	0,7985	9,6	3	**0,00**
400p4c1	0,6204	0,5989	2,7	4	3,47	0,6204	12,8	4	**0,00**
400p17c1	0,5524	0,5138	8,5	2	6,99	0,5524	20,2	15	**0,00**
500p3c	0,8249	0,8249	2,3	3	**0,00**	0,8249	10,1	3	**0,00**
500p4c1	0,6597	0,6597	2,6	3	**0,00**	0,5150	9,9	3	21,93
500p6c1	0,6684	0,6287	4,2	6	5,94	0,6684	16,7	6	**0,00**
600p3c1	0,7209	0,7209	4,8	3	**0,00**	0,7028	9,6	3	2,51
600p15c	0,7812	0,7812	37,4	15	**0,00**	0,7526	37,1	16	3,66
700p4c	0,7969	0,7969	9,7	4	**0,00**	0,7969	12,9	4	**0,00**
700p15c1	0,6799	0,6799	15,1	15	**0,00**	0,6595	22,9	17	3,00
800p4c1	0,7143	0,6643	8,2	4	7,00	0,7143	15,9	4	**0,00**
800p10c1	0,5071	0,4681	10,2	2	7,69	0,5071	29,7	8	**0,00**
800p18c1	0,6941	0,6914	35,9	19	0,39	0,6941	42,2	19	**0,00**
800p23c	0,7873	0,7873	44,9	23	**0,00**	0,7387	47,7	22	6,17
900p5c	0,7160	0,7160	36,4	5	**0,00**	0,6129	24,4	7	14,40
900p12c	0,8408	0,8408	51,2	12	**0,00**	0,7975	63,1	10	5,15
1000p5c1	0,6391	0,6391	31,3	5	**0,00**	0,5580	38,0	7	12,69
1000p6c	0,7356	0,7356	28,3	6	**0,00**	0,7356	38,3	6	**0,00**
1000p14c	0,8306	0,8306	166,5	14	**0,00**	0,7674	51,6	11	7,61
1000p27c1	0,5631	0,4769	43,6	2	15,31	0,5631	67,4	14	**0,00**
1100p6c1	0,6847	0,6717	50,7	6	1,90	0,6847	39,9	6	**0,00**
1300p17c	0,8229	0,8229	120,7	17	**0,00**	0,7379	80,4	15	10,33
1500p6c	0,6941	0,6941	67,4	6	**0,00**	0,6845	73,6	5	1,38
1500p6c1	0,6597	0,6436	70,8	6	2,44	0,6597	61,7	6	**0,00**
1500p20c	0,8232	0,8232	311,2	20	**0,00**	0,6874	92,9	13	16,50
1800p22c	0,8036	0,8036	969,3	22	**0,00**	0,6433	136,1	18	19,95
2000p9c1	0,6230	0,6230	325,6	9	**0,00**	0,5354	123,2	7	14,06
2000p11c	0,7129	0,7129	438,4	11	**0,00**	0,5402	93,6	8	24,22
Media			21.72		1,58		20.42		6,98

Em relação aos parâmetros utilizados no algoritmo ILS_PCA, estes foram definidos a partir de testes preliminares. Foi utilizado o valor de u entre 1.5 e 4.5 e o valor de v igual a 2 (somente para instâncias com mais de 200 pontos. Caso contrário, o valor de v é 0). O valor *min* é igual a 4. O valor de MaxIter é igual a 5. O tamanho da LRC é de 7 elementos. O número de iterações realizadas varia entre os valores l_i e l_s da LRC. A busca local Reconexão por Caminhos é executada quatro vezes, nas iterações que correspondem a 50% , 65%, 75% e 85% do total de iterações. O tamanho do conjunto CE foi fixado com cinco elementos. Para a execução do CLUES, não é necessária a especificação de parâmetros.

A Tabela 1 mostra a comparação entre o CLUES e o ILS_PCA. Os algoritmos foram executados 5(cinco) vezes para cada instância. As colunas têm os seguintes significados: a coluna *Nome* contém os nomes das instâncias. A coluna *Melhor* contém o maior valor da função Índice Silhueta encontrado pelos algoritmos. A coluna *Média* contém a média dos valores da função Índice Silhueta que cada algoritmo encontrou. A coluna *t(s)* contém a média dos tempos de execução de cada algoritmo em segundos e a coluna *NC* contém o número de clusters que a melhor solução de cada algoritmo encontrou. A coluna *Desvio* contém o desvio da média em relação à melhor solução, conforme a seguinte definição: Desvio = (Melhor - Média) /Melhor * 100. *Os melhores resultados estão realçados em negrito.*

Na comparação do ILS_PCA com o CLUES, a diferença entre os Desvios é grande, de 1.58 para 6.98, significando que o ILS_PCA está mais próximo das melhores soluções. Além de possuir uma média melhor, o ILS_PCA também alcança os melhores valores em uma quantidade maior de instâncias. O ILS_PCA chega aos melhores valores em 36 instâncias, num total de 53. O CLUES chega aos melhores valores em 27 instâncias. Em relação ao tempo, o CLUES possui uma média um pouco melhor, com valor de 20,42s contra 21.72s do ILS_PCA. Avaliando os tempos, o que se observa é no conjunto de instâncias menores (de ruspiniDataSet até 1000p6c), o algoritmo ILS_PCA possui uma média melhor, de 12.66s contra 14,56s do CLUES. Porém, em instâncias com mais de 1500 pontos, o CLUES consegue os tempos bem menores. Em relação à quantidade de clusters (NC) encontrados por cada algoritmo, é percebido que encontrar o número ideal de clusters é a grande dificuldade dos algoritmos. Em instâncias de até 800 pontos, onde os clusters estão bem definidos (as instâncias onde os nomes terminam em c acrescidas de RuspiniDataSet, IrisPlantsDatabase, MaronnaDataSet, 200DATA, e Broken Ring) é verificado que os dois algoritmos encontram o valor ideal dos clusters na maioria dos casos. Em instâncias com mais de 800 pontos e onde os clusters estão bem definidos, somente o ILS_PCA encontrou o número ideal de clusters. Porém, onde os clusters não

estão bem definidos (as instâncias onde os nomes terminam em 1 acrescidas de VowelDataSet) é percebido que os algoritmos divergem bastante em relação ao número de clusters. Isto se deve a característica destas instâncias, onde os pontos estão muito próximos, sendo muito difícil determinar onde acaba um cluster e começa um outro. Porém o ILS_PCA consegue chegar nas melhores configurações de clusters, pois alcança os maiores valores da função Índice Silhueta. É importante ressaltar que o CLUES não precisa especificar qualquer parâmetro na sua execução, que é um diferencial em relação ao ILS_PCA.

5 Conclusão

Este trabalho aborda o PCA. Foi desenvolvido um Algoritmo ILS que utiliza várias buscas locais e Memória Adaptativa para melhorar a qualidade das soluções obtidas, após as perturbações. Constatamos que a utilização de uma Fase de Construção de boa qualidade melhora bastante o desempenho do algoritmo, uma vez que reduz o número de objetos a serem agrupados.

Acreditamos que, as buscas locais utilizadas, foram outro fator importante para o bom desempenho do ILS_PCA. Embora aumentem o tempo de processamento por iteração, verificamos que o ILS necessita de um número menor de iterações para atingir um valor alvo.

Finalmente, os resultados do ILS_PCA comparados com um algoritmo da literatura mostram a eficiência do algoritmo proposto, uma vez que conseguiu resultados melhores.

Como trabalho futuro, tentaremos implementar outras heurísticas utilizando as metaheurísticas GRASP e VNS, uma vez que a maioria dos trabalhos para o PCA que utilizam metaheurísticas, utilizam Algoritmos Genéticos ou Evolutivos.

6 Referências Bibliográficas

[1] Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P. (2005). Automatic Subspace Clustering of High Dimensional Data. Data Mining and Knowledge Discovery, 11, pp. 5–33

[2] Brito, J. A. M., Ochi, L. S., Montenegro F., Maculan, N. (2010). An ILS Approach Applied to the Optimal Stratification Problem. To appear in International Transaction in Operational Research (ITOR) - Wiley-Blackwell – (aceito em 02/2010).

[3] Derya, B., Alp, K. (2007). ST-DBSCAN: An algorithm for clustering spatial–temporal data. Data & Knowledge Engineering 60 , pp. 208-221

[4] Duan, L. Xu, L., Guo, F., Lee, J., Yan, B. (2007); A local-density based spatial clustering algorithm with noise, Information Systems 32;pp. 978-986

[5] Fisher, R. (1936). The use of multiple measurements in taxonomic problems. Annual Eugenics 7,pp. 179-188.

[6] Garai, G. , Chaudhuri, B. B.(2004). A novel genetic algorithm for automatic clustering. Pattern Recognition Letters, pp. 173-187.

[7] Glover, F.; Kochenberger, G. A. (2003) Handbook of Metaheuristics. Kluwer Academic Publishers.

[8] Hastie, t.;Tibshirani, R.; Friedman, J. (2001). The Elements of Statistical Learning. Data Mining, Inference, and prediction. Springer.

[9] Kaufman, L., Rousseeum, P. J. (1990). Finding Groups in Data : An introduction to cluster Analysis. A Wiley-Intercience publication.

[10] MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1, pp. 281-297

[11] Maronna, R.; Jacovkis, P. M. (1974). Multivariate clustering procedures with variable metrics. Biometrics30, pp. 499-505.

[12] Pelleg, D, Moore, A. (2000). X-means: Extending K-means with efficient estimation of the number of clusters. Proceeding of the 17th International Conference on Machine Learning, pp.727-734.

[13] Ruspini, E. H. (1970). Numerical methods for fuzzy clustering. Information Science.pp. pp. 319-350.

[14] Santos, H. G., Ochi, L. S., Marinho, E. H., Drummond, L. M. (2006).Combining an Evolutionary Algorithm with Data Mining to solve a Vehicle Routing Problem. NEUROCOMPUTING Journal - ELSEVIER, volume 70 (1-3), pp. 70-77

[15] Silva, A. R. V., Ochi, L. S. (2010). Hybrid Algorithms for Dynamic Resource-Constrained Project Scheduling Problem. To appear in Lecture Notes in Computer Science (LNCS) - Proc. of the 7th International Workshop on Hybrid Metaheuristics (HM2010)., Vienna

[16] Soares, S. S. R., Ochi, L. S., Drummond, L. M. (2006). Um algoritmo de construção e busca local para o Problema de Clusterização de Bases de Dados. Tendências de Matemática Aplicada e Computacional, Vol. 7(1), pp. 109-118.

[17] Trindade, A. R., Ochi, L. S. (2006). Um algoritmo evolutivo híbrido para a formação de células de manufatura em sistemas de produção. PESQUISA OPERACIONAL, Vol. 26(2), pp. 255-294.

[18] Tseng, L. Y., Yang, S. B (2001). A genetic approach to the automatic clustering problem; Pattern Recognition 34, pp. 415- 424

[19] Zalik, K. R. (2008). An efficient k´-means clustering algorithm; Pattern Recognition Letters 29, pp. 1385-1391

[20] Yujian, L. (2006). A clustering algorithm based on maximal θ-distant subtrees, Pattern Recognition, pp. 1425-1431

[21] Wang, X., Qiu, W., Zamar, R. H. (2007). CLUES: A non-parametric clustering method based on local shrinking. Computational Statistics & Data Analysis 52, pp. 286-298.

[22] Welch, J. W. (1983). Algorithmic complexity: three NP-hard problems in computacional statistics. Journal of Statistical Computation and Simulation 15. pp. 17-25.