Learning and Nonlinear Models (L&NLM) – Journal of the Brazilian Neural Network Society, Vol. 8, Iss. 4, pp. 227-239, 2010.

UM ALGORITMO EVOLUTIVO COM MEMÓRIA ADAPTATIVA PARA O PROBLEMA DE CLUSTERIZAÇÃO AUTOMÁTICA

Marcelo Dib Cruz[1], Luiz Satoru Ochi[2]
[1]departamento de Matemática – Universidade Federal Rural do Rio de Janeiro (UFRRJ)
[2] Instituto de Computação – Universidade Federal Fluminense (UFF)
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 mais 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 de clusterização, 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 Evolutivo Híbrido para a solução do PCA, incluindo módulos de busca local e de memória adaptativa. Resultados computacionais realizados para um conjunto de instâncias mostram a eficiência e a robustez da heurística proposta.

Palavra Chave. Clusterização Automática, Algoritmos Evolutivos, 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 Hybrid Evolutionary Algorithm for ACP including 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, Evolutionary Algorithms, Adaptive 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 [30], mas o fato do valor de *k* não ser previamente conhecido torna o problema ainda mais complexo, aumentando substancialmente o número de soluções possíveis.

Existem várias aplicações relacionadas à clusterização incluindo: Particionamento de Grafos, Problema de Manufatura flexível, Reconhecimento de padrões onde se destaca Processamento de Imagens, Biologia Computacional, Pesquisa de Mercado, Classificação de Documentos, Mineração de Dados, entre outros. Devido à 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 [22] [25]. Neste contexto, acreditamos haver a necessidade de explorar melhor o potencial das metaheurísticas, procurando desenvolver versões mais eficientes de algoritmos heurísticos [2][3][4][14][18][20][21][23][24].

Neste contexto, o objetivo deste trabalho é a de propor uma heurística híbrida, denominada *Algoritmo Evolutivo com Memória Adaptativa (AEMA)*, para o PCA. Para isso, inicialmente é efetuado um pré-processamento para tentar reduzir as dimensões dos dados de entrada do problema, bem como gerar soluções iniciais com uma certa qualidade através de um algoritmo construtivo. Após esta primeira fase, é utilizado um Algoritmo Genético (AG) com módulos de Busca Local, cuja função é refinar uma parte das soluções encontradas pelo AG. O algoritmo proposto utiliza adicionalmente conceitos de memória adaptativa na estrutura de um conjunto de *soluções elite*, cuja função é armazenar as melhores soluções distintas geradas até o momento pelo AG. Finalmente, uma busca local exaustiva é utilizada tentando melhorar o *conjunto final de soluções elite* encontrado pelo AG.

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 Clusterização Automática (PCA)

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 grupos ou clusters, *k* não conhecido previamente, tal que 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:

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

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

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

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 (4):

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

Ás vezes podemos ter objetos de entrada que não são similares a nenhum outro objeto de *X*. Tais objetos são denominados *outliers* (ruídos) e ocorre usualmente por erro na coleta de dados, erro de digitação, fraudes, etc. Os *outliers* são uma dificuldade a mais na procura de soluções de boa qualidade num PCA.

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* [12], 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 [16] adapta o *k-means* para o PCA. É um algoritmo recursivo, que executa divisões binárias do espaço até que se chegue no melhor valor de k dentro de limites fornecidos. O X-means utiliza o índice BIC (*Bayesian Information Criterion*) para decidir qual o valor de k será retornado. O trabalho de [27] é mais recente mas também adapta o *k-Means* para resolver o PCA.

Alguns trabalhos encontrados na literatura utilizam algoritmos evolutivos como em [22] e [26], mas eles apresentam uma versão tradicional dos algoritmos genéticos (AGs) ou evolutivos (AEs), sem incluir módulos de busca local ou busca intensiva.

Os métodos propostos em [5] [6] 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 relativamente grande de objetos são candidatos potenciais para formarem um cluster. O resultado do método depende do tamanho das células, que normalmente é um parâmetro de entrada.

Um método apresentado por [28] 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] [29] 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 Evolutivo Proposto

O *Algoritmo Evolutivo com Memória Adaptativa (AEMA)* aqui proposto, é um método composto de duas fases: Fase de Construção e o Módulo Evolutivo. A fase de Construção tenta reduzir as dimensões dos dados de entrada do problema e ao mesmo tempo gerar soluções iniciais de boa qualidade. Nos Algoritmos Genéticos tradicionais isto não ocorre, pois a população inicial é normalmente gerada de forma aleatória. Estas duas metas da primeira fase são obtidas através de um algoritmo construtivo baseado em conceitos de *componentes conexas*. A segunda fase do AEMA é composta por um Algoritmo Genético com buscas locais e que utiliza conceitos de memória adaptativa, cujo objetivo é buscar a melhor configuração de uma solução possível. Para melhor entendimento do algoritmo, o AEMA é apresentado em partes, definidas como: *Fase de Construção, Representação da Solução, Função de Aptidão, Memória Adaptativa, busca local Inversão Individual, busca local Troca entre Pares, busca local Reconexão por Caminhos e Módulo Evolutivo.*

3.1. A Fase de Construção

A Fase de Construção do AEMA é uma etapa inicial que tem por objetivos tentar reduzir a cardinalidade dos dados de entrada do problema, bem como facilitar a geração de soluções iniciais de boa qualidade para o Algoritmo Genético. O procedimento é baseado no critério de densidade [8] [26]. 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.

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≠Ø)* ***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

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 (≤ *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).

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 na fase de construção, após o Procedimento Construtivo e o JCIA, são denominados clusters iniciais. Como a seqüência de análise dos clusters de menor cardinalidade são escolhidos de forma aleatória, ao final, uma população de soluções iniciais distintas, com diferentes números de clusters, podem ser geradas por este procedimento.

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*** *(* $\exists x_r \in B_i \; e \; \exists x_s \in B_k \; tal \; que \; || x_r - x_s || < d_{adj}$) ***ENTÃO***
8. *$B_k = B_k \; U \; B_i$*
9. ***FIM SE***
10. ***FIM SE***
11. ***FIM PARA***
12. *retornar B = {B_1,B_2,...B_m} os clusters iniciais, onde m <= t*
13. ***FIM JCIA***

Figura 2: Descrição do procedimento JCIA

3.2. Representação da Solução

Considere os clusters iniciais gerados na fase de construção 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 novo individuo (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 $B^o = \emptyset$.

3.3. Função de Aptidão

Para avaliar a qualidade da solução encontrada, é utilizada uma função de Aptidão F (equação 10), proposta por Kaufman e Rousseeum [11], conhecida como *média das Silhuetas* dos pontos do conjunto de entrada. 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,

$$b(x_i) = Min \;\; d(x_i, C_t) \;, \quad C_t \neq C_w \quad , \quad C_t \in C \;. \tag{8}$$

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 média das silhuetas, é definida como

$$F = \frac{1}{n}\sum_{i=1}^{n} s(x_i) \tag{10}$$

3.4. Memória Adaptativa

O uso de memória Adaptativa em metaheuristicas ainda não tem sido muito explorado principalmente em algoritmos evolutivos, mas tem se mostrado bastante promissor como descrito em [9] [15] [21]. Basicamente podemos definir este procedimento como uma forma de armazenar informações relevantes obtidas durante a execução de um algoritmo iterativo; para que tais informações possam ser utilizadas nas iterações remanescentes tentando com isso melhorar o processo de busca de uma solução ótima ou simplesmente uma solução de boa qualidade. Neste contexto, um caminho para tornar um algoritmo heurístico auto-adaptativo, pode ser implementado através do uso de algum tipo de memória que vai se atualizando no decorrer das execuções das iterações. As informações em si, podem estar relacionado ao processo de calibração de seus parâmetros e/ou uso de informações relevantes das melhores soluções geradas até este momento. 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 durante a execução das iterações do algoritmo proposto. Ou seja, a memória adaptativa utiliza um conjunto com as melhores soluções distintas geradas nas iterações passadas. Neste trabalho é utilizado um conjunto, denominado *CE* (Conjunto ELITE) de tamanho fixo, que armazena as melhores soluções. Quando uma nova solução de melhor qualidade for encontrada, esta substitui a pior solução contida no CE, mas lembrando que todas as soluções do *pool* devem ser distintas. O CE é utilizado em dois momentos distintos no algoritmo proposto: durante a execução do módulo evolutivo, para efetuar uma busca local por Reconexão por Caminhos (descrita na próxima sessão) e no final do algoritmo evolutivo, para efetuar uma busca local uma busca exaustiva.

3.5. As Buscas Locais

A finalidade de empregar busca local numa num algoritmo heurístico, e em particular num algoritmo genético (AG), é com o intuito de refinar as soluções obtidas pelo AG. Em testes empíricos efetuados, foi observado que as buscas locais tendem a melhorar bastante as soluções encontradas pelo genético. Uma busca local consiste basicamente em dado uma solução inicial, analisar sua vizinhança procurando por soluções de melhor qualidade; neste contexto, o que se faz, é buscar um ótimo local em problemas de otimização. A primeira busca local utilizada neste trabalho, denominada *Inversão Individual*, cuja função, é analisar soluções próximas às melhores soluções geradas pelo AG, permutando cada elemento do individuo. Outra busca promissora em problemas de otimização combinatória, é efetuar uma busca mais intensiva e próxima de soluções de boa qualidade. Para isso é utilizada a busca *Reconexão por Caminhos* ou *Path Relinking* descritas com detalhes na seção posterior.

3.5.1 A Busca Local Inversão Individual

A idéia básica desta busca, denominada *Inversão Individual*, é tentar melhorar a solução corrente analisando soluções próximas a ela. Para isso, essa busca permuta o valor de cada elemento do indivíduo (1 por 0, ou 0 por 1), um por vez, e calcula o novo valor da função de aptidão. Porém, o algoritmo só aceita a mudança, se o novo valor da função de aptidão for maior (melhor) que o valor anterior.

Por exemplo, imagine que o indivíduo corrente é *{0101101}*. Primeiramente, é trocado o primeiro elemento do individuo. Então o novo indivíduo é *{1101101}*. Se este indivíduo tem valor da função de aptidão maior que o anterior, então ele será o novo indivíduo corrente. E então, é permutado o status do segundo elemento. O novo indivíduo agora é *{1001101}*. Se este possuir valor da função aptidão maior que o anterior, então a solução corrente é atualizada. Caso contrario, o indivíduo corrente é mantido. A busca acaba quando todos os elementos do individuo forem testados.

Quando a busca é feita em um indivíduo e nenhuma melhora é alcançada, e ainda, o valor da função de aptidão é pior do que os valores contidos no CE, então o algoritmo descarta o individuo, e um novo indivíduo é gerado. Porém agora, este indivíduo não é gerado aleatoriamente no tocante ao número de clusters pais. Ele é gerado utilizando o *número de clusters pais da melhor solução encontrada até o momento* e priorizando os clusters de maior cardinalidade. Com isso estas novas soluções tenderão a ser *mais próximas* das melhores soluções existentes até o momento.

A busca local *Inversão Individual* se justifica, pois encontrar o número ideal de clusters é um dos objetivos do PCA, e a inclusão ou retirada de um cluster pai pode melhorar a solução corrente.

3.5.2. A Busca Local Troca Entre Pares

A busca local *Troca Entre Pares* é uma busca intensiva que troca o status de dois elementos do indivíduo com valores diferentes. Por exemplo, suponha que o indivíduo é *{10111010}*. Primeiro é trocado o primeiro e o segundo elemento do individuo. Então o novo indivíduo é *{01111010}*. Se este novo indivíduo melhorar o valor da função de aptidão, então ele será aceito e será a nova solução corrente. A próxima troca é feita entre o primeiro e o terceiro elemento sobre a solução corrente. A busca local termina quando todas as permutações entre dois genes com valores distintos são testadas.

A idéia desta busca local, ao contrário da anterior, é tentar encontrar indivíduos diferentes sem alterar o número de clusters pais encontrados pelas melhores soluções geradas pelo AG. Devido ao elevado tempo computacional exigido por este módulo, esta busca é feita somente ao final do algoritmo e somente nos três melhores indivíduos do CE final. O objetivo da busca local *Troca Entre Pares* é investigar possíveis soluções diferentes com o mesmo número de clusters pais.

3.5.3. A Busca Local Reconexão por Caminhos

A busca local Reconexão por Caminhos (RC), proposta inicialmente por F. Glover para as metaheuristicas *Tabu Search* e *Scatter Search* [9], tem como objetivo, dadas duas soluções extremas de boa qualidade geradas até o momento por uma heurística iterativa, encontrar soluções intermediárias de melhor qualidade. O princípio básico da RC é que, entre duas soluções de qualidade, pode existir uma terceira melhor que as soluções extremas.

A RC consiste em traçar o caminho entre uma *solução base* e uma *solução destino* (alvo) que apresentem boa qualidade e avaliar as soluções intermediárias obtidas ao longo do trajeto. O objetivo deste trajeto é encontrar soluções melhores que a base e a alvo.

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}). O objetivo da RC é inserir, a cada iteração, 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, e repetir este procedimento para todos os elementos que compõe uma solução. A cada permutação, uma nova solução candidata é gerada. Cada solução candidata é avaliada e a melhor (a que retorna a melhor solução) é 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. O processo é repetido até que a solução base seja igual a solução alvo. A figura 3 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}).

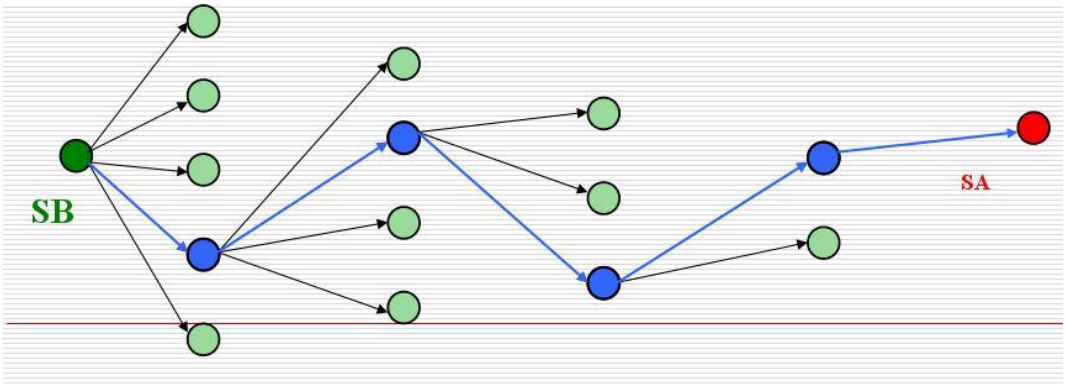

Figura 3: Descrição da RC.

Adicionalmente, outra justificativa para o uso da RC no PCA é o 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.

3.6. O Módulo Evolutivo

O módulo evolutivo é composto de um Algoritmo Genético Híbrido incluindo módulo de Memória Adaptativa e de três buscas locais. A racionalidade de utilizar Algoritmos Genéticos é gerar aleatoriamente um número qualquer de clusters pais. Como o número ideal de clusters é um dos objetivos do problema, o Algoritmo Genético permite gerar a cada iteração, números diferentes de clusters pais. Porém, o Algoritmo Genético (AG) na sua forma tradicional nem sempre consegue bons resultados, principalmente em problemas de Otimização Combinatória onde já existem heurísticas muito eficientes. Neste contexto, um caminho que tem se mostrado muito promissor [4] [14] [15] [19] [21] [23] [24], é a inclusão de módulos de busca local para tentar melhorar a qualidade das soluções geradas pelo AG. Neste trabalho, está se propondo a utilização das buscas locais *Inversão Individual, Troca entre Pares e Reconexâo por Caminhos.*

Para construir a população inicial, o algoritmo gera um número bem maior de indivíduos (dez vezes o tamanho da população) e escolhe aqueles com os maiores valores da função de aptidão. Este procedimento permite começar o AG com uma população de melhor qualidade.

No Algoritmo Genético, são utilizados operadores clássicos como o operador de mutação e proposto uma forma alternativa de seleção por cruzamentos.

Para efetuar a seleção dos indivíduos para cruzamento, são utilizados dois operadores que se alternam. O primeiro operador escolhe aleatoriamente os dois indivíduos pais dentre os 60% com melhores valores da função de aptidão. E o segundo operador, escolhe um indivíduo aleatoriamente dentre os 60% com melhores valores da função de aptidão e o outro,

entre os 40% restantes. O operador de cruzamento utilizado no AEMA é o cruzamento de dois pontos, que atua sobre dois indivíduos com genótipos diferentes. Este operador funciona da seguinte maneira: pares de pontos de cruzamento são obtidos de forma aleatória e os valores dos indivíduos localizados entre cada par de pontos de cruzamento são trocados. Os indivíduos são submetidos ao cruzamento com probabilidade p_c. Os dois pontos de corte definem os segmentos dos vetores que serão trocados entre os mesmos para gerar novos indivíduos.

O operador de mutação realiza trocas aleatórias de alguns valores dos indivíduos com probabilidade p_m, com o intuito de pesquisar novas áreas do espaço de busca, a partir de indivíduos selecionados.

Após a aplicação dos operadores de cruzamento e mutação, os descendentes que obtiverem valores da função de aptidão melhores que os valores da população atual são inseridos na nova população.

A cada t iterações (onde *t* é um parâmetro de entrada), os melhores indivíduos da população passam pela busca local *Inversão Individual*. O objetivo é intensificar a procura de soluções diferentes no conjunto de soluções existentes.

A cada iteração, a melhor solução é armazenada no conjunto elite (CE). A cada r iterações (onde *r* é um parâmetro de entrada), o melhor indivíduo da população e o melhor indivíduo do CE, passam pelo busca local *Reconexão por Caminhos*.

No final das iterações do AG, o CE passa pela busca local *Troca entre Pares*.

4. Resultados Computacionais

Para analisar o desempenho do algoritmo AEMA aqui proposto, testes computacionais foram realizados. O AEMA foi comparado com um dos algoritmos mais recentes da literatura, denominado CLUES, que foi publicado em 2007 pela revista *Computacional Statistics & Data Analysis* [29]. 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 AEMA 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[17], Iris Plants Database [7], MaronnaDataSet[13], 200DATA[7], VowelDataset[10] e Broken Ring[29]. 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 [22]. 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.

Em relação aos parâmetros utilizados no algoritmo AEMA, 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. A taxa de cruzamento dos indivíduos de cada geração foi fixada em 80% do tamanho da população, e a taxa de mutação foi fixada em 10% do tamanho da população. Portanto, as probabilidades de cruzamento P_c e mutação P_m foram fixadas em 0.20 e 0.90. O tamanho da população escolhido foi de 1/3 do tamanho do indivíduo com um valor máximo de 20(vinte) indivíduos. O tamanho do conjunto elite (CE) foi fixado com 5(cinco) elementos. O critério de parada do algoritmo é o número máximo de iterações, fixado como 50, valor este definido à partir de testes preliminares. O valor de t, que indica a periodicidade da busca local Inversão Individual foi fixado em 5 iterações. Esta busca é realizada somente nos 3 melhores elementos da população. A busca local Reconexão por Caminhos é executada 4 vezes, nas iterações 18, 28 , 38, e 48. Para a execução do CLUES, não é necessária a especificação de parâmetros.

Tabela 1: Tabela de Comparação do AEMA com/sem as buscas locais (BL) e Memória Adaptativa (MA)

	AEMA com as BL e MA						AEMA sem as BL e MA				
Instância	**Melhor**	**Média**	**t(s)**	**Iter**	**NC**	**Desvio**	**Média**	**t(s)**	**Iter**	**NC**	**Desvio**
RuspiniDataSet	0,7376	0,7376	0.8	50	4	**0,00**	0,7376	0.8	117	4	**0,00**
200DATA	0,8231	0,8231	4.3	50	3	**0,00**	0,8231	4.3	138	3	**0,00**
300p6c1	0,6636	0,6636	5.4	50	8	**0,00**	0,6525	5.4	286	9	1,67
400p4c1	0,5989	0,5989	6.2	50	4	**0,00**	0,5753	6.2	272	4	4,11
500p6c1	0,6287	0,6287	8.5	50	6	**0,00**	0,6287	8.5	264	6	**0,00**
600p15c	0,7812	0,7812	32.7	50	15	**0,00**	0,7107	32.7	507	17	9,02
700p15c1	0,6804	0,6804	23.4	50	15	**0,00**	0,6314	23.4	457	19	7,20
1000p14c	0,8306	0,8306	84.7	50	14	**0,00**	0,7568	84.7	278	18	8,89
1500p6c	0,6941	0,6941	214.7	50	6	**0,00**	0,6774	214.7	482	5	2,41
2000p9c1	0,623	0,623	344.2	50	9	**0,00**	0,5382	344.2	578	14	13,61

Inicialmente, para verificar o potencial das buscas locais e da memória adaptativa propostas para o algoritmo genético, o AEMA foi executado para um conjunto reduzido de instâncias, com e sem as buscas locais e a memória adaptativa, utilizando o mesmo tempo de execução como critério de parada. O tempo de execução utilizado é o tempo necessário para executar 50 iterações no algoritmo AEMA com as Buscas Locais (BL) e a Memória Adaptativa (MA). O resultado das simulações é mostrado na Tabela 1.

Os algoritmos foram executados 5(cinco) vezes para cada instância. As colunas têm os seguintes significados: a coluna *Instância* contém os nomes das instâncias. A coluna *Melhor* contém o maior valor da função Índice Silhueta encontrado pelos dois algoritmos. A coluna *Média* contém a média dos valores da função Índice Silhueta que 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. A coluna I*ter* contém a média dos números de iterações (ou gerações) que cada algoritmo executou, 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 partir destes resultados verifica-se a importância de se incluir procedimentos eficientes de buscas locais e memória adaptativa no algoritmo proposto. De fato, em todas as instâncias o AEMA com as BL e MA conseguiu os melhores valores, tendo desvio de 0.00% em todas as instâncias. O AEMA sem as BL e MA encontrou os melhores valores em apenas 3 das 10 instâncias executadas, executando um numero muito maior de iterações. Em outras 7 instâncias, mesmo executando um número bem maior de iterações, esta versão não alcançou as melhores soluções, tendo desvio médio entre 1,67% e 13,61%. Embora as buscas locais aumentem o tempo de processamento por iteração, verificamos que esta versão com BL e MA necessita de um número menor de iterações para atingir os melhores valores.

A Tabela 2 mostra a comparação entre o CLUES e o AEMA. Os algoritmos foram executados 5(cinco) vezes para cada instância. As colunas têm os mesmos significados da Tabela 1.

Na comparação do AEMA com o CLUES, a diferença entre os desvios médios é grande, de 1.31 para 7.07, significando que o AEMA está mais próximo das melhores soluções. Além de possuir um desvio médio 5(cinco) vezes menor, o AEMA também alcança os melhores valores em uma quantidade maior de instâncias. O AEMA 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 é mais rápido, com uma média de 27,29s contra 53,03s do AEMA. Avaliando os tempos, o que se observa é que com instâncias menores, com até 800 pontos (de RuspiniDataSet até 800p23c), em 39 instâncias de um total de 53, os algoritmos têm tempos semelhantes, obtendo médias próximas, variando de 11.19s do AEMA para 11.70s do CLUES, alternando quem tem o menor tempo. Porém, em instâncias maiores, o CLUES consegue os tempos bem menores.

Tabela 2: Tabela de Comparação dos algoritmos AEMA e CLUES

		AEMA				CLUES			
Instância	**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**
Iris Plants Datset	0,6862	0,6862	2,7	3	**0,00**	0,5626	1,3	3	18,01
MaronnaDataSet	0,5745	0,5745	2,8	4	**0,00**	0,5745	4,2	4	**0,00**
200data	0,8231	0,8231	4,3	3	**0,00**	0,4622	2,8	3	43,85
VowelDataset	0,4483	0,4246	15,4	27	5,29	0,4483	26,8	9	**0,00**
Broken Ring	0,4995	0,4995	39,6	5	**0,00**	0,4995	31,8	5	**0,00**
100p2c1	0,7427	0,7427	1,8	2	**0,00**	0,5213	3,7	5	29,81
100p3c	0,7858	0,7858	2,2	3	**0,00**	0,7858	1,6	3	**0,00**
100p3c1	0,5966	0,5802	2,4	3	2,75	0,5966	1,8	3	**0,00**
100p5c1	0,7034	0,6958	1,6	7	1,08	0,7034	3,1	6	**0,00**
100p7c	0,8338	0,8338	1,8	7	**0,00**	0,8338	3,7	7	**0,00**
100p7c1	0,5511	0,4911	1,9	7	10,89	0,5511	4,1	7	**0,00**
100p10c	0,8336	0,8336	1,8	10	**0,00**	0,8336	2,7	10	**0,00**
200p2c1	0,7642	0,7642	2,3	2	**0,00**	0,5912	3,1	3	22,64
200p3c1	0,6797	0,6797	3,1	3	**0,00**	0,6740	3,9	3	0,84
200p4c	0,7725	0,7725	3,1	4	**0,00**	0,7725	3,6	4	**0,00**
200p4c1	0,7544	0,7449	3,2	4	1,26	0,7544	3,4	4	**0,00**
200p7c1	0,5759	0,5759	2,7	13	**0,00**	0,5553	8,8	9	3,58
200p12c1	0,5753	0,5753	2,8	13	**0,00**	0,5624	8,5	9	2,24
300p2c1	0,7764	0,7764	5,1	2	**0,00**	0,4899	4,7	5	36,90
300p3c	0,7663	0,7663	7,2	3	**0,00**	0,5520	9,2	3	27,97
300p3c1	0,6768	0,6768	6,4	3	**0,00**	0,5924	4,4	4	12,47
300p4c1	0,5910	0,5910	5,7	2	**0,00**	0,5563	4,6	7	5,87
300p6c1	0,6636	0,6636	5,4	8	**0,00**	0,5788	11,9	7	12,78
300p13c1	0,5994	0,5644	5,3	13	5,84	0,5994	10,8	9	**0,00**
400p3c	0,7985	0,7985	9,1	3	**0,00**	0,7985	9,6	3	**0,00**
400p4c1	0,6204	0,5989	6,2	4	3,47	0,6204	12,8	4	**0,00**
400p17c1	0,5524	0,5138	10,6	2	6,99	0,5524	20,2	15	**0,00**
500p3c	0,8249	0,8249	9,5	3	**0,00**	0,8249	10,1	3	**0,00**
500p4c1	0,6595	0,6595	8,1	5	**0,00**	0,5150	9,9	3	21,91
500p6c1	0,6684	0,6287	8,5	6	5,94	0,6684	16,7	6	**0,00**
600p3c1	0,7209	0,7209	18,3	3	**0,00**	0,7028	9,6	3	2,51
600p15c	0,7812	0,7812	32,7	15	**0,00**	0,7526	37,1	16	3,66
700p4c	0,7969	0,7969	31,5	4	**0,00**	0,7969	12,9	4	**0,00**
700p15c1	0,6804	0,6804	23,4	15	**0,00**	0,6595	22,9	17	3,07
800p4c1	0,7143	0,7021	38,6	4	1,71	0,7143	15,9	4	**0,00**
800p10c1	0,5071	0,4681	34,7	2	7,69	0,5071	29,7	8	**0,00**
800p18c1	0,6941	0,6914	24,9	19	0,39	0,6941	42.2	19	**0,00**
800p23c	0,7873	0,7873	55,4	23	**0,00**	0,7387	47.7	22	6,17
900p5c	0,7160	0,7160	71,2	5	**0,00**	0,6129	24,4	7	14,40
900p12c	0,8408	0,8408	70,8	12	**0,00**	0,7975	63.1	10	5,15
1000p5c1	0,6391	0,6391	71,5	5	**0,00**	0,5580	38.0	7	12,69
1000p6c	0,7356	0,7356	76,7	6	**0,00**	0,7356	38.3	6	**0,00**
1000p14c	0,8306	0,8306	84,7	14	**0,00**	0,7674	51.6	11	7,61
1000p27c1	0,5631	0,5186	112,3	25	7,90	0,5631	67.4	14	**0,00**
1100p6c1	0,6847	0,6717	91,5	6	1,90	0,6847	39.9	6	**0,00**
1300p17c	0,8229	0,8229	121,3	17	**0,00**	0,7379	80.4	15	10,33
1500p6c	0,6941	0,6941	214,7	6	**0,00**	0,6845	73.6	5	1,38
1500p6c1	0,6597	0,6436	205,6	6	2,44	0,6597	61.7	6	**0,00**
1500p20c	0,8232	0,8232	243,5	20	**0,00**	0,6874	92.9	13	16,50
1800p22c	0,8036	0,8036	305,1	22	**0,00**	0,6433	136.1	18	19,95
2000p9c1	0,6230	0,6230	344,2	9	**0,00**	0,5354	123.2	7	14,06
2000p11c	0,7129	0,7129	354,7	11	**0,00**	0,6062	93.6	6	14,97
Media			53.03s		1,31		27.29s		7,07

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 AEMA 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 AEMA 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 AEMA.

Uma observação a ser feita, é que o mais razoável seria também utilizarmos testes usando outros critérios de parada como por exemplo um tempo limite, ou o alcance de um valor alvo, mas isso não foi possível pois o CLUES é executado como uma "caixa preta", não podendo alterar, para cima ou para baixo, o numero de iterações e, conseqüentemente, o seu tempo de execução.

5. Conclusão

Este trabalho aborda o Problema de Clusterização Automática - PCA. Foi desenvolvido um Algoritmo Evolutivo híbrido que utiliza buscas locais e Memória Adaptativa para melhorar a qualidade das soluções obtidas. Foi constatado que a utilização de um processamento inicial (Fase de Construção) otimizado melhora bastante o desempenho final do Módulo Evolutivo, uma vez que este módulo inicial na média já reduz significativamente o número de objetos a serem agrupados, além de efetuar um pré-agrupamento de parte dos objetos. Outro fator importante para o bom desempenho do AEMA foram as buscas locais Elas se mostraram muito eficientes em relação ao desempenho dos Algoritmos Genéticos tradicionais sem busca local. Embora aumentem o tempo de processamento por iteração, verificamos que estas versões de AG necessitam em contrapartida de um número menor de iterações para atingir um determinado valor alvo. Finalmente, os resultados do AEMA comparados com um algoritmo da literatura mostram a eficiência do algoritmo proposto, uma vez que conseguiu resultados melhores na maioria das instâncias analisadas.

Como trabalho futuro, sugerimos implementar outras metaheurísticas com memória adaptativa para o PCA como por exemplo, o GRASP e o Iterative Local Search (ILS) que tem se mostrado bastante competitivas na solução de diferentes problemas de otimização combinatória [4], [14], [15], [24].

Agradecimentos: *Os autores agradecem ao apoio parcial dos seguintes órgãos de fomento: CNPq: 550740/2007-4 (CT-INFO); 474268/2007-1 (UNIVERSAL); 302833/2009-9 (Bolsa de Produtividade); CAPES: 23038.031353/2008-95 (Pró-Engenharia); FAPERJ: E-26/110.358/2007 (PENSA RIO); E-26/102.853/2008 (Cientista do Estado do Rio de Janeiro).*

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] Bastos, L. O., and Ochi, L. S. (2008). A Genetic Algorithm with Evolutionary Path-Relinking for the Sonet Ring Assignment Problem. Proc.of the International Conference on Engineering Optimization (EngOpt2008).

[3] Boeres, C., Rios, E., and Ochi, L. S. (2005). Hybrid Evolutionary Static Scheduling for Heterogeneous Systems. Proc. of the IEEE Conference on Evolutionary Computation (IEEE - CEC 2005), Book 3, pp. 1929-1937, Edinburgh, 2005.

[4] 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).

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

[6] 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

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

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

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

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

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

[12] 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

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

[14] Mine, M. T., Silva, M. S. A., Souza, M. J. F., Ochi, L. S. (2010). O problema de roteamento de veículos com coleta e entrega simultânea: Uma abordagem via Iterated Local Search e GENIUS". Capítulo 3 do Livro: Transporte em Transformação Volume XIV, pg. 61-80, ISBN: 978-85-99082-10-2. Publicado pela Confederação Nacional do Transporte (CNT) em conjunto com o ANPET – Associação Nacional de Pesquisa e Ensino em Transportes.

[15] Pailla, A., Parada, V., Trindade, A. R., and Ochi, L. S. (2010). A numerical comparison between simulated annealing and evolutionary approaches to the cell formation problem". Expert Systems with Application - ELSEVIER – Volume 37, pp. 5076-5083.

[16] 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.

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

[18] 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

[19] Santos, H. G., Uchoa, E., Ochi, L. S.,Maculan, N. (2010). Strong Bounds with Cut and Column Generation for Class-Teacher Timetabling. To appear in Annals of Operations Research (Special Issue PATAT2008) – Springer.

[20] Silva, A. R. V. and Ochi, L. S. (2007). A hybrid evolutionary algorithm for the dynamic resource constrained task scheduling problem. Proc. of the 10th International Workshop on Nature Inspired Distributed Computing (NIDISC' 07) held in conjunction with Parallel and Distributed Processing Symposium (IPDPS 2007), USA.

[21] 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

[22] 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. TEMA-Tendências de Matemática Aplicada e Computacional, Vol. 7(1), pp. 109-118.

[23] Souza, M. J. F., Mine, M. T., Silva, M. S. A., Ochi, L. S., Subramanian, A. (2010). A hybrid heuristic, based on Iterated Local Search and GENIUS, for the Vehicle Routing Problem with Simultaneous Pickup and Delivery**.** To appear in International Journal of Logistics Systems Management (IJLSM) based on selected ICIL 2010 papers. Inderscience Publishers.

[24] Subramanian, A., Drummond, L. M. A., Ochi, L. S., Bentes, C., Fariasn, R. (2010). A Parallel Metaheuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery. In Computers & Operations Research – ELSEVIER , Volume 37(11), pp. 1899-1911.

[25] 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.

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

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

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

[29] 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.

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